Empowering Runtime Verification with Polyhedra

by

Masaki Waga

Dissertation

submitted to the Department of Informatics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

The Graduate University for Advanced Studies, SOKENDAI
September 2020

Committee

Advisor	Dr. Ichiro Hasuo Associate professor of the Graduate University for Advanced Studies (SOKENDAI)/National Institute of Informatics
Subadvisor	Dr. Kensuke Fukuda Associate professor of the Graduate University for Advanced Studies (SOKENDAI)/National Institute of Informatics
Subadvisor	Dr. Fuyuki Ishikawa Associate professor of National Institute of Informatics
Examiner	Dr. Jaco van de Pol Professor of Aarhus University/University of Twente
Examiner	Dr. Tomohiro Yoneda Professor of the Graduate University for Advanced Studies (SOKENDAI)/National Institute of Informatics

Acknowledgements

My Ph.D. study has been an exciting and fantastic experience, supported by too many people to list up here. First and foremost, I would like to thank my supervisor Ichiro Hasuo. His teaching, encouragement, advice, help, arrangement, and many other supports were obviously needed for finishing this thesis. He opened my eyes to the world of automata theory and formal methods. His extensive knowledge, experience, and vision helped me a lot in pursuing my Ph.D. research. My gratitude extends to the thesis committee members Kensuke Fukuda, Fuyuki Ishikawa, Jaco van de Pol, and Tomohiro Yoneda for their comments from various perspectives. I would also like to thank my M.Sc. supervisor Masami Hagiya for a lot of supports during and after my M.Sc. study.

It was a lot of pleasure to collaborate with various researchers. I am particularly grateful for the opportunity to work with Étienne André during his stay in Tokyo and my visit to Nancy. He led me to the theory of parametric timed automata that resulted in our contribution to runtime verification with parametric timed automata. I would give a special thanks to Takamasa Okudono, Taro Sekiyama, Elena Gutiérrez, Kohei Suenaga, and Takumi Akazaki for the fruitful and inspiring discussion through the collaboration with them.

It was also a great pleasure to work with the colleagues in the ERATO MMSD project. Special thanks go to Sasinee Pruekprasert, Zhenya Zhang, Paolo Arcaini, and Stefan Klikovits for a lot of fruitful discussions. I am also thankful to the industrial collaborators for the comments and discussion on the practical relevance of my research.

Finally, I would like to thank my family, especially my parents Masanobu and Takako, for everything.

Masaki Waga
Tokyo, July 2020

Abstract

This thesis aims to improve the practical effectivity of runtime verification, monitoring with logics. Our main application is the safety monitoring of cyber-physical systems (CPSs), e. g., cars and robots. Our technical gadget is the symbolic analysis with convex polyhedra. The high-level contribution of this thesis is to show that the polyhedra-based symbolic analysis plays an essential role in various advanced runtime verification algorithms.

Monitoring is, in general, an activity to observe system behavior. In the development and maintenance of systems, it is essential to monitor the system behavior. For example, in development, engineers have to monitor the system behavior to identify the necessary modification to the system under development. In maintenance, for example, engineers have to monitor the running system so that they can replace the system's worn-out components as soon as possible.

Runtime verification (or specification-based monitoring) is an automated monitoring technique using logic. Given a formal specification of an unsafe behavior expressed by some logical formalism, runtime verification observes a system execution and evaluates if the observation satisfies the specification.

Although quite a lot of research and engineering efforts have been devoted to runtime verification of CPSs, there are remaining challenges. We identify the following important but often missing features of runtime verification. Generic algorithms that are applicable to a wide class of runtime verification problems rather than one problem setting Flexible runtime verification algorithm that does not require complete knowledge of the specifications or behaviors Informative results such as quantitative satisfaction rather than Boolean satisfaction

In this thesis, we present enhanced runtime verification algorithms focusing on the three features above. In our improvements, the use of polyhedra for symbolic analysis plays an essential role. In the symbolic analysis, we utilize discrete abstraction of the continuous value domains represented by polyhedra: each of which stands for infinitely many concrete values; and thus, we can analyze infinitely many values. This is in contrast to the analysis of each value, where only finitely many values can be analyzed in finite time.

Such a polyhedra-based analysis is useful, for example, in the runtime verification with an ambiguous specification. Consider the following specification: "whenever the gear of a car becomes low, the gear should remain low for a while," where the definition of "for a while" is unclear. Moreover, the threshold defining "for a while" may depend on the context. When the specification contains such an unspecified threshold, we have to monitor the log considering all the possible thresholds. Since there are infinitely many possibilities, we cannot try each threshold in a one-by-one manner, and we need a polyhedra-based symbolic analysis.

This thesis's high-level contribution is to show the usefulness of the polyhedra-based analysis in runtime verification. To show the usefulness, we conducted three concrete improvements. The following summarizes the usages of polyhedra and the enhanced features in this thesis.

Firstly, in Chapters 3 and 4, we study runtime verification with ambiguous specifications containing unknown thresholds in the constraints. Such a runtime verification algorithm does not require complete knowledge of the monitored specification and is flexible. We use polyhedra for symbolic analysis of infinitely many possible thresholds. In Chapter 3, we introduce and solve the parametric timed pattern matching problem, where we can use a specification with timing parameters to leave some thresholds in the timing constraints unspecified. For example, in the example above, we can represent the timing constraint "for a while" by "for p seconds" using a timing parameter p. In Chapter 4, we generalize the parametric timed pattern matching problem to allow the parameters also in data values. For example, consider the following specification: "whenever the temperature becomes high, the air conditioner must be turned on within 5 seconds," where the high-temperature threshold is unspecified. In this example, we can represent the condition on high-temperature by "more than T degree," where T is a data parameter representing the high-temperature threshold. Moreover, our algorithm is generic because it allows any data with a suitable data structure for symbolic analysis such as polyhedra for rationals and an ad hoc data structure for strings.

Secondly, in Chapter 5, we study quantitative timed pattern matching that is a mathematical formulation of quantitative runtime verification of real-valued signals. Quantitative timed pattern matching returns the degree of unsafety and is more informative than returning Boolean results. We propose an online algorithm for quantitative timed pattern matching that can monitor a running system. Our notion of unsafe degree and our proposed algorithm are based on semiring valued weighted automata. Thanks to the algebraic genericity of semirings, our algorithm works for various quantitative semantics capturing different safety criteria such as the worst deviation from the specification and the accumulated deviation from the threshold over time. We use polyhedra to obtain discrete abstraction of the continuous possibility of switching in a temporal specification. Consider the specification "in the beginning, the acceleration of the car is high, and later, the velocity becomes high." When monitoring such a temporal specification, we have to consider all the possible timing of the switching from the "beginning" to the "later." Since there are continuously many possibilities, we utilize polyhedra-based symbolic analysis to consider all such switching.

Thirdly, in Chapter 6, we study runtime verification, where we only have intermittent samples of the signal values. We introduce and solve the model-bounded monitoring problem, where we interpolate the signal values between the samples considering the bounding model. Thanks to the bounding model, model-bounded monitoring is precise even if we reduce the sampling frequency, and thus it is flexible. More precisely, if the bounding model overapproximates the actual system behavior, model-bounded monitoring is guaranteed to detect every unsafe behavior independent of the sampling frequency. Although we may have false alarms, we have fewer false alarms for a more precise bounding model. We use polyhedra to consider all the possible interpolation.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

This thesis aims to improve the practical applicability and efficiency of *runtime verification* [BFFR18, BF18], especially of real-time properties. Our technical vehicle is *polyhedra* for symbolic analysis of continuous space. The high-level contribution of this thesis is to show that the polyhedra-based symbolic analysis plays an essential role in various advanced runtime verification algorithms.

The main application of the technical results in this thesis is the monitoring of *cyber-physical systems* (*CPSs*) [BDD+18], e. g., cars, robots, smart-cities, and medical systems. Among them, the monitoring of cars or robots is of our main interest. Nevertheless, the application of our methods is not limited to monitoring of CPSs but also includes monitoring of both purely software- or hardware-oriented systems.

1.1 System monitoring

Monitoring[1] [BDD+18] is, in general, an activity to observe system behavior. Both in the development and maintenance of systems, it is essential to monitor the system behavior. For example, in development, engineers modify a system to satisfy requirements. To identify the necessary modification, they have to monitor the difference between the actual system behavior and the expected behavior. In maintenance, engineers have to replace the worn-out components in the system. This decision also requires monitoring because they have to detect if the current system behavior deviates from the original behavior.

The following two examples show more concrete usage scenarios of monitoring.

Example 1.1 (continuous monitoring of a deployed system)**.** Consider a scenario in which we developed a factory automation system, and we have just deployed it to a factory. Just after the deployment, the system performs pretty well, e. g., there is no overshoot, and the system behaves robustly. But as time passes, the system gradually becomes erroneous. For instance, a slight overshoot might happen due to the wearing out of some hardware. Such an issue might

[1] In the literature, the terms monitoring and runtime verification are often used interchangeably. In this thesis, following [BDD+18], we use monitoring for the general activity while we use runtime verification and specification-based monitoring for the monitoring with formal specifications.

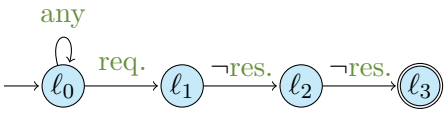


Figure 1.1: An automaton accepting the observation such that there is no response within two time steps after the request

stop the system; therefore, we need continuous monitoring of the system to detect and deal with an issue as soon as possible. ◇

Example 1.2 (monitoring as a test oracle)**.** Consider a scenario in which we develop a new engine from scratch, and we have several prototypes either as a simulator or a hardware. Because it is at the very beginning of the development, each prototype most likely contains issues, and we have to figure out the problem, e. g., by testing. In testing, we need an oracle to decide if the behavior of the system under test is erroneous or not; this is also monitoring. ◇

1.2 Primitive forms of monitoring

One of the most primitive forms of monitoring is by a human: given a specification in a document, an engineer checks if the system observation meets the specification. Manual monitoring is ubiquitous in system development. For example, it is natural to check if the system under development satisfies the given requirement through manual trials.

Although manual monitoring is easily applicable and ubiquitous in system development, it has a big issue, namely its poor scalability. The amount of the available observation tends to be huge in CPS development, and the scalability of monitoring must be high. For a human, it is very challenging to monitor massive system observation. Another related issue is that manual monitoring is usually costly because human resources are expensive.

In order to overcome the scalability issue, we can automate monitoring by implementing a dedicated software or hardware. These automated monitors are necessary, especially for continuous monitoring of a running system, e. g., indicators of cars or airplanes. It is, however, challenging to handcraft a correct monitor for a complicated specification.

1.3 Runtime verification — monitoring with logics

Runtime verification (or *specification-based* monitoring) is a monitoring method using logics. Given a formal specification typically of an unsafe behavior expressed in some logical formalism, e. g., automata, temporal logic formalism, or regular expressions, runtime verification observes a system execution and evaluates if the observation satisfies the specification. Since these logical specifications can be automatically translated into a monitor, we can automate the monitoring task and solve the scalability issue without the effort of monitor construction.

For discrete systems such as software or digital hardware, a system execution is a sequence of states or events at each step, and the specification represents the set of sequence of our interest. For example, the automaton in Fig. 1.1 accepts a sequence such that the server does not respond within two time steps after the request, which is undesired.

This high-level picture of runtime verification is the same for CPSs. However, due to the continuous nature of CPSs, more continuous formalisms are usually used for observation and specification. For example, the system behavior in Fig. 1.4a is a sequence of events equipped with continuous timestamps rather than discrete time steps.

1.4 Towards generic, flexible, and informative runtime verification

In order to cope with the aforementioned continuous nature of CPSs, quite a lot of research and engineering effort have been devoted. For example, after the introduction of signal temporal logic [MN04] to monitor continuous-time real-valued signals, various extensions have been proposed, e. g., an extension to compare with previous signal value using an additional operator called *freezing operator* [BDSV14], an extension to monitor the frequency domain using *Fourier transform* [DMB+12], and an extension to handle properties on *spaces* [NBC+18]. See [BDD+18] for a survey. Nevertheless, there are remaining challenges in runtime verification of CPSs. We identify the following important but often missing features of the runtime verification problems and the algorithms.

Genericity One issue in the current runtime verification is that we have to construct one algorithm for each problem setting. For instance, when we monitor a log with numeric data, we cannot reuse a runtime verification algorithm for strings. A *generic* runtime verification algorithm works for a class of problems rather than one problem and reduces the effort to give a new algorithm and implementation for a new problem setting.

Flexibility In runtime verification, it is a usual assumption to have complete knowledge of the specification and the system observation (i. e., log). However, these assumptions are not always realistic. One issue is that, for the system under development, it is often the case that the specification is not precisely determined. Notably, it is more challenging to determine the exact threshold in the specification than ambiguously stating, e. g., "the velocity should not be too large" or "the system should respond reasonably soon." Another issue is that, in monitoring with an embedded system, we want to increase the sampling interval to reduce energy consumption, which also reduces the available information for the monitor. *Flexible* runtime verification algorithms that relax these issues improve the practicality of runtime verification.

Informativeness It is also beneficial to return *more information* than Boolean satisfaction. For example, by returning a *quantitative* safety degree (or *robustness*), one can compare and differentiate the safety of two system logs, even if both satisfy the given specification. Such quantitative monitoring also plays an essential role in search-based safety assurance, e. g., *falsification* [Don10, ALFS11] of CPSs.

1.5 High-level contribution: advanced runtime verification with polyhedra

In this thesis, we present enhanced runtime verification algorithms focusing on the three features in Section 1.4. This thesis's high-level contribution is to show the usefulness of the *polyhedra-based analysis* in runtime verification through the concrete improvements.

In our improvements, the use of *polyhedra* for symbolic analysis plays an essential role. See Fig. 1.2 for an illustration of the symbolic analysis with polyhedra. In the symbolic analysis, we utilize discrete abstraction of the continuous value domains represented by polyhedra: each of which stands for infinitely many concrete values; and thus, we can analyze infinitely many values. For example, in the right of Fig. 1.2, the three polyhedra represent all the points in the

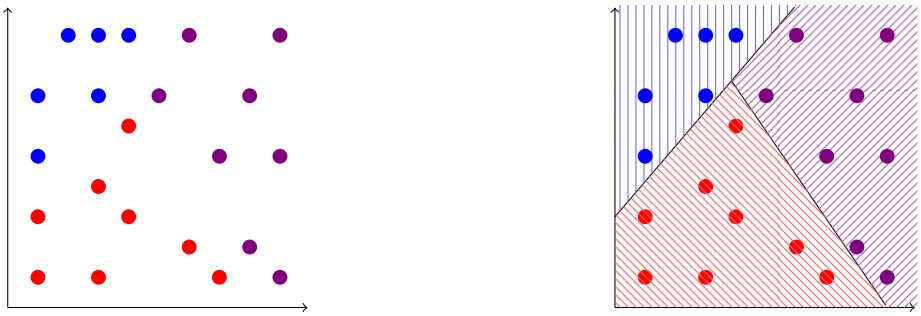


Figure 1.2: Concrete points in a continuous 2-dimensional value domain (left) and their polyhedra-based abstraction (right). By using the polyhedra-based abstraction (right), we can symbolically analyze continuously many points in each area while by evaluating the concrete points (left), we can evaluate only finite points.

continuous value domain, and we can analyze all the points by the symbolic analysis of these three polyhedra. This is in contrast to the analysis of each value, where only finitely many values can be analyzed in finite time.

Such a polyhedra-based analysis is useful, for example, in runtime verification with an ambiguous specification. Consider the following specification: "whenever the gear of a car becomes low, the gear should remain low for a while," where the definition of "for a while" is unclear. Moreover, the threshold defining "for a while" may depend on the context. When the specification contains such an unspecified threshold, we have to monitor the log considering all the possible thresholds. Since there are infinitely many possibilities, we cannot try each threshold in a one-by-one manner, and we need a polyhedra-based symbolic analysis.

We note that polyhedra-based symbolic analysis is also an essential ingredient of the reachability checking, a fundamental technique for exhaustive verification, of various kinds of automata e. g., timed automata [AD94], linear hybrid automata [HPR94], and parametric timed automata [AHV93, ACEF09]. Our high-level contribution is to employ the techniques originally developed for exhaustive verification to improve the practicality of runtime verification. See Section 1.7.1 for a comparison between exhaustive verification and runtime verification.

1.6 Highlight of the improvements

In this section, we summarize the concrete contributions of each chapter focusing on the application rather than the technical background. Fig. 1.3 shows the relationships among the contributions.

Since monitoring of cars is an important application, we have at least one vehicle monitoring example for each subsection. Using various automotive examples, we show various features of our runtime verification algorithms. In addition to the automotive examples, we also use many other examples from various domains to show the applicability to other domains, e. g., monitoring of other CPSs or purely software-oriented systems.

1.6.1 Parametric timed pattern matching

In Chapter 3, we extend the *timed pattern matching problem* [UFAM14, WAH16] with parameters. This enables us to write a specification without deciding the thresholds in timing constraints.

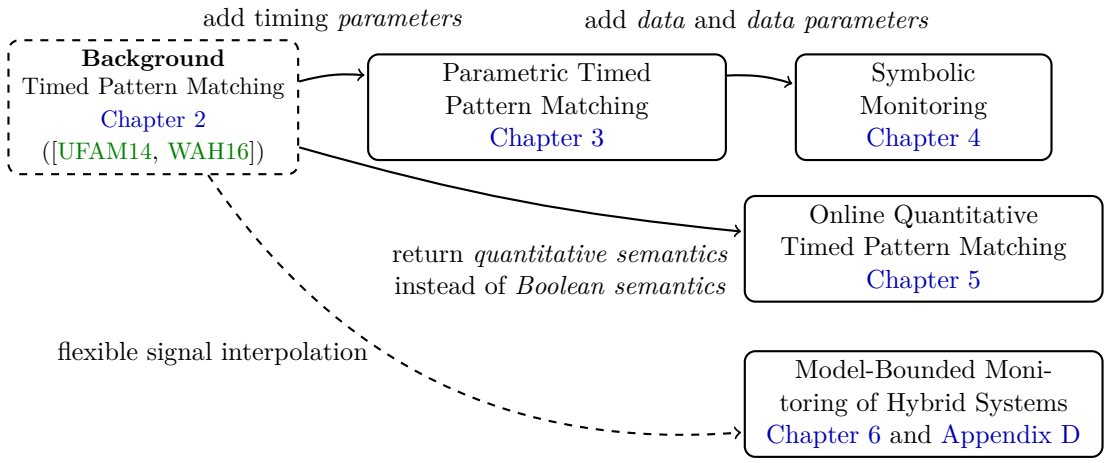
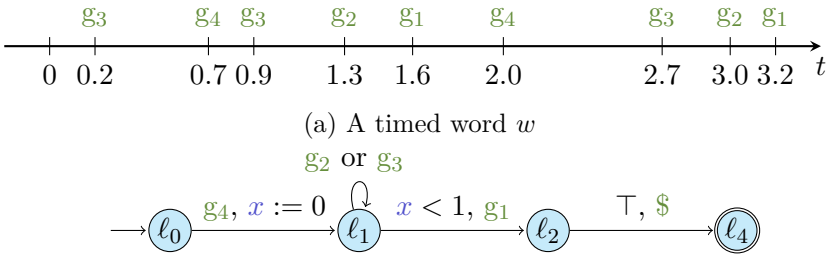


Figure 1.3: Relationships among the contributions. Parametric timed pattern matching (Chapter 3) and quantitative timed pattern matching (Chapter 5) generalizes timed pattern matching [UFAM14, WAH16]. Symbolic monitoring (Chapter 4) generalizes parametric timed pattern matching. Model-bounded monitoring (Chapter 6) is not a direct generalization of timed pattern matching; however, the generalization in Appendix D allows timed pattern matching with flexible signal interpolation.



(a) A timed word w

(b) A timed automaton $\mathcal{A}$

Figure 1.4: An example of timed pattern matching

1.6.1.1 Motivation: Specification with unspecified thresholds

Timed pattern matching *Timed pattern matching* [UFAM14] is a mathematical formulation of runtime verification. Given a system log and a specification, timed pattern matching answers in which part of the log, the specification is satisfied. Typically, the specification represents an unsafe behavior. In this case, timed pattern matching finds when such an unsafe behavior occurs in the given log.

After the introduction in [UFAM14], the timed pattern matching problem has been studied in various problem settings with technical improvements (e. g., [UFAM14, UFAM16, WAH16, WHS17]). In [WAH16], the timed pattern matching problem is defined with *timed words* and *timed automata (TAs)* [AD94]. A timed word is a sequence of events with timestamps. A TA is a *nondeterministic finite automaton (NFA)* equipped with clock variables to represent the timing constraints on timed words.

Example 1.3 (gear changes of a car)**.** Consider the example in Fig. 1.4. The timed word w in Fig. 1.4a is the log of the gear changes of a car. The objective of the runtime verification with the TA $\mathcal{A}$ in Fig. 1.4b is to find too frequent gear changes. Roughly speaking, $\mathcal{A}$ matches if the gear changes from g_4 to g_1 within 1 time unit, which is too soon and undesirable. On the

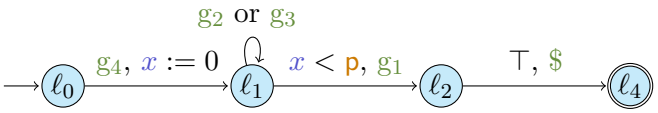


Figure 1.5: The PTA $\mathcal{A}$ such that the timing constraint $x < 1$ in Fig. 1.4b is replaced with a parametric timing constraint $x < \mathsf{p}$

one hand, because we have g_4 at 0.7 and g_1 at 1.6 in w, $\mathcal{A}$ matches around $[0.7, 1.6]$. On the other hand, although we have g_4 at 2.0 and g_1 at 3.2 in w, $\mathcal{A}$ does not match around $[2.0, 3.2]$ because the time difference is longer than 1.0.

Formally, timed pattern matching returns the set of matching intervals $\{(t, t') \mid w|_{(t,t')} \in \mathcal{L}(\mathcal{A})\} = \{(t, t') \mid t \in [0.2, 0.7), t' \in (1.6, 2.0]\}$, which is the set of open intervals (t, t') such that the restriction $w|_{(t,t')}$ of the timed word w in (t, t') is accepted by the TA $\mathcal{A}$. Here the notion "around $[0.7, 1.6]$" is formalized as the open intervals containing $[0.7, 1.6]$ but not including any other events i. e., not including either 0.2 or 2.0. We note that \$ in the TA $\mathcal{A}$ is the special terminal character. ◇

Issue: Difficulty in writing specifications In practice, it is often challenging to specify a concrete specification. Notably, it tends to be hard to determine the exact threshold in the specification. For example, for an automatic transmission system of a car, we do not want to have too frequent gear changes, and the following is a reasonable requirement: "the gear should not change *too soon* after the latest gear change." When we give a specification in a TA, we have to specify the concrete threshold to determine what it means for a gear change to be "*too soon*." It is, however, not easy to give such a threshold because it requires an in-depth domain knowledge of the developed system. This issue can be a significant obstacle when using timed pattern matching in practice.

1.6.1.2 Contribution: Pattern matching with parametric specifications

Parametric timed pattern matching Our solution to the issue above is to use *parameters for unspecified thresholds*. More precisely, we use *parametric timed automata (PTAs)* [AHV93] for parametric specifications. Fig. 1.5 shows an example of a PTA. A PTA is a TA such that we can use *parameters* as well as the constant numbers in the timing constraints. For instance, by using the parameter p for the unspecified threshold, we can represent "the gear should not change *for* p *seconds* after the latest gear change" in a PTA.

Parametric timed pattern matching is a generalization of timed pattern matching where the specification is given in a PTA $\mathcal{A}$ rather than a TA. Given a timed word w and a PTA $\mathcal{A}$, parametric timed pattern matching returns in which part of the timed word w and for what parameter valuation v assigning a real-value to the parameters, the parametric specification represented by the PTA $\mathcal{A}$ is satisfied.

Example 1.4 (gear changes of a car)**.** Consider the timed word w in Fig. 1.4a and the PTA $\mathcal{A}$ in Fig. 1.5. Here, the PTA $\mathcal{A}$ matches if the gear changes from g_4 to g_1 within p time unit. Because we have g_4 at 0.7 and g_1 at 1.6 in w, $\mathcal{A}$ matches around $[0.7, 1.6]$ for $\mathsf{p} > 0.9$. Because we have g_4 at 2.0 and g_1 at 3.2 in w, $\mathcal{A}$ matches around $[2.0, 3.2]$ for $\mathsf{p} > 1.2$.

Formally, the output of parametric timed pattern matching is as follows, where (t, t') is a

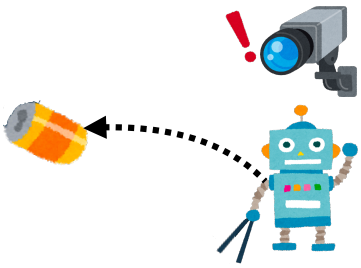


Figure 1.6: An automated cleaning system. Once the camera finds trash, it notifies a robot cleaner to remove the trash.

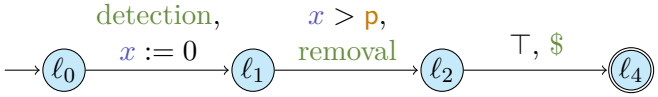


Figure 1.7: A PTA matching too late removal of the trash after its detection

matching interval and v is one of the corresponding parameter valuations.

$$\begin{aligned}\{(t,t',v) \mid w|_{(t,t')} \in \mathcal{L}(v(\mathcal{A}))\} =& \{(t,t',v) \mid t \in [0.2, 0.7), t' \in (1.6, 2.0], v(\mathsf{p}) > 0.9\} \\ \cup& \{(t,t',v) \mid t \in [1.6, 2.0), t' \in (3.2, \infty), v(\mathsf{p}) > 1.2\}\end{aligned}$$

◇

Example 1.5 (automated cleaning system)**.** Consider an automated cleaning system in Fig. 1.6. There are cameras in a public space, and once a camera finds trash, it notifies a robot cleaner to remove the trash. One reasonable specification is that the robot should remove the trash as soon as the camera finds the trash. However, it is hard to define the exact threshold determining if the robot is too late because it depends on various factors, e. g., the size of the public space, the number of the working robots, and how crowded it is.

By using a PTA, we can represent this specification as in Fig. 1.7. By parametric timed pattern matching, for each trash, we obtain how long it took to remove the trash after the detection of it. We note that it is also possible to ignore clearly safe behaviors by a giving a global constraint of the parameter p. ◇

Algorithms and implementations We give two algorithms for the parametric timed pattern matching: an algorithm via model checking and a dedicated algorithm. Both methods are similar in the sense that they utilize the finite abstraction of the clock and parameter valuations by *polyhedra*. The latter dedicated algorithm is optimized by using *skipping* [FJS07] that is originally from string matching.

For the algorithm via model checking, we used IMITATOR, which is a model checking tool for PTAs. For the dedicated algorithm, we implemented a prototype tool ParamMONAA. Our experimental evaluation shows the efficiency of our methods especially for ParamMONAA. For example, ParamMONAA monitors about 35,000 entries per second for a realistic benchmark called Gear and about 120,000 entries per second for a realistic benchmark called Accel 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). This is as fast as CAN bus or hundreds of high-speed sensors.

1.6.1.3 Summary

Parametric timed pattern matching is a parametric extension of timed pattern matching with parameters in the timing constraints. Our contribution to parametric timed pattern matching is summarized as follows.

- We introduce the problem of parametric timed pattern matching.
- We give two approaches to solve the parametric timed pattern matching problem.
- Our experimental result shows the efficiency of our results for some benchmarks.

We used polyhedra to obtain a finite abstraction of the clock and the parameter valuations. Our contribution makes timed pattern matching more flexible and informative as follows.

Flexibility Using parameters, we can use timing constraints with unspecified thresholds in the specification. This allows us to give a specification without precisely defining timing constraints, which tends to be challenging in practice.

Informativeness By using parametric timed pattern matching, we can synthesize how robustly the timing constraints are satisfied (or violated). This is more informative than returning Boolean satisfaction. The idea is to enlarge the timing constraints using parameters representing the robustness of the satisfaction. For example, by replacing the guard $x < 1$ of Fig. 1.4b with $x < 1 + \mathsf{p}_x$, where p_x is a parameter, we can synthesize the robust satisfaction degree of $x < 1$.

The material in Chapter 3 is based on the joint work [AHW18, WA19] with Étienne André and Ichiro Hasuo. Useful comments from the anonymous referees of these papers are gratefully acknowledged.

1.6.2 Symbolic monitoring against specifications parametric in time and data

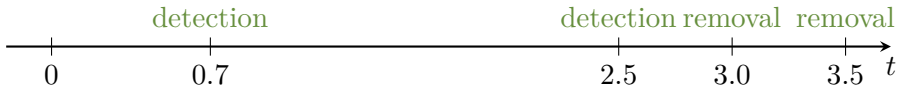
1.6.2.1 Motivation: Beyond finite domain events

By parametric timed pattern matching in Section 1.6.1, we can conduct monitoring even if the timing constraints in the specification contains unspecified thresholds represented by parameters. The use of PTAs instead of TAs is the essence of parametric timed pattern matching.

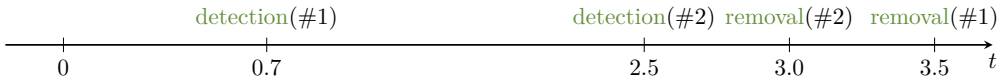
In parametric timed pattern matching, the monitored log is a timed word, which is the same as timed pattern matching. A timed word is a sequence of events with timestamps, where the events are from a *finite* domain. This finiteness of the events is often too restrictive to represent system behavior with *infinite* or *unbounded* nature.

Example 1.6 (automated cleaning system)**.** Consider again the scenario in Example 1.5. In Example 1.5, we abstract the observed actions into two events: detection and removal. This abstraction is too coarse when we have multiple pairs of detection and removal. Consider the following observation.

- The camera observed trash (say #1) at 0.7.
- The camera observed another trash (say #2) at 2.5.



(a) The timed word over the alphabet {detection, removal} modeling the observation in Example 1.6.



(b) The timed data word modeling the observation in Example 1.6. The events (detection or removal) is equipped with an identifying string (#1 or #2).

Figure 1.8: The timed word and the timed data word modeling the observation in Example 1.6 of the automatic cleaning system

```
1 @1.5  deposit  100
2 @3.8  withdraw  50
3 @4.2  withdraw  30
4 @6.7  withdraw  80
5 @8.9  deposit  30
```

Figure 1.9: Example transactions of a bank account

- The cleaning robot removed #2 at 3.0.
- The cleaning robot removed #1 at 3.5.

Fig. 1.8a shows the timed word modeling this observation using the events in Example 1.5. The issue here is that the information on the correspondence between the detection and the removal is abstracted away, and we cannot monitor it as we would like. One possible solution is to equip an identifying string for each event. The technical issue is that the number of such strings is *unbounded* due to the unbounded number of observed trash. ◇

Example 1.7 (bank account)**.** Consider the transactions of a bank account shown in Fig. 1.9. We can model it as a timed word by abstracting each transaction to either withdraw or deposit. However, this abstraction is too coarse because the amount is essential in the modeling of the transaction; without the amount of the deposit or withdrawal, we cannot monitor, e. g., too much decrease of the balance in a short period. The amount of each withdrawal or deposit is a number, which is also from an infinite domain. ◇

Overall, timed words are often not expressive enough, and we need to equip each event with data from some infinite domain.

1.6.2.2 Contribution: Symbolic monitoring of infinite domain data

Symbolic monitoring against parametric timed data automata To overcome the issue above on the expressive power of timed words, we introduce the *symbolic monitoring* of infinite domain data. In symbolic monitoring, the log is given by a *timed data word*, and the specification is given by a *parametric timed data automaton (PTDA)*.

A timed data word is a timed word such that each event is equipped with infinite domain data as well as a timestamp. For example, Fig. 1.8b shows an example of the timed data word for the automated clearing system; here, #1 and #2 are the strings to identify each trash.

PTDA is an extension of PTA with infinite domain data. PTDA can represent constraints on data using data variables, data guards, and variable updates. We note that we can use

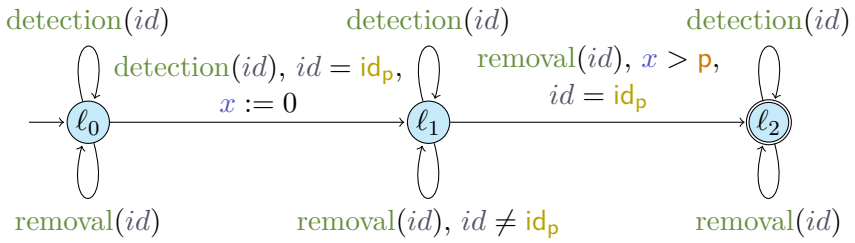


Figure 1.10: A PTDA accepting a timed data word containing a removal of trash that is more than p time units after the detection, where $\mathsf{id_p}$ is the data parameter for the identifying string of the trash, and p is the timing parameter.

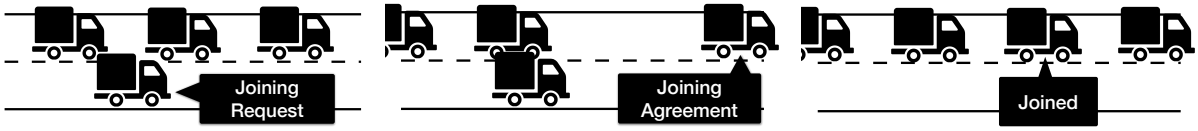


Figure 1.11: A joining process of a vehicle platoon. When a non-member truck sends a joining request (left), the member trucks increase the inter-vehicle distance so that the non-member truck can join. Once the inter-vehicle distance becomes large enough, one of the member truck sends the agreement to the joining truck (center), and the truck joins the platoon (right).

parameters for data as well as the timing constraints. For example, Fig. 1.10 shows an example of the PTDA for the automated clearing system with timing parameter p and data parameter $\mathsf{id_p}$. This PTDA accepts a timed data word containing a removal of trash that is more than p time units after the detection.

Given a timed data word and a PTDA, symbolic monitoring returns the set of parameter valuations such that the PTDA accepts the timed data word.

Example 1.8 (automated cleaning system)**.** Continue the scenario in Example 1.6. The observation in Example 1.6 contains two pairs of the detection and the removal of trash; the first trash (say #1) is detected at 0.7 and removed at 3.5; the second trash (say #2) is detected at 2.5 and removed at 3.0. We alert if the removal of the trash $\mathsf{id_p}$ is more than p time unit after the detection, where $\mathsf{id_p}$ is the string-valued data parameter to identify the trash and p is the timing parameter for the timing constraint. For the above observation, we alert for $\mathsf{id_p} = \#1 \wedge \mathsf{p} < 2.8$ and $\mathsf{id_p} = \#2 \wedge \mathsf{p} < 0.5$.

Formally, these observation and specification are represented by the timed data word w in Fig. 1.8b and the PTDA $\mathcal{A}$ in Fig. 1.10, respectively. The result of the symbolic monitoring for w and $\mathcal{A}$ is the following set of parameter valuations v, where $v(\mathsf{id_p})$ is a string and $v(\mathsf{p})$ is a rational number.

$$\{v \mid v(\mathsf{id_p}) = \#1, v(\mathsf{p}) < 2.8)\} \cup \{v \mid v(\mathsf{id_p}) = \#2, v(\mathsf{p}) < 0.5)\}$$

◇

Example 1.9 (vehicle platooning)**.** Consider monitoring of a *vehicle platooning* [KDM+17] system. Vehicle platooning is a style of self-driving vehicles such that a group of vehicles drives in a line. Thanks to the inter-vehicle communication, vehicle platooning can reduce the inter-vehicle distance and the brake usage, and thus, vehicle platooning improves the efficiency of the road usage and the energy consumption.

Our goal is to monitor a joining process to a platoon shown in Fig. 1.11 and check if the acceptance of the joining is too late. In the log in Fig. 1.12, we observe that the acceptance

```
1  @0.5    request   #A   #2
2  @4.8    accept    #A
3  @8.5    request   #B   #4
4  @20.3   accept    #B
```

Figure 1.12: Example communication in joining processes of a vehicle platoon. The request event with two strings id and $position$ is for the joining request by the vehicle id at $position$. The accept event with one string id is for the acceptance of the joining by the vehicle id. We need infinite domain data for the identification because the number of the vehicles in a platoon is unbounded.

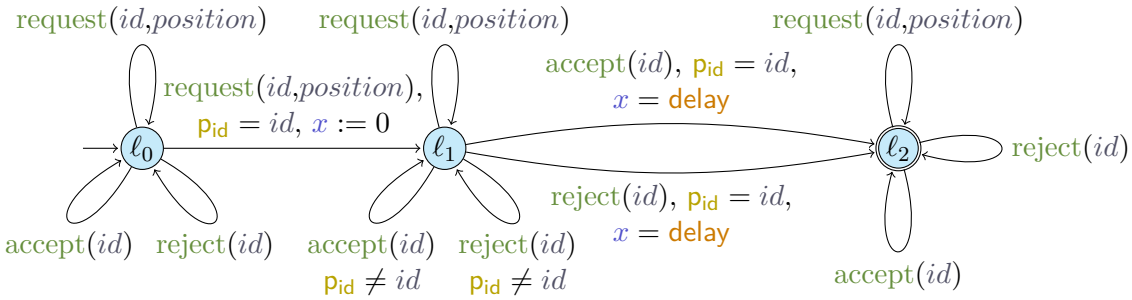


Figure 1.13: A PTDA to detect too late acceptance or rejection of the joining request to platoon, where x is the clock variable for the duration after the request, delay is the timing parameter for the delay of the response, and p_{id} is the string-valued data parameter to identify the joining vehicle

of the vehicle #B takes much longer time than #A. We can use the PTDA $\mathcal{A}$ in Fig. 1.13 to detect such delay of the acceptance. The result of the symbolic monitoring is as follows.

$$\{v \mid (v(\mathsf{p}_{\mathsf{id}}), v(\mathsf{delay})) = (\#\mathrm{A}, 4.3) \text{ or } (\#\mathrm{B}, 11.8)\}$$

◇

In principle, symbolic monitoring does not answer where unsafe behavior is detected, but it only answers the acceptance with the corresponding parameter valuations. However, we can locate in which part of the log the unsafe behavior occurs by using timing parameters. In Example 1.10, we show a simpler pattern matching problem than (parametric) timed pattern matching. See Section 4.4.3 for the encoding of parametric timed pattern matching with symbolic monitoring.

Example 1.10 (bank account)**.** Continue the scenario in Example 1.7. Our goal is to locate in which part of the log, the balance decreases for more than 100 in total. In the log in Fig. 1.9, we observe that such a decrease happens between 3.8 and 6.7, and between 3.8 and 8.9.

We can use the PTDA $\mathcal{A}$ in Fig. 1.14 to locate such a decrease. The PTDA $\mathcal{A}$ contains two timing parameters t and t' for the interval $[\mathsf{t}, \mathsf{t}']$ where such a decrease happens. In $\mathcal{A}$, we take the summation of the amount of the withdrawals and deposits between t and t'. The result of the symbolic monitoring is as follows.

$$\{v \mid (v(\mathsf{t}), v(\mathsf{t}')) = (3.8, 6.7) \text{ or } (3.8, 8.9)\}$$

◇

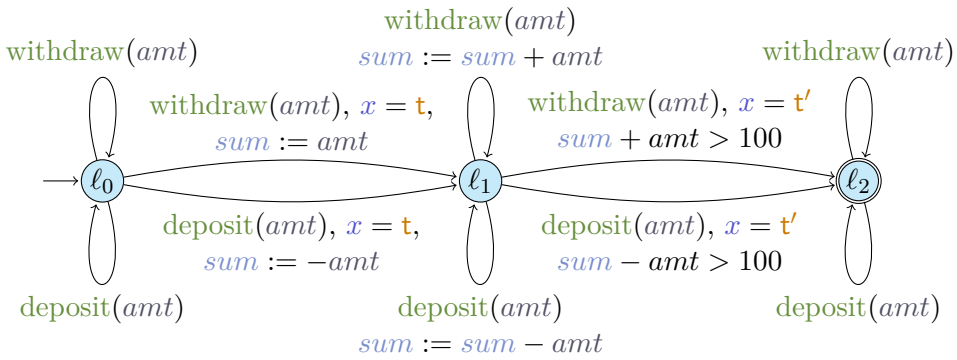


Figure 1.14: A PTDA to locate the interval $[\mathsf{t}, \mathsf{t}']$ in the log where the balance decreases for more than 100 in total, where x is the clock variable for the global time, t and t' are timing parameters, and sum is the variable to contain how much the balance decreases.

Algorithms and implementations We give an algorithm to solve the symbolic monitoring problem. Thanks to the finite abstraction of the time and data valuations, e. g., by convex polyhedra for rational-valued data and a list-based ad hoc data structure for string-valued data, the main body of the algorithm is a breadth-first search.

We implemented a tool SyMon for symbolic monitoring. Our experimental evaluation shows the efficiency of our method. For example, SyMon monitors about 130 entries per second for a realistic benchmark called DOMINANT 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). This is fast enough for online monitoring using 10 sensors with sampling interval 100ms.

1.6.2.3 Summary

Symbolic monitoring is a generalization of parametric timed pattern matching with data from some infinite domain. Our contribution to symbolic monitoring is summarized as follows.

- We introduce symbolic monitoring as well as its input formalisms.
- We give an algorithm for symbolic monitoring.
- Our experimental result shows the efficiency of our algorithm.

We used polyhedra to obtain a finite abstraction of rational-valued data and parameter valuation as well as clock and timing parameter valuations. Our contribution is generic, flexible, and informative.

Genericity Our symbolic monitoring algorithm works for any "data" from domain that offers suitable operations. For example, our algorithm works for both rationals and strings.

Flexibility We can use data parameters as well as timing parameters we used in Section 1.6.1. In this sense, the specification in symbolic monitoring is more flexible.

Informativeness Similarly to Section 1.6.1, we can synthesize the safety degree using the parameters. Moreover, we can extract timing and data information from the log. For example, in Example 1.10, by monitoring the log, we extracted the intervals of the withdrawals for each range of amounts.

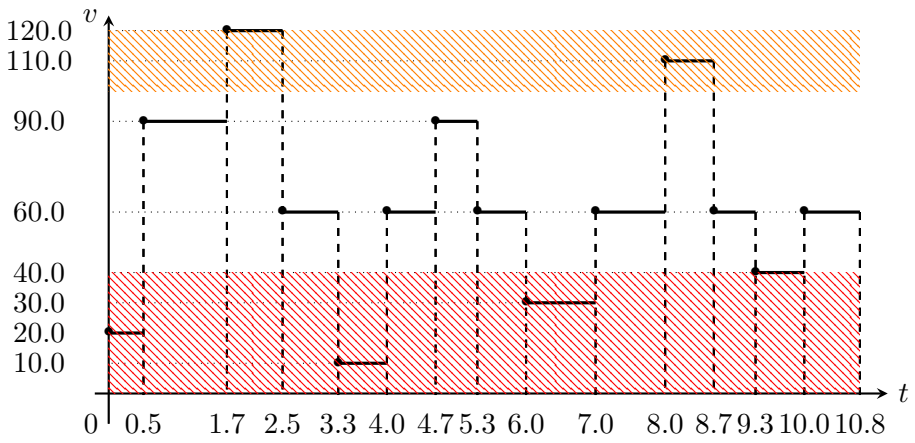


Figure 1.15: A signal σ showing the velocity of a car. A subsignal is unsafe if: 1) it starts in the red area; 2) it ends in the orange area; and 3) its duration is shorter than 3.0.

The material in Chapter 4 is based on joint work [WAH19] with Étienne André and Ichiro Hasuo. Useful comments from the anonymous referees of the paper are gratefully acknowledged.

1.6.3 Online quantitative timed pattern matching with semiring-valued weighted automata

1.6.3.1 Motivation: Online algorithm for quantitative timed pattern matching

More informative timed pattern matching Consider monitoring of a driving record of a self-driving car. For example, we locate in which part of the driving record, the acceleration is too sudden. More precisely, we alert if the velocity v changes from $v < 40$ to $v > 100$ within 3 time units. For the signal of the velocity in Fig. 1.15, such a pattern appears in $[0, 2.5)$ and $[6.0, 8.7)$. Timed pattern matching locates such areas.

However, we observe that this result is not informative enough to tell the following: the acceleration is more sudden in $[0, 2.5)$ than in $[6.0, 8.7)$; the acceleration in $[3.3, 5.3)$ is more sudden than in $[9.3, 10.8)$, although both of them are not too sudden according to the specification. In order to give a more fine-grained and *informative* answer, for each area, we want the *degree of unsafety* rather than the *Boolean verdict* of unsafety.

Quantitative timed pattern matching [BFMU17] is a quantitative extension of timed pattern matching returning the quantitative semantics representing the satisfaction degree of the given specification rather than the Boolean satisfaction. For each subsignal, quantitative timed pattern matching returns how robustly the given specification is satisfied or violated. When the specification represents unsafe behavior, we obtain the degree of unsafety for each subsignal.

Example 1.11. For the signal σ in Fig. 1.15, the quantitative semantics $\alpha(\sigma([0, 2.5)))$ for the subsignal $\sigma([0, 2.5))$ in $[0, 2.5)$ is 20 because the inequality constraints ($v < 40$ and $v > 100$) are satisfied even if we shift the signal values up or down, by a gap up to 20. See Fig. 1.16 for an illustration. Because the quantitative semantics $\alpha(\sigma([6.0, 8.7)))$ for the subsignal $\sigma([6.0, 8.7))$ is 10, $\sigma([0, 2.5))$ satisfies the specification more robustly than $\sigma([6.0, 8.7))$. Since the specification represents unsafe behavior, we conclude that σ is safer in $[6.0, 8.7)$ than in $[0, 2.5)$. Similarly, the quantitative semantics $\alpha(\sigma([3.3, 5.3)))$ and $\alpha(\sigma([9.3, 10.8)))$ for the subsignals $\sigma([3.3, 5.3))$ and

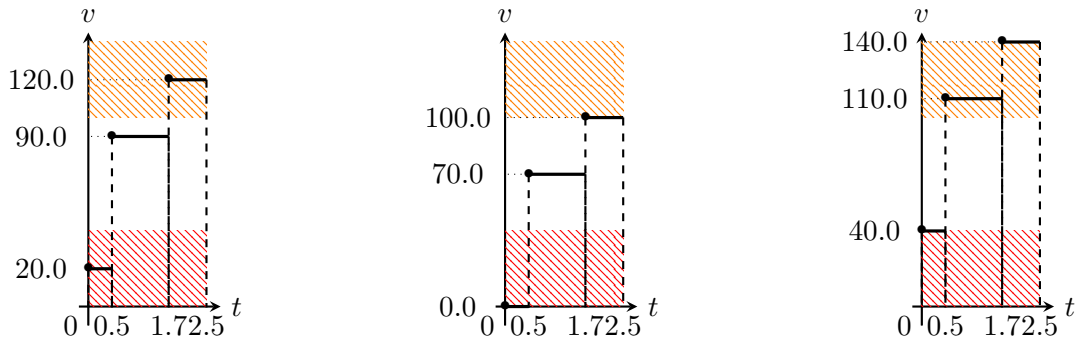


Figure 1.16: An illustration of the quantitative semantics. A signal satisfies the specification if it starts in the red area and ends in the orange area. Since this specification represents unsafe behavior, the quantitative semantics shows an unsafety degree. The quantitative semantics of the specification for the original signal (left) is 20 because it keeps satisfying the specification even after decreasing (center) or increasing (right) the value up to 20.

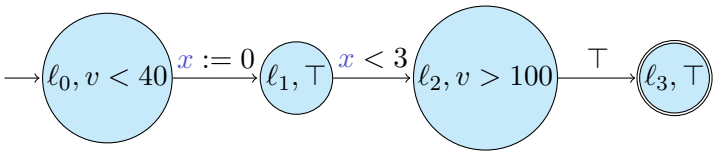


Figure 1.17: The TSWA showing that the velocity v changes from $v < 40$ to $v > 100$ within 3 time units. More precisely, at location ℓ_0 and ℓ_2, we require the constraints $v < 40$ and $v > 100$, respectively; we have to enter ℓ_2 less than 3 time units after we leave ℓ_0. The transition from ℓ_2 to ℓ_3 shows that we have no constraint on the time elapse in ℓ_2.

$\sigma([9.3, 10.8))$ are -10 and -40, respectively; therefore, $\sigma([9.3, 10.8))$ is safer than $\sigma([3.3, 5.3))$. ◇

Issue: Online monitoring capability In [BFMU17], an *offline* algorithm for quantitative timed pattern matching is introduced. This algorithm requires the entirety of the monitored signal to start monitoring. Such a monitoring algorithm is *offline* monitoring capable: for a log of previous system execution, this algorithm computes the quantitative semantics for each subsignal. However, such a monitoring algorithm is not *online* monitoring capable: for a stream of a log of the running system, this algorithm cannot compute the quantitative semantics along with the system execution. This limitation is critical for some usage scenarios. For example, an offline monitoring algorithm cannot give feedback to a running system using the monitoring result before the execution is complete. The introduction of an online algorithm is practically valuable because it expands the application area.

1.6.3.2 Contribution: Quantitative timed pattern matching with weighted automata

Online algorithm with automata We introduce an online algorithm for quantitative timed pattern matching. Our algorithm utilizes *weighted automata*, which is a quantitative extension of NFAs. The use of automata makes an online algorithm simpler.

By combining the concepts from both timed automata and weighted automata, we introduce *timed symbolic weighted automata (TSWAs)*. Fig. 1.17 shows an example of a TSWA. In

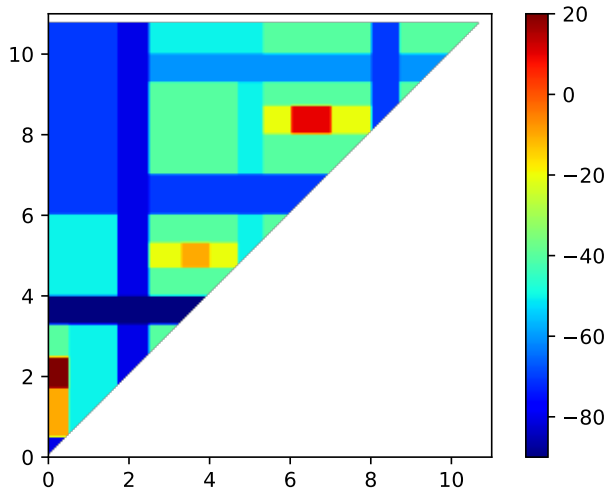


Figure 1.18: An illustration of the result of the quantitative timed pattern matching. Each point (t, t') shows the quantitative semantics $\alpha(\sigma([t, t']))$ for the subsignal $\sigma([t, t'))$ of the input signal σ in the interval $[t, t')$. For example, the value at $(3.3, 5.3)$ is -10, which shows that the quantitative semantics $\alpha(\sigma([3.3, 5.3)))$ for the subsignal $\sigma([3.3, 5.3))$ in the interval $[3.3, 5.3)$ is -10.

a TSWA, each location has a constraint on the signal values. The other structure of TSWAs is the same as TAs.

Example 1.12 (velocity monitoring)**.** Continue the scenario in Example 1.11. Let σ be the signal shown in Fig. 1.15. The TSWA in Fig. 1.17 represents the specification "the velocity v changes from $v < 40$ to $v > 100$ within 3 time units". We note that the constraints at locations ℓ_1 and ℓ_3 are $\top$, and we do not care the signal values there.

For the subsignal $\sigma([0, 2.5))$ of σ in $[0, 2.5)$, the quantitative semantics $\alpha(\sigma([0, 2.5)))$ is 20 because of Eq. (1.1), where $\sigma(0)$ and $\sigma(1.7)$ are the signal values at 0 and 1.7, respectively, and 40 and 100 are the thresholds in the specification. This follows the intuition in Fig. 1.16.

$$\alpha(\sigma([0, 2.5))) = \min\{(40 - \sigma(0)), (\sigma(1.7) - 100)\} = \min\{(40 - 20), (120 - 100)\} = 20 \quad (1.1)$$

Quantitative timed pattern matching answers such a quantitative semantics for each subsignal. Fig. 1.18 illustrates the result of quantitative timed pattern matching, where the color at point (t, t') shows the quantitative semantics $\alpha(\sigma([t, t']))$ for the subsignal $\sigma([t, t'))$. ◇

Genericity via semirings We note that our contribution is not only to provide an online algorithm but also to generalize the notion of quantitative semantics. We utilize an algebraic structure called semiring in our algorithm, and our algorithm works for various quantitative semantics. For example, we can take the summation of the difference from the threshold rather than taking the minimum value as in Eq. (1.1). Mathematically, this is to use the *sup-plus* semiring instead of the *sup-inf* semiring.

The preference between these semantics depends on the usage scenario. For example, as shown in Example 1.13, when we use the quantitative semantics $\alpha_{\text{sup-+}}$ with sup-plus semiring, the "safe" values compensate for the "unsafe". This semantics may be an option, e. g., for costs or energy consumption, but not suitable, e. g., for physical distance because we can cover the debt with revenue while very long inter-vehicular distance cannot compensate for a collision of

cars. Nevertheless, the quantitative semantics by the sup-inf semantics would be the standard semantics thanks to the intuition in Fig. 1.16.

Example 1.13 (quantitative semantics by summation)**.** Continue the scenario in Example 1.12. Here, we use the quantitative semantics $\alpha_{\sup\text{-}+}$ defined by sup-plus semiring instead of sup-inf semiring. For the subsignals $\sigma([0, 2.5))$, $\sigma([3.3, 5.3))$, $\sigma([6.0, 8.7))$, and $\sigma([9.3, 10.8))$, the quantitative semantics are as follows.

$$\begin{aligned}
\alpha_{\sup\text{-}+}(\sigma([0, 2.5))) &= (40 - 20) + (120 - 100) = 40\\
\alpha_{\sup\text{-}+}(\sigma([3.3, 5.3))) &= (40 - 10) + (90 - 100) = 20\\
\alpha_{\sup\text{-}+}(\sigma([6.0, 8.7))) &= (40 - 30) + (110 - 100) = 20\\
\alpha_{\sup\text{-}+}(\sigma([9.3, 10.8))) &= (40 - 40) + (60 - 100) = -40
\end{aligned}$$

For the quantitative semantics defined by sup-plus semiring, we have $\alpha_{\sup\text{-}+}(\sigma([3.3, 5.3))) = \alpha_{\sup\text{-}+}(\sigma([6.0, 8.7)))$ while for the quantitative semantics defined by sup-inf semiring, $\alpha(\sigma([6.0, 8.7)))$ is larger than $\alpha(\sigma([3.3, 5.3)))$. This inconsistency is because, in the quantitative semantics $\alpha_{\sup\text{-}+}$ by sup-plus semiring, small deviation from the specification by the velocity ($v = 10$) at the beginning of $\sigma([6.0, 8.7))$ affects the quantitative semantics, while in the quantitative semantics α by sup-inf semiring, large deviation by the velocity ($v = 90$) masks it. ◇

Generic algorithm and implementation We give an online algorithm for quantitative timed pattern matching using the shortest distance over (semiring-valued) weighted graphs. Our algorithm works for any semiring satisfying a certain condition. For example, our algorithm captures both of the quantitative semantics above.

We used polyhedra-based abstraction when reducing the size of the weighted graph to finite. In signal monitoring, there are continuously many possible timings of the switching in a temporal specification due to the continuity of the signals. For example, there are continuously many possible durations to stay at ℓ_0 in the TA in Fig. 1.17. We use polyhedra to obtain finite abstraction of such possibilities and reduce the size of the weighted graph finite so that we can apply the existing shortest distance algorithm.

We implemented a prototypical tool QTPM for online quantitative timed pattern matching. Our experimental evaluation shows the efficiency of our algorithm, especially when the specification has a time-bound, i. e., the length of each matching is bounded. For example, for a realistic benchmark called Overshoot with time-bound of length 150 seconds, when the sampling interval is 10 seconds, QTPM monitors more than 3,000 entries, i. e., 30,000 seconds in the simulation time, per second 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). This assumption on the time-bound is not too restrictive because we are usually not interested in too long matching. For instance, for the TA in Fig. 1.17, although the length of the matching is unbounded, we are usually not interested in much longer matching than 3 time unit, which is the maximum duration between $v < 40$ and $v > 100$. Therefore, it does not harm the practical usefulness to give such a bound of the length of the matching.

In addition to the number of the monitored entries, we also measured the shortest feasible sampling interval of QTPM. In signal monitoring, the shortest feasible sampling interval is an important efficiency criterion. This is because by making the sampling interval shorter, we can make the monitoring result more accurate, while it makes monitoring more computationally demanding because the algorithm has to handle more information in a certain length of time.

For the aforementioned benchmark OVERSHOOT, the shortest feasible sampling interval of QTPM is about 450 milliseconds 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). Although this is not fast enough to monitor the raw sensor values of sampling interval 100 milliseconds, the bound of matching length (150 seconds) is much longer than the available sampling intervals, and the performance of QTPM may be enough for some applications.

1.6.3.3 Summary

Quantitative timed pattern matching is a quantitative extension of timed pattern matching computing the quantitative semantics rather than the Boolean satisfaction. Our contribution to quantitative timed pattern matching is summarized as follows.

- We define the quantitative timed pattern matching problem for TSWAs.
- We give an online algorithm for quantitative timed pattern matching.
- Our experimental result shows the efficiency of our results.

We used polyhedra-based symbolic analysis to obtain a finite abstraction of the weighted graph used in the algorithm. Our contribution makes runtime verification more generic and informative.

Genericity The quantitative semantics is parameterized with a semiring. Our algorithm works for different semirings provided they satisfy some conditions.

Informativeness For each subsignal, quantitative timed pattern matching computes the unsafe degree rather than deciding the Boolean satisfaction.

The material in Chapter 5 is based on work [Wag19]. Useful comments from the anonymous referees are gratefully acknowledged.

1.6.4 Model-bounded monitoring of hybrid systems

1.6.4.1 Motivation: Bounding possible signals between samples

Signal monitoring requires sampling and interpolation In the monitoring of CPSs, we monitor a signal, and this signal is thought of as a behavior of the system. For example, in Section 1.6.3, monitoring the signal σ shown in Fig. 1.15, we compute the quantitative semantics for each subsignal of σ. However, in reality, what we observe is not a *continuous signal* but a series of *discrete sampling points*, and we interpret them as a continuous signal by *interpolation*. For example, in Fig. 1.15, what we observe is the left end points of the constant pieces of a signal, and we interpret these observed samples as a signal by the piecewise-constant interpolation. In practice, the piecewise-constant or piecewise-linear interpolations (see Fig. 1.19) are commonly used, but we can also use other interpolations, e. g., with spline curves.

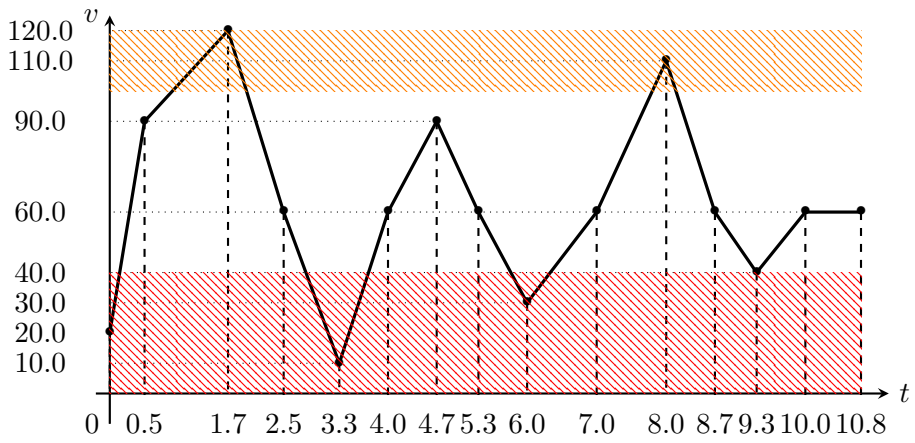


Figure 1.19: The signal constructed by the piecewise-linear interpolation from the samples in Fig. 1.15

Issue: no guarantee between samples In signal monitoring, due to the sampling and interpolation mechanism above, we have a methodological difficulty in guaranteeing the correctness of our analysis. Namely, we cannot determine the safety of the system between samples.

We note that this issue is ignorable when the sampling intervals are short, implicitly assuming that the system state does not change a lot in a short period. However, this can be a big issue when the sampling interval is longer, which is the case to reduce power consumption in an IoT-based monitoring system.

1.6.4.2 Contribution: Model-bounded monitoring for flexible interpolation

Idea: Exploit the knowledge of the monitored system Our solution is to bound the unknown behavior between samples using the prior knowledge of the monitored system. The latter is called *bounding model*; it is an abstraction of the monitored system. Utilizing the bounding model, we bound the behavior between the samples and determine if there is a possibility of unsafe behavior between samples. We call this procedure *model-bounded monitoring*.

We use *linear hybrid automata (LHAs)* as the formalism for bounding models. The use of LHAs is due to the following observation: on the one hand, LHAs are expressive, e. g., it allows discrete switching as well as bounding the derivative by the Lipschitz constant; on the other hand, LHAs is a limited subclass of hybrid automata, and its reachability analysis is tractable using convex polyhedra analysis.

Example 1.14 (thermostatic chamber)**.** Consider the monitoring of a thermostatic chamber. The thermostatic chamber has two modes, on and off, and at each location, the rate of the temperature change satisfies $\dot{T} \in [2.0, 4.0]$ and $\dot{T} \in [-2.0, 0]$, respectively. We note that the actual dynamics can be much more complicated, but we assume that these intervals bound the actual derivatives of the temperature under the current condition e. g., the output of the heater, range of the temperature, amount of the water, and room temperature. The LHA in Fig. 1.21 models such a thermostatic chamber.

Fig. 1.20 illustrates the model-bounded monitoring procedure. Our goal is to alert if the temperature enters the unsafe area; the red areas in Fig. 1.20 show the unsafe areas. We observe that all the samples (the black points $x_1, \ldots, x_4$) are in the safe area. For each pair (x_{i-1}, x_i)

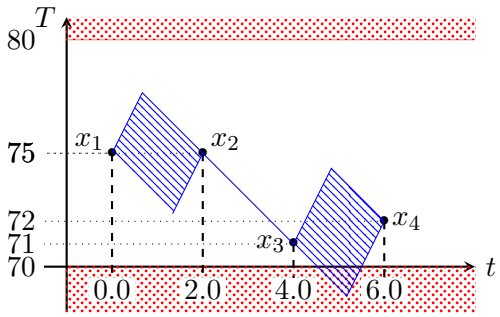


Figure 1.20: A signal of a thermostatic chamber: the black points $(x_1, \ldots, x_4)$ are the samples; the red dotted areas are the unsafe areas; the blue hatched areas are the reachable areas.

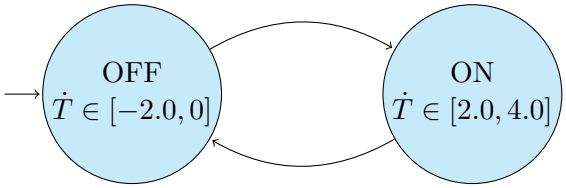


Figure 1.21: A bounding model of the thermostatic chamber in LHA. These intervals are e. g., the Lipschitz constants of the actual derivative.

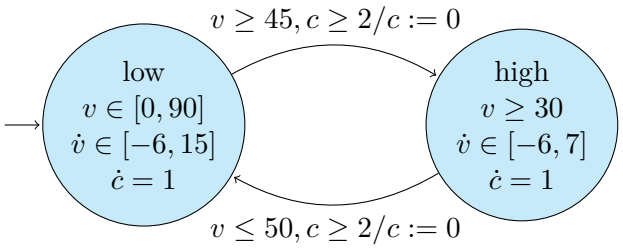


Figure 1.22: A bounding model of an automatic transmission system in LHA. These intervals are e. g., the Lipschitz constants of the actual derivative.

of samples, we compute the reachable area (the blue areas) through which the temperature moves from the previous sample x_{i-1} to the next sample x_i. The reachable area is represented by a finite union of convex polyhedra. Here, we observe that the blue and red areas intersect in between x_3 and x_4, which means that the unsafe area might be visited. Therefore, we alert that there is a potential safety violation, although we do not observe any violation in the samples. ◇

Example 1.15 (automatic transmission system)**.** Consider the monitoring of a car. The car has two gear modes, low and high, and at each location, the acceleration of the car satisfies $\dot{v} \in [-6, 15]$ and $\dot{v} \in [-6, 7]$, respectively. At low and high gear modes, the velocity must be in $v \in [0, 90]$ and $v \geq 30$, respectively. The gear can change from low to high and high to low when $v \geq 45$ and $v \leq 50$, respectively. There must be at least 2 time units between each gear change for the comfortability. The LHA in Fig. 1.22 models such a car.

Fig. 1.23 illustrates the model-bounded monitoring procedure. Our goal is to alert if the velocity is too high, i. e., $v \geq 120$; the red area in Fig. 1.23 shows the unsafe area. We observe that the reachable area between v_2 and v_3 is non-convex. This is because the maximum acceleration is different between the low velocity and the high velocity. Nevertheless, the

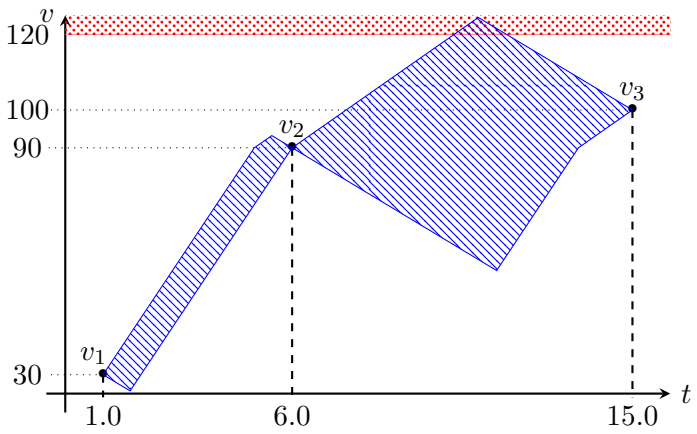


Figure 1.23: A signal showing the velocity of a car: the black points $(v_1, \ldots, v_3)$ are the samples; the red dotted area is the unsafe area; the blue hatched areas are the reachable areas.

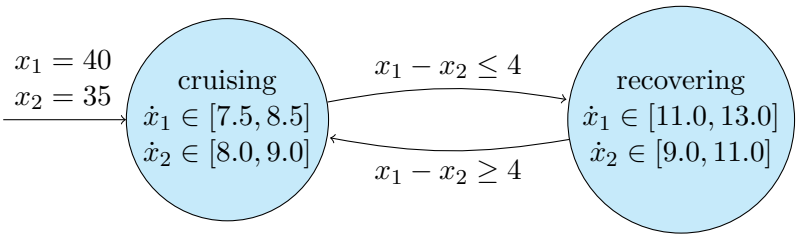


Figure 1.24: A bounding model $\mathcal{A}$ for the platooning example, expressed as an LHA

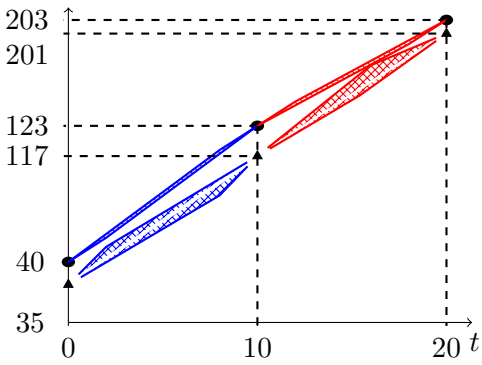


Figure 1.25: A signal of an automotive platooning system: the black circles are the samples of the first car; the black triangles are the samples of the second car; the hatched areas are the reachable areas.

reachable area is represented by finite union of convex polyhedra, and we can use the symbolic analysis of LHA with convex polyhedra. We observe that all the samples (the black points $v_1, \ldots, v_3$) are in the safe area. Nevertheless, we alert that there is a potential safety violation because we observe that the blue and red areas intersect in between v_2 and v_3. ◇

Example 1.16 (vehicle platooning)**.** Consider a *vehicle platooning* scenario where two vehicles drive one after the other, with their distance kept small. As mentioned in Example 1.9, such vehicle platooning attracts interest as a measure for enhanced road capacity as well as for fuel efficiency.

The vehicle platooning system has two modes, *cruising* and *recovering*. In the normal cruising mode, the relative distance has a drift and the inter-vehicle distance decreases. The recovering mode is used to recover the decreased inter-vehicle distance. At each location, the velocity of each car is $\dot{x}_1 \in [7.5, 8.5]$ and $\dot{x}_2 \in [8.0, 9.0]$; and $\dot{x}_1 \in [11.0, 13.0]$ and $\dot{x}_2 \in [9.0, 11.0]$, respectively. The LHA in Fig. 1.24 models such a vehicle platooning system.

Fig. 1.25 illustrates the model-bounded monitoring procedure. Our goal is to alert if the two vehicles potentially touched each other. Physical contact of the vehicles is not observed in Fig. 1.25, but we cannot be sure what happened between the sampling instants. The hatched areas show the reachable area i. e., the symbolic interpolation using the bounding model.

Here, we observe that in the first interpolation (the blue areas), there is no potential physical contact, while in the second interpolation (the red areas), there is a potential physical contact. Therefore, we alert that there is a potential safety violation, although we do not observe the violation in the samples. ◇

On the usage of the model One might say that the application area of model-bounded monitoring is smaller than the other runtime verification methods because it requires a model. Certainly, model-bounded monitoring requires an LHA as a bounding model, which is much like a system model for the exhaustive verification of hybrid systems [AHH96]. See Section 1.7.1 for a comparison between runtime verification and exhaustive verification. It is usually challenging to obtain system models, and the use of models usually limits the application area of the exhaustive verification.

However, we remark that in model-bounded monitoring, the model construction is not such a big issue as in exhaustive verification because the bounding model does not have to be as precise as the system model for the exhaustive verification. On the one hand, the analysis in exhaustive verification is *global*, and the modeling error accumulates over time; thus, even a small modeling error can cause a huge impact on the analysis. On the other hand, the reachability analysis in model-bounded monitoring is *local*, and the modeling error is reset at each sample. For example, in Fig. 1.20, we observe that the reachable area converges to one point at each sample.

Algorithms and implementations We show two approaches to solve the model-bounded monitoring problem: by a reduction to reachability analysis; and by a dedicated algorithm. We implemented the former approach with a model checker PHAVerLite and the latter approach in a prototype tool HAMoni. Both of the approaches share the technical essence, namely the bounded-time reachability analysis of LHAs. The reachability analysis of PHAVerLite is optimized with recent techniques, while HAMoni is a simple prototype tool using Parma Polyhedra Library for convex polyhedra operations. Our experimental results show the feasibility of model-bounded monitoring, especially by HAMoni. For example, for a benchmark called ACCD, HAMoni monitors 1,000 samples about in 12 seconds for dimension 5 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).

1.6.4.3 Summary

Model-bounded monitoring is a monitoring method guaranteed to detect every unsafe behavior even if the signal is coarsely sampled. Although we may have false alarms, we have fewer false

alarms for a more precise bounding model. Our contribution to model-bounded monitoring is summarized as follows.

- We define the model-bounded monitoring problem for bounding models in LHAs.
- We give two algorithms for model-bounded monitoring.
- Our experimental results show the feasibility of our approaches.

We use polyhedra for the symbolic interpolation between samples. Our contribution is flexible and informative.

Flexibility Model-bounded monitoring utilizes a flexible interpolation of the sampled logs considering the prior knowledge of the system called bounding model. Model-bounded monitoring is guaranteed to detect every unsafe behavior even if the log is sampled sparsely.

Informativeness By using parameters, we can synthesize parameter valuations and unobserved signal values much like in Sections 1.6.1 and 1.6.2. We note that the use of parameters is detailed in Appendix D while Chapter 6 focuses on a simpler setting.

The material in Chapter 6 is based on the unpublished joint work [WAH20] with Étienne André and Ichiro Hasuo. Useful comments from the anonymous referees are gratefully acknowledged.

1.7 Related work

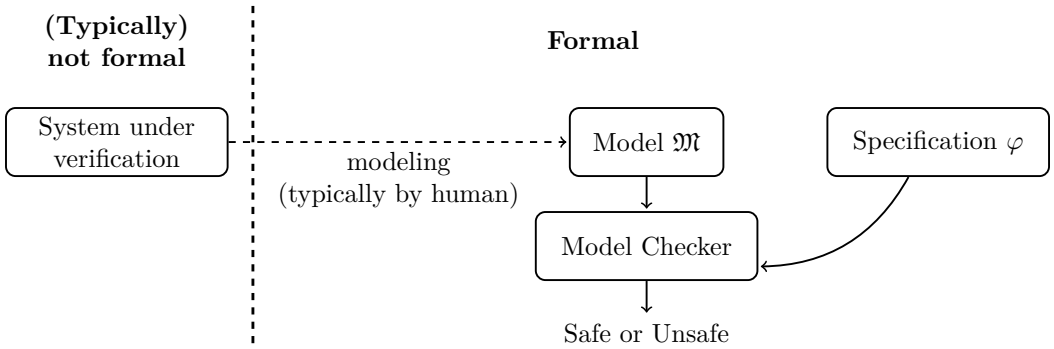
In this section, we give a high-level comparison with related research areas. See the related work sections in the coming chapters for the detailed discussion of works related to each contribution.

1.7.1 Polyhedra for verification

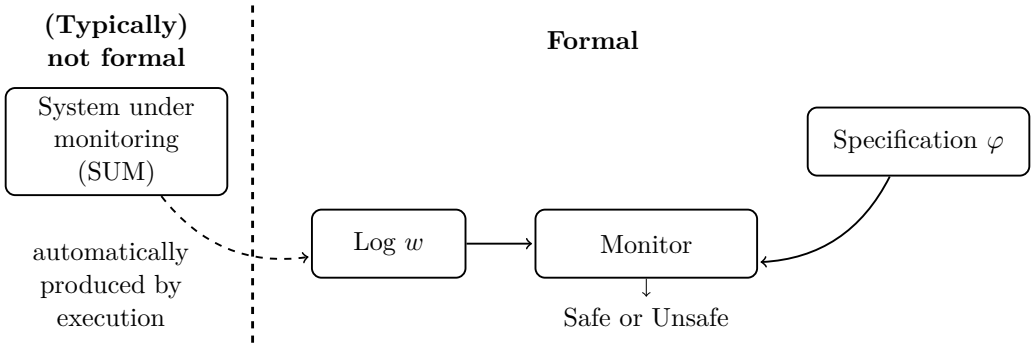
1.7.1.1 Exhaustive verification

"Are we creating the system correctly?" This is one of the fundamental questions in system development. *Exhaustive verification* (also called *formal verification*) mathematically answers this question by proving the satisfaction of the formal specification by the system. *Model checking* [BK08] is an automata-theoretic approach to prove the correctness of the system. Fig. 1.26a shows the typical scheme of model checking. Given a system model $\mathfrak{M}$ and a specification φ, model checking decides if the system model satisfies the specification for any execution.

The essence of the model checking is to reduce exhaustive verification to *graph analysis*, e. g., reachability checking and finding the strongly connected components. A model checking algorithm with *Büchi automata* and *linear temporal logic* (*LTL*) is shown in [Var95]. Since we can consider both of these formalisms as finite state systems, we can reduce the model checking of them to finite graph analysis. For example, the reachability to ℓ_3 in the automaton in Fig. 1.27a can be verified by breadth-first search.



(a) Typical model checking scheme. In the verification of CPSs, we usually have only informal information of the system under verification, for example, because of the third-party components or machine learning-based components whose internal mechanism is unknown. Moreover, it is highly challenging to precisely understand and model external environmental information such as road conditions, weather, and pedestrians' behavior. We note that usually, the model $\mathfrak{M}$ must be precise enough to verify the specifications φ.

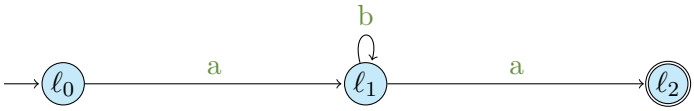


(b) Typical runtime verification scheme. We note that the absence of the model $\mathfrak{M}$ is not mandatory; For example, a bounding model is used in model-bounded monitoring (Chapter 6), while the model does not have to be as precise as that for model checking.

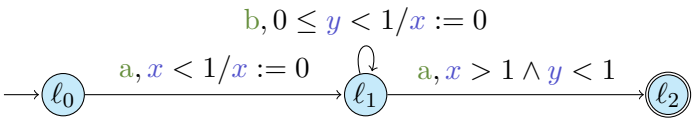
Figure 1.26: Comparison between model checking and runtime verification

Model-checking of CPSs with polyhedra-based abstraction In model checking of CPSs, it is not straightforward to apply these graph analysis methods because of the *continuous* state spaces to model *continuous-time* behavior (i. e., not *integer*-valued but *rational*- or *real*-valued time) in timed automata [AD94] or even the *continuous-valued signals* in hybrid automata [Hen96]. For example, the reachability to ℓ_2 in the automaton in Fig. 1.27b cannot be verified by naive breadth-first search because the underlying configuration is the *infinite* graph in Fig. 1.27c. Nevertheless, *polyhedra-based abstraction* solves this issue for a specific class of systems by discretizing the continuous state space as shown in Fig. 1.27d. See Section 2.3 for the detail. By using the polyhedra-based abstraction, various model checking tools have been introduced for various formalisms, e. g., for timed automata [LLN18], parametric timed automata [AFKS12], and linear hybrid automata [Fre08, BZ19]; they have been used in various case studies [LMNS05, MLR+10, AFMS19]. We note that for parametric timed automata and linear hybrid automata, model checking may not terminate because of the discrete but not always finite abstraction.

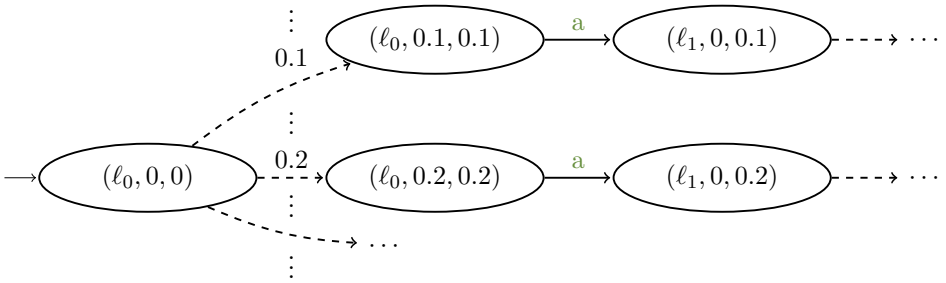
Exhaustive verification of CPSs is hard The result of exhaustive verification is strong; it rigorously and exhaustively proves the correctness of the system model. However, especially



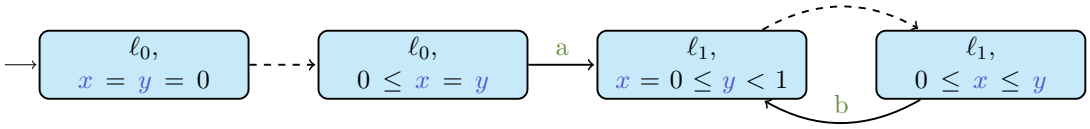
(a) An automaton with discrete time. The reachability to ℓ_2 can be verified by breadth-first search.



(b) A timed automaton with continuous time. Timing constraints are represented by guards, e. g., $x < 1$ that enables the transition only if we have $x < 1$, and the resets, e. g., $x := 0$ that resets the value of x. See Section 2.2 for the detail of timed automata.



(c) A part of the configuration space of the timed automaton in Fig. 1.27b. Each node (ℓ, x, y) is a tuple of the location ℓ, and the values of the clock variables x and y. The dashed edges are for the time elapse and the solid edges are for the transitions in Fig. 1.27b. This graph has infinite branching due to the infinite possibility of the time elapse.



(d) A symbolic abstraction of the configuration space in Fig. 1.27c. Each node consists of a location and a constraint representing a set of the clock valuations. This graph has only finite space and we can verify that ℓ_2 is unreachable by breadth-first search.

Figure 1.27: Comparison between exhaustive verification of an automaton with discrete time (Fig. 1.27a) and continuous time (Fig. 1.27b), where the reachability to ℓ_2 is verified.

for CPS, we have a difficulty in the model construction. In model checking, what is verified is not the *actual system* but the *system model*. Therefore, we need a system model to be accurate enough to verify the given specification. This is highly challenging for CPSs because we have to model the external environment of the system e. g., road condition, object, behavior of the pedestrians, which is not easy. To make matters worse, CPSs often contain third-party components whose internal mechanism are usually not specified; therefore, the construction of the precise system model (without the environment model) is still challenging.

1.7.1.2 Runtime verification

Runtime verification [BFFR18] (also called *specification-based monitoring*) is an alternative verification method. Fig. 1.26b shows a scheme of runtime verification. Runtime verification is a verification of the system observation (also called log) rather than a verification of the system model; therefore, it does not (necessarily) require system models. Given a system observation

Table 1.1: Summary of the comparison with related research areas

	system model	specification	background
Exhaustive Verification	necessary	given	logic
Runtime Verification	(typically) unnecessary	given	logic
Anomaly Detection	unnecessary	implicit/generated	statistics

w and a specification φ, runtime verification typically decides if the system observation w satisfies the specification φ. Since runtime verification is a verification of a system observation, it is not exhaustive with respect to all possible behavior of the system. Nevertheless, it is practically appealing that runtime verification does not require system models.

Runtime verification of CPSs Runtime verification of CPSs [BDD+18] as well as software systems is an active research topic given the recent trend of the highly sophisticated CPSs, e. g., self-driving cars and autonomous robots. This difference in the application area also affects the formalism. For example, in the runtime verification of software systems, LTL [BLS11] is widely used to express the monitored specification. In the monitoring of CPSs, however, these formalisms are usually not expressive enough to model the *continuous*-time or *real*-valued behavior. Therefore, the formalisms with continuous behaviors, e. g., *metric temporal logic (MTL)* [Koy90], *signal temporal logic (STL)* [MN04], timed regular expressions [ACM02], and timed automata [AD94], have been used in runtime verification of CPSs [TR05, MN04, DM10, UFAM14, WAH16]. These research efforts result in various monitoring tools e. g., [Don10, BKZ17, BKT17, Ulu17, NLM+18, WHS18].

Polyhedra-based abstraction for runtime verification In naive runtime verification, polyhedra-based abstraction is unnecessary because we have only finitely many possibilities thanks to the finiteness of the observation. For example, when we check if an observation matches a specification represented by a timed automaton, although the configuration space is infinitely many as shown in Fig. 1.27c, we only have to consider the time elapse in the observation, and thus, the relevant part of the configuration is only finite.

For advanced runtime verification techniques, however, we have to handle infinite possibilities, and polyhedra-based symbolic analysis plays an essential role in the algorithms, as we show in Section 1.6. Although polyhedra-based symbolic analysis is a common technique for exhaustive verification, it is utilized only in a few algorithms, e. g., in [ADMN11, BFM18, UFAM14, BFN+18]. In this thesis, we extensively use polyhedra-based symbolic analysis in various runtime verification algorithm to show its wide applicability.

1.7.2 Anomaly detection

Another research direction for system monitoring is *anomaly detection* [CBK09]. See Table 1.1 for a summary of the comparison. In anomaly detection, the monitoring is based on statistical analysis of *data* rather than logical *specifications*. The goal of anomaly detection is to detect *statistically anomalous* behaviors, i. e., rare events in the input space distribution. Anomaly detection learns the feature of the normal behavior from data and utilizes the learned feature as a detection criteria.

This data-driven approach of anomaly detection has both advantages and disadvantages. On the one hand, anomaly detection can raise false alarms because the detection criteria is not if the behavior is safe or unsafe but if the behavior is statistically rare or not. Runtime

verification does not detect such behavior as long as the specification precisely represents the safety. On the other hand, anomaly detection is applicable even if the specification is entirely unknown. This makes the application area of anomaly detection larger than that of runtime verification.

We note that a combination of runtime verification and anomaly detection is also studied [FdSC+17]. In [FdSC+17], the authors utilize formal specifications with uninterpreted predicates, which are similar to the parametric specification we use in Chapters 3 and 4. It is an interesting future work to combine our parametric runtime verification techniques with anomaly detection.

1.8 Information for readers

Fig. 1.3 shows the relations among the chapters. The preliminaries on the theory of timed automata and timed pattern matching in Chapter 2 are used in the remaining chapters. The contribution in Chapter 3 is generalized in Chapter 4; therefore, we recommend reading Chapter 3 before reading Chapter 4. Chapters 5 and 6 do not have such dependencies.

Although this thesis is meant to be self-contained, a deeper understanding of the timed automata theory, e. g., in [BY03a], will help read this thesis. We also note that Chapter 5 is mathematically more demanding than the other chapters because of the abstract discussion with semirings.

CHAPTER 2

Background: Timed Automata and Timed Pattern Matching

In this chapter, we review the common background on timed automata and timed pattern matching.

2.1 Timed words

We use *timed words* to represent system behavior[1]. A timed word is a sequence of events equipped with timestamps. By using a timed word, we represent a sequence of events that occurred in the system with timing information.

Definition 2.1 (timed word)**.** For an alphabet Σ, a *timed word* w is a sequence

$$(a_1, \tau_1), (a_2, \tau_2), \ldots, (a_n, \tau_n)$$

of pairs (a_i, τ_i) of a character $a_i \in \Sigma$ and a timestamp $\tau_i \in \mathbb{R}_{\geq 0}$ satisfying $\tau_i \leq \tau_{i+1}$. We let $\tau_0 = 0$. We denote the *length* n of w by $|w|$. The set of timed words over Σ is denoted by $\mathcal{T}(\Sigma)$ and for $n \in \mathbb{Z}_{\geq 0}$, we denote $\mathcal{T}^n(\Sigma) = \{w \in \mathcal{T}(\Sigma) \mid |w| = n\}$. For a timed word w, we often denote it by $(\overline{a}, \overline{\tau})$, where $\overline{a} \in \Sigma^*$ is the sequence $a_1, a_2, \ldots, a_n$ and $\overline{\tau} \in \mathbb{R}^*_{\geq 0}$ is the sequence $\tau_1, \tau_2, \ldots, \tau_n$. For a timed word $w = (\overline{a}, \overline{\tau})$, we denote the subsequence $(a_i, \tau_i), (a_{i+1}, \tau_{i+1}), \ldots, (a_j, \tau_j)$ by $w(i, j)$. For a timed word $w = (\overline{a}, \overline{\tau})$ and $t \in \mathbb{R}$ satisfying $-\tau_1 < t$, the *t-shift* of w is $(\overline{a}, \overline{\tau}) + t = (\overline{a}, \overline{\tau} + t)$ where $\overline{\tau} + t = \tau_1 + t, \tau_2 + t, \ldots, \tau_{|\tau|} + t$.

Let $w = (\overline{a}, \overline{\tau})$ and $w' = (\overline{a}', \overline{\tau}')$ be timed words. When $\tau_{|w|} \leq \tau'_1$, their *absorbing concatenation* $w \circ w'$ is the timed word

$$w \circ w' = (\overline{a} \cdot \overline{a}', \overline{\tau} \cdot \overline{\tau}') = (a_1, \tau_1), (a_2, \tau_2), \ldots, (a_{|w|}, \tau_{|w|}), (a'_1, \tau'_1), (a'_2, \tau'_2), \ldots, (a'_{|w'|}, \tau'_{|w'|})$$

[1] We note that we can also use a *signal* for the same purpose. See e. g., [ACM02] for comparison in the context of timed automata.

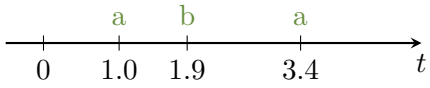


Figure 2.1: An example of a timed word w

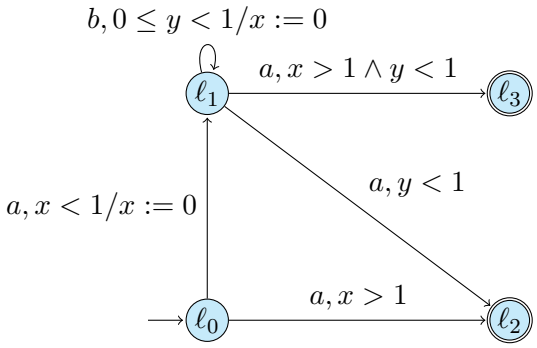


Figure 2.2: An example of a timed automaton $\mathcal{A}$

and their *non-absorbing concatenation* $w \cdot w'$ is the timed word $w \cdot w' = w \circ (w' + \tau_{|\overline{\tau}|}) = (\overline{a} \cdot \overline{a}', \overline{\tau} \cdot (\overline{\tau}' + \tau_{|\overline{\tau}|}))$. The concatenations on $\mathcal{T}(\Sigma)$ are also defined similarly. ◇

Example 2.2. The timed word w in Fig. 2.1 represents a sequence of events with timestamps such that "a happens at 1.0, b happens at 1.9, and a happens at 3.4". ◇

2.2 Timed automata

A property on timed words can be defined as a language of a *timed automaton* [AD94]. A timed automaton is an NFA equipped with *clock variables*, *clock resets*, and *guards* to represent constraints on the timestamps. Clock variables are also called *clocks* for simplicity. The following is the definition of clock valuations and guards. Though disjunctions are missing in our definition of guards, it does not harm the expressive power of timed automata because disjunctions can be encoded by nondeterministic branching.

Definition 2.3 (clock valuation, guard)**.** For a finite set $\mathbb{X}$ of clock variables, a *clock valuation* ν is a function $\nu \colon \mathbb{X} \to \mathbb{R}_{\ge 0}$. We write $\vec{0}$ for the clock valuation assigning 0 to all clock variables. For a clock valuation ν and $t \in \mathbb{R}_{\ge 0}$, the *t-shift* $\nu + t$ of ν is the clock valuation satisfying $(\nu + t)(x) = \nu(x) + t$ for each $x \in \mathbb{X}$. For a clock valuation $\nu \colon \mathbb{X} \to \mathbb{R}_{\ge 0}$ and a set $R \subseteq \mathbb{X}$ of clock variables, the clock valuation $[\nu]_R$ is such that $([\nu]_R)(x) = 0$ if $x \in R$, and otherwise, $([\nu]_R)(x) = \nu(x)$.

For a finite set $\mathbb{X}$ of clock variables, $\Phi(\mathbb{X})$ is the set of *guards* defined as follows.

$$g ::= \top \mid x \bowtie c \mid g \wedge g \qquad (x \in \mathbb{X}, \bowtie \in \{>, \ge, \le, <\}, \text{ and } c \in \mathbb{Z}_{\ge 0})$$

For a clock valuation $\nu \colon \mathbb{X} \to \mathbb{R}_{\ge 0}$ and a guard g over $\mathbb{X}$, we denote $\nu \models g$ if the guard g holds for the clock valuation ν. ◇

In timed automata, in addition to the structure from NFAs, each edge is associated with a guard g and a set R of the reset clock variables. For example, in the timed automaton in Fig. 2.2, the edge from ℓ_0 to ℓ_2 is associated with the guard $g = x > 1$ and the set $R = \emptyset$ of the reset clock variables i. e., no clock variable is reset at this guard.

Definition 2.4 (syntax of a TA)**.** A *timed automaton* (*TA*) $\mathcal{A}$ is a tuple $(\Sigma, L, L_0, L_F, \mathbb{X}, E)$ such that

- Σ is the alphabet,
- L is the finite set of locations,
- $L_0 \subseteq L$ is the set of initial locations,
- $L_F \subseteq L$ is the set of accepting locations,
- $\mathbb{X}$ is the finite set of clock variables, and
- $E \subseteq L \times \Phi(\mathbb{X}) \times \Sigma \times \mathcal{P}(\mathbb{X}) \times L$ is the set of edges.

◇

Example 2.5. The timed automaton $\mathcal{A} = (\Sigma, L, L_0, L_F, \mathbb{X}, E)$ in Fig. 2.2 is such that

- $\Sigma = \{a, b\}$,
- $L = \{\ell_0, \ell_1, \ell_2, \ell_3\}$,
- $L_0 = \{\ell_0\}$,
- $L_F = \{\ell_2, \ell_3\}$,
- $\mathbb{X} = \{x, y\}$, and
- $E = \{e_1, e_2, e_3, e_4, e_5\}$, where $e_1 = (\ell_0, x < 1, a, \{x\}, \ell_1)$, $e_2 = (\ell_0, x > 1, a, \emptyset, \ell_2)$, $e_3 = (\ell_1, 0 \leq y < 1, b, \{x\}, \ell_1)$, $e_4 = (\ell_1, y < 1, a, \emptyset, \ell_2)$, and $e_5 = (\ell_1, x > 1 \wedge y < 1, a, \emptyset, \ell_3)$.

We have the guard $y < 1$ at the edge from ℓ_1 to ℓ_2. Since the clock y is never reset in $\mathcal{A}$, we can use this edge only within 1 time unit from the beginning. ◇

Let us now recall the semantics of TAs.

Definition 2.6 (semantics of a TA)**.** For a TA $\mathcal{A} = (\Sigma, L, L_0, L_F, \mathbb{X}, E)$, the (concrete) semantics of $\mathcal{A}$ is defined by the *timed transition system* (*TTS*) $(S, S_0, \to)$, with

- $S = L \times (\mathbb{R}_{\geq 0})^{\mathbb{X}}$
- $S_0 = L_0 \times \{\vec{0}\}$,
- $\to$ consists of the discrete and (continuous) delay transition relations:
  1. *discrete transitions*: $(\ell, \nu) \overset{e}{\mapsto} (\ell', \nu')$, with $a \in \Sigma$, if there exists $e = (\ell, g, a, R, \ell') \in E$, such that $\nu' = [\nu]_R$, and $\nu \models g$.
  2. *delay transitions*: $(\ell, \nu) \overset{d}{\mapsto} (\ell, \nu + d)$, with $d \in \mathbb{R}_{\geq 0}$.

◇

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

Given a TA $\mathcal{A}$ with concrete semantics $(S, S_0, \to)$, we refer to the states S as the *(concrete) states* of $\mathcal{A}$.

Definition 2.7 (run, language)**.** A *run* of $\mathcal{A}$ is an alternating sequence of concrete states s_i of $\mathcal{A}$ and pairs (e_i, d_i) of edges and delays starting from an initial state $s_0 \in S_0$ of the form $s_0, (e_1, d_1), s_1, \ldots, s_n$ with $i = 1, 2, \ldots, n$, $e_i \in E$, $d_i \in \mathbb{R}_{\geq 0}$ and $s_{i-1} \xrightarrow{(e_i, d_i)} s_i$. Given such a run, the associated *timed word* is $(a_1, \tau_1), (a_2, \tau_2), \ldots, (a_n, \tau_n)$, where a_i is the action of edge e_i, and $\tau_i = \sum_{1 \leq j \leq i} d_j$, for $i = 1, 2, \ldots, n$. Given a concrete state $s = (\ell, \nu)$, we say that s is *reachable* in $\mathcal{A}$ if s appears in a run of $\mathcal{A}$. By extension, we say that ℓ is reachable; and by extension again, given a set K of locations, we say that K is reachable if there exists $\ell \in K$ such that ℓ is reachable in $\mathcal{A}$.

A run is *accepting* if its last state (ℓ, ν) is such that $\ell \in L_F$. The (timed) *language* $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is defined to be the set of timed words associated with at least one accepting run of $\mathcal{A}$. ◇

Example 2.8. In the TA $\mathcal{A}$ in Fig. 2.2, the location ℓ_2 is reachable because of the existence of the following run, where e_1 is the edge from ℓ_0 to ℓ_2, and we denote the clock valuation ν over $\{x, y\}$ by the pair $(\nu(x), \nu(y))$. The associated timed word to this run is $(a, 1.5)$. Since $\ell_2 \in L_F$, this run is accepting, and we have $(a, 1.5) \in \mathcal{L}(\mathcal{A})$.

$$(\ell_0, (0, 0)) \xrightarrow{(e_1, 1.5)} (\ell_2, (1.5, 1.5))$$

◇

2.3 Reachability checking of timed automata with polyhedra

Reachability checking is one of the most fundamental problems in the analysis of automata. In this section, we review the reachability checking of TAs with polyhedra as an example usage for polyhedra. For the simplicity, we show an algorithm for checking the reachability to a location. An extension for a set of locations or a set of concrete states is straightforward. We can also check the emptiness of the language by checking the reachability to the accepting locations.

The (single location) reachability checking problem:
INPUT: A TA $\mathcal{A} = (\Sigma, L, L_0, L_F, \mathbb{X}, E)$ and a location $\ell \in L$
PROBLEM: Decide if ℓ is reachable in $\mathcal{A}$

For a TA $\mathcal{A}$, the concrete states of $\mathcal{A}$ is in general infinite. Therefore, we cannot simply use BFS or DFS to check the reachability of a location. Nevertheless, for TAs, the reachability checking is decidable and proved to be PSPACE-complete [AD94].

One of the key ingredients in the reachability checking of TAs is *zones* [Dil89], which is a class of convex polyhedra. The reachability can be checked by *i*) constructing a (finite) graph from the TTS by abstracting the clock valuations using zones; and *ii*) conducting the standard finite graph reachability checking, e. g., with BFS. Such a graph construction is called *zone construction* and the constructed graph is called *zone graph*.

Definition 2.9 (zones)**.** Let $\mathbb{X}$ be the finite set of clock variables. A *zone* is a $|\mathbb{X}|$-dimensional convex polyhedron specified with a conjunction of the constraints of the form $x - x' \bowtie c$ or $x \bowtie c$, where $\bowtie \in \{<, \leq, \geq, >\}$, $x, x' \in \mathbb{X}$, and $c \in \mathbb{Z}$.

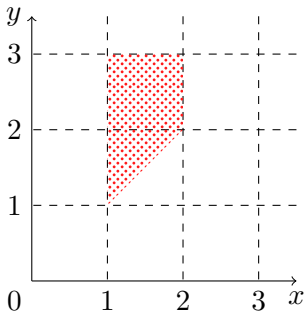


Figure 2.3: A zone Z representing $1 < x < 2 \wedge 1 < y < 3 \wedge x < y$

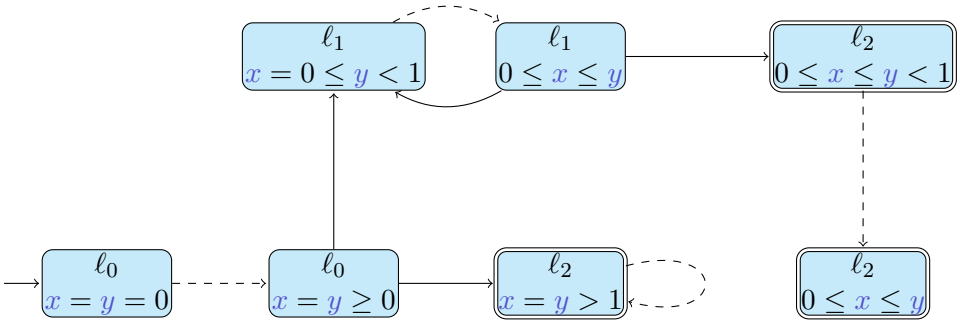


Figure 2.4: The zone graph of the TA in Fig. 2.2 with 1-normalization. The solid lines are for discrete transitions and the dashed lines are for delay transitions.

By a zone Z, we also denote the set of clock valuations ν satisfying $\nu \models Z$. ◇

Example 2.10. Fig. 2.3 shows an example of a zone Z. The zone Z represents all the clock valuations ν satisfying $1 < \nu(x) < 2$, $1 < \nu(y) < 3$, and $\nu(x) < \nu(y)$. ◇

Definition 2.11 (symbolic state)**.** For a TA $\mathcal{A} = (\Sigma, L, L_0, L_F, \mathbb{X}, E)$, a *symbolic state* is a pair (ℓ, Z) of a location $\ell \in L$ and a zone Z. A symbolic state (ℓ, Z) represents concrete states $\{(\ell, \nu) \mid \nu \in Z\}$. ◇

Since the number of the symbolic states is infinite, the reachability checking over the symbolic states may not terminate. Therefore, we need a finite abstraction to make the state space finite. One of such finite abstraction methods is the *k-normalization* [BY03b] $\mathfrak{a}_k(Z)$. For a zone Z, the k-normalization $\mathfrak{a}_k(Z)$ is the zone Z such that any constraint with constant larger than k are removed. [2]

Example 2.12. For the zone Z representing $1 < x < 2 \wedge 1 < y < 3 \wedge x < y$, the 2-normalization $\mathfrak{a}_2(Z)$ is $1 < x < 2 \wedge 1 < y \wedge x < y$. ◇

Example 2.13. Fig. 2.4 shows the zone graph of the TA in Fig. 2.2 with 1-normalization. By breadth-first search, we can check that ℓ_0, ℓ_1, and ℓ_2 are reachable from the initial concrete state while ℓ_3 is unreachable. ◇

Algorithm 1 shows an algorithm for the reachability checking by BFS with an on-the-fly finite abstraction. The loop from line 2 is the BFS over symbolic states. In the loop from line 6,

[2] We note that there are other finite abstraction methods (e. g., [BBFL03, BBLP06, HSW10, TM17]) and they come with different efficiency. Our choice of k-normalization is due to its simplicity.

Algorithm 1: Reachability checking of a TA by the zone construction

Input: A TA $\mathcal{A} = (\Sigma, L, L_0, L_F, \mathbb{X}, E)$, $k \in \mathbb{Z}_{\geq 0}$, and $\ell_f \in L$
Output: Return $\top$ if ℓ_f is reachable in $\mathcal{A}$, $\bot$ otherwise

1 $Conf_{\text{new}} \leftarrow \{(\ell_0, \vec{0}) \mid \ell_0 \in L_0\}$; $Conf \leftarrow Conf_{\text{new}}$ // initialize
2 **while** $Conf_{\text{new}} \neq \emptyset$ **do**
3 **if** $\exists (\ell_f, Z) \in Conf$ **then**
4 **return** $\top$
5 $Conf_{\text{tmp}} \leftarrow \emptyset$
6 **for** $(\ell, Z) \in Conf_{\text{new}}$ **do**
7 $Z_{\text{elapsed}} \leftarrow \{\nu + t \mid t \in \mathbb{R}_{\geq 0}, \nu \in Z\}$ // apply the time elapse
8 **for** $(\ell, g, a, R, \ell') \in E$ **do**
9 $Z_{\text{guarded}} \leftarrow \{\nu \mid \nu \in Z_{\text{elapsed}}, \nu \models g\}$ // the clock valuations satisfying the guard
10 **if** $Z_{\text{guarded}} \neq \emptyset$ **then**
11 $Z_{\text{reset}} \leftarrow \{[\nu]_R \mid \nu \in Z_{\text{guarded}}\}$ // apply the reset of the clock variables
12 **if** $\forall (\ell', Z') \in Conf. Z_{\text{reset}} \not\subseteq Z'$ **then**
13 **push** $(\ell', \mathfrak{a}_k(Z_{\text{reset}}))$ **to** $Conf_{\text{tmp}}$
14 $Conf_{\text{new}} \leftarrow Conf_{\text{tmp}}$; $Conf \leftarrow Conf \cup Conf_{\text{tmp}}$
15 **return** $\bot$

for each symbolic state (ℓ, Z) in $Conf_{\text{new}}$, we try to use each edge (ℓ, g, a, R, ℓ'). Namely, after applying the time elapse (line 7), we symbolically check if the guard g is satisfied (lines 9 and 10) and apply the reset to the clock variables (line 11). In line 13, we push the symbolic state to the $Conf_{\text{tmp}}$, which is the set of newly visited symbolic states.

Overall, Algorithm 1 terminates because *i*) it visits the symbolic states of $\mathcal{A}$ at most once and *ii*) the number of the symbolic states is finite after the abstraction. The correctness of Algorithm 1 is straightforward thanks to the following correctness of the abstraction.

Proposition 2.14 (correctness of the abstraction)**.** *Let $\mathcal{A} = (\Sigma, L, L_0, L_F, \mathbb{X}, E)$ be a timed automaton, $k \in \mathbb{Z}_{\geq 0}$ be the maximum constant in the guards of $\mathcal{A}$, Z be a zone. For any guard g in $\mathcal{A}$, we have $\exists \nu \in Z. \nu \models g$ if and only if we have $\exists \nu \in \mathfrak{a}_k(Z). \nu \models g$.*

Proof. Since $Z \subseteq \mathfrak{a}_k(Z)$, we have $\exists \nu \in Z. \nu \models g \Rightarrow \exists \nu \in \mathfrak{a}_k(Z). \nu \models g$.

Assume $\exists \nu \in \mathfrak{a}_k(Z). \nu \models g$ and let $\nu \in \mathfrak{a}_k(Z)$ be such a clock valuation. If for any $x \in \mathbb{X}$ we have $\nu(x) \leq k$, we have $\nu \in Z$ and thus $\exists \nu \in Z. \nu \models g$ holds. Otherwise, there is a clock valuation $\nu' \in Z$ such that for any $x \in \mathbb{X}$, *i*) $\nu(x) \leq k$ if and only if $\nu'(x) \leq k$; and *ii*) $\nu(x) \leq k$ implies $\nu(x) = \nu'(x)$. Since the constants in g is less than or equal to k, we have $\nu' \models g$. Therefore, we have $\exists \nu \in \mathfrak{a}_k(Z). \nu \models g \Rightarrow \exists \nu \in Z. \nu \models g$. □

2.4 Timed pattern matching

2.4.1 Problem definition

Timed pattern matching [UFAM14] is a mathematical formulation of runtime verification. We employ the definition using timed words in [WAH16, WHS17, WHS18] while the original definition in [UFAM14] is by signals. Given a timed word w and a timed automaton $\mathcal{A}$, the timed pattern matching problem locates in which part of w, the property specified by $\mathcal{A}$ is satisfied. The mathematical formulation is as follows.

Definition 2.15 (timed word segment)**.** For a timed word $w = (\overline{a}, \overline{\tau})$ on Σ and $t, t' \in \mathbb{R}_{\geq 0}$ satisfying $t < t'$, a *timed word segment* $w|_{(t,t')}$ is defined by the timed word $(w(i,j)-t)\circ(\$, t'-t)$ on the augmented alphabet $\Sigma \amalg \{\$\}$, where i, j are chosen so that $\tau_{i-1} \leq t < \tau_i$ and $\tau_j < t' \leq \tau_{j+1}$. Here the fresh symbol $\$$ is called the *terminal character*. ◇

Timed pattern matching problem:
INPUT: a timed word w over an alphabet Σ and a TA $\mathcal{A}$ over the augmented alphabet $\Sigma \amalg \{\$\}$
PROBLEM: compute all the intervals (t, t') for which the segment $w|_{(t,t')}$ is accepted by $\mathcal{A}$. That is, it requires the *match set* $\mathcal{M}(w, \mathcal{A}) = \{(t, t') \mid w|_{(t,t')} \in \mathcal{L}(\mathcal{A})\}$.

The match set $\mathcal{M}(w, \mathcal{A})$ is in general uncountable; however it allows finite representation, as a finite union of zones. See Example 1.3 for an example.

As in [WAH16, WHS17, WHS18], we make the following assumption on the TA in timed pattern matching, which is reasonable for the purpose of pattern matching.

Assumption 2.16. *We assume that all transitions to this accepting locations are labeled with* $\$$. ◇

2.4.2 Algorithm

Here, we review the online algorithm for timed pattern matching in [WAH16]. This algorithm finds all the intervals $(t, t') \in \mathcal{M}(w, \mathcal{A})$ by a breadth-first search. This algorithm is online in the following sense: after reading the i-th element (a_i, τ_i) of the timed word $w = (\overline{a}, \overline{\tau})$, it immediately outputs all the intervals (t, t') over the available prefix $(a_1, \tau_1), (a_2, \tau_2), \ldots, (a_i, \tau_i)$ of w.

Firstly, we define the auxiliary for the online algorithm for timed pattern matching. We introduce an additional variable t representing the absolute time of the beginning of the matching. We use a function $\rho\colon \mathbb{X} \to (\mathbb{R}_{>0} \amalg \{t\})$ to represent the latest reset time of each clock variable $x \in \mathbb{X}$. Intuitively, $\rho(x) = \tau \in \mathbb{R}_{>0}$ means the latest reset of x is at τ, and $\rho(x) = t$ means x is not reset after the beginning of the matching.

Definition 2.17 ($\mathrm{eval}(\rho, \tau), \mathrm{reset}(\rho, R, \tau), \rho_\emptyset$)**.** Let $\mathbb{X}$ be the set of clock variables and t be the variable for the beginning of a matching. For a function $\rho\colon \mathbb{X} \to (\mathbb{R}_{\geq 0} \amalg \{t\})$ and the current time $\tau \in \mathbb{R}_{\geq 0}$, $\mathrm{eval}(\rho, \tau)$ is the following constraint on $\mathbb{X} \amalg \{t\}$.

$$\mathrm{eval}(\rho, \tau) = \bigwedge_{x \in \mathbb{X}} (x = \tau - \rho(x))$$

For a function $\rho\colon \mathbb{X} \to (\mathbb{R}_{\geq 0} \amalg \{t\})$, the set $R \subseteq \mathbb{X}$ of clocks to be reset, and the current time $\tau \in \mathbb{R}_{\geq 0}$, $\mathrm{reset}(\rho, R, \tau)\colon \mathbb{X} \to (\mathbb{R}_{\geq 0} \amalg \{t\})$ is the following function.

$$\mathrm{reset}(\rho, R, \tau)(x) = \begin{cases} \tau & \text{if } x \in R \\ \rho(x) & \text{if } x \notin R \end{cases}$$

By $\rho_\emptyset\colon \mathbb{X} \to (\mathbb{R}_{\geq 0} \amalg \{t\})$, we denote the function mapping each $x \in \mathbb{X}$ to t. ◇

Intuitively, $\mathrm{eval}(\rho, \tau)$ is the constraint corresponding to the clock valuation, and $\mathrm{reset}(\rho, R, \tau)$ is the operation to reset the clock variables $x \in R$ at τ.

Algorithm 2: An algorithm for timed pattern matching [WAH16]

Input: A timed word $w = (\overline{a}, \overline{\tau})$, and a TA $\mathcal{A} = (\Sigma, L, L_0, L_F, \mathbb{X}, E)$.
Output: $\bigvee Z$ is the match set $\mathcal{M}(w, \mathcal{A})$

1 $\mathit{CurrConf} \leftarrow \emptyset$; $Z \leftarrow \emptyset$
2 **for** $i \leftarrow 1$ **to** $|w|$ **do**
3 **push** $(\ell_0, \rho_\emptyset, [\tau_{i-1}, \tau_i))$ **to** $\mathit{CurrConf}$
4 **for** $(\ell, \rho, I) \in \mathit{CurrConf}$ **do** // lines 4 to 7 try to insert $ in $(\tau_{i-1}, \tau_i]$.
5 **for** $\ell_f \in L_F$ **do**
6 **for** $(\ell, g, \$, R, \ell_f) \in E$ **do**
7 **push** $(t \in I \wedge (\tau_{i-1} < t' \le \tau_i) \wedge g \wedge \mathrm{eval}(\rho, t'))$ **to** Z
8 $(\mathit{PrevConf}, \mathit{CurrConf}) \leftarrow (\mathit{CurrConf}, \emptyset)$
9 **for** $(\ell, \rho, I) \in \mathit{PrevConf}$ **do** // lines 9 to 13 try to go forward using (a_i, τ_i).
10 **for** $(\ell, g, a_i, R, \ell') \in E$ **do**
11 $I' \leftarrow \{\tau \in I \mid (t = \tau) \wedge g \wedge \mathrm{eval}(\rho, \tau_i) = \top\}$
 // Narrow the interval I to satisfy the guard g.
12 **if** $I' \neq \emptyset$ **then**
13 **push** $(\ell', \mathrm{reset}(\rho, R, \tau), I')$ **to** $\mathit{CurrConf}$
14 **push** $(\ell_0, \rho_\emptyset, (\tau_{|w|}, \infty)\})$ **to** $\mathit{CurrConf}$ // for the trimming after the final event
15 **for** $(\ell, \rho, I) \in \mathit{CurrConf}$ **do** // lines 15 to 18 try to insert $ in $(\tau_{|w|}, \infty)$.
16 **for** $\ell_f \in L_F$ **do**
17 **for** $(\ell, g, \$, R, \ell_f) \in E$ **do**
18 **push** $((t \in I) \wedge (\tau_{|w|} \le t' < \infty) \wedge g \wedge \mathrm{eval}(\rho, t'))$ **to** Z

Algorithm 2 shows an algorithm for timed pattern matching. In the pseudocode, we used *CurrConf*, *PrevConf*, and Z. *CurrConf* and *PrevConf* are finite sets of triples (ℓ, ρ, I) made of a location $\ell \in L$, a mapping $\rho\colon \mathbb{X} \to (\mathbb{R}_{\ge 0} \amalg \{t\})$ denoting the latest reset of each clock, and an interval I. Z is a finite set of constraints over $\{t, t'\}$. As a running example, we use the TA and the timed word in Fig. 1.4.

At first, the counter i is 1 (line 2), and we start the matching trial from $t \in [\tau_0, \tau_1) = [0, 0.2)$. At line 3, we add the new configuration $(\ell_0, \rho_\emptyset, [\tau_0, \tau_1))$ to *CurrConf*, which means we are at the initial location ℓ_0, we have no reset of the clock variables yet, and we can potentially start the matching from any $t \in [\tau_0, \tau_1)$. In lines 5 to 8, we try to insert \$ (i. e., the end of the matching) in $(\tau_0, \tau_1]$. In our running example in Fig. 1.4, since there is no edge from ℓ_0 to the accepting location ℓ_4, we immediately jump to line 8. Then, in lines 10 to 14, we consume $(a_1, \tau_1) = (\mathrm{g}_3, 0.2)$ and try to transit from ℓ_0 to ℓ_1. Since $a_1 = \mathrm{g}_3$ and there is no edge from ℓ_0 labeled with g_3, we go back to line 3.

When $i = 2$ at line 3, we add the new configuration $(\ell_0, \rho_\emptyset, [\tau_1, \tau_2))$ to *CurrConf*. Similarly, we immediately jump to line 8, and we try the edge from ℓ_0 to ℓ_1 in lines 10 to 14. This time, because the edge from ℓ_0 to ℓ_1 is labeled with $a_2 = \mathrm{g}_4$, we use the edge to ℓ_1. The guard at the label is $\top$, and we push the next configuration $(\ell_1, \mathrm{reset}(\rho_\emptyset, \{x\}, \tau_2), [\tau_1, \tau_2))$ to *CurrConf* at line 13.

When $i = 3$ and $i = 4$, since $a_i \neq \mathrm{g}_4$, the new configurations $(\ell_0, \rho_\emptyset, [\tau_{i-1}, \tau_i))$ cannot go from ℓ_0 to ℓ_1 while the existing configuration $(\ell_1, \mathrm{reset}(\rho_\emptyset, \{x\}, \tau_2), [\tau_1, \tau_2))$ remains by using the loop at ℓ_1. Overall, we have $\mathit{CurrConf} = \{(\ell_1, \mathrm{reset}(\rho_\emptyset, \{x\}, \tau_2), [\tau_1, \tau_2))\}$ when $i = 4$ at line 13.

When $i = 5$, similarly to the previous loop, after adding the new configuration $(\ell_0, \rho_\emptyset, [\tau_4, \tau_5))$

to *CurrConf*, we immediately jump to line 8. For $(\ell_0, \rho_\emptyset, [\tau_4, \tau_5))$, we cannot use the edge from ℓ_0 to ℓ_1 because of $a_5 \neq \mathrm{g}_4$. For $(\ell_1, \mathrm{reset}(\rho_\emptyset, \{x\}, \tau_2), [\tau_1, \tau_2))$, we use the edge from ℓ_1 to ℓ_2 because $a_5 = \mathrm{g}_1$. The guard $x < 1$ at the edge from ℓ_1 to ℓ_2 is examined at line 11. Since $\mathrm{eval}(\rho, \tau_5) = \{x = \tau_5 - \tau_2 = 0.9\}$, the constraint at line 11 is $t = \tau \wedge x < 1 \wedge x = 0.9$, which is satisfied by any $\tau \in I = [\tau_1, \tau_2)$ and thus we have $I' = I$. Because $I' \neq \emptyset$, we push the new configuration $(\ell_2, \mathrm{reset}(\rho_\emptyset, \{x\}, \tau_2), [\tau_1, \tau_2))$ to *CurrConf* at line 13.

When $i = 6$, since we have an edge from ℓ_2 to the accepting location ℓ_3, we go to the accepting location from the configuration $(\ell_2, \mathrm{reset}(\rho_\emptyset, \{x\}, \tau_2), [\tau_1, \tau_2))$. At line 7, we push the following constraint to Z, which is equivalent to the result of timed pattern matching shown in Example 1.3.

$$t \in [0.2, 0.7) \wedge t' \in (1.6, 2.0] \wedge \top \wedge x = t' - 0.7$$

Algorithm 2 terminates because the size of *CurrConf* is always finite. Algorithm 2 is correct because it symbolically keeps track of all the runs of $\mathcal{A}$ over $w|_{(t,t')}$ for any $(t, t') \subseteq \mathbb{R}_{\geq 0}$.

CHAPTER 3

Parametric Timed Pattern Matching

In this chapter, we introduce the *parametric timed pattern matching problem* by using parametric timed automata instead of timed automata. See Section 1.6.1 for a motivation and a summary of the contribution. This chapter is based on the joint work [AHW18, WA19] with Étienne André and Ichiro Hasuo. Useful comments from the anonymous referees of these papers are gratefully acknowledged.

Organization of the chapter We introduce the necessary definitions in Section 3.1 and define the parametric timed pattern matching problem in Section 3.2. We present and evaluate our method based on parametric timed model checking in Section 3.3. We then present and evaluate a *dedicated* method, enhanced with automata-based skipping, in Section 3.4. In Section 3.5, we discuss the comparison between the approaches in Sections 3.3 and 3.4. After reviewing the related work in Section 3.6, we conclude in Section 3.7.

3.1 Preliminaries: Parametric timed automata

In this section, we show the definition of *parametric timed automata*, which are a parametric extension of timed automata in Section 2.2.

We assume a set $\mathbb{X} = \{x_1, \ldots, x_H\}$ of *clock variables*, i. e., real-valued variables that evolve at the same rate. We assume a set $\mathbb{P} = \{p_1, \ldots, p_M\}$ of *parameters*, i. e., unknown constants. A parameter *valuation* v is a function $v : \mathbb{P} \to \mathbb{Q}_+$. We assume $\bowtie \in \{<, \leq, =, \geq, >\}$. A guard g is a constraint over $\mathbb{X} \cup \mathbb{P}$ defined by a conjunction of inequalities of the form $x \bowtie d$, or $x \bowtie p$ with $d \in \mathbb{N}$ and $p \in \mathbb{P}$. Given g, we write $\nu \models v(g)$ if the expression obtained by replacing each x with $\nu(x)$ and each p with $v(p)$ in g evaluates to true.

A linear term over $\mathbb{X} \cup \mathbb{P}$ is of the form $\sum_{1 \leq i \leq H} \alpha_i x_i + \sum_{1 \leq j \leq M} \beta_j p_j + d$, with $x_i \in \mathbb{X}$, $p_j \in \mathbb{P}$, and $\alpha_i, \beta_j, d \in \mathbb{Z}$. A *constraint* C (i. e., a convex polyhedron) over $\mathbb{X} \cup \mathbb{P}$ is a conjunction of inequalities of the form $lt \bowtie 0$, where lt is a linear term. Given a set $\mathbb{P}$ of parameters, we denote by $C{\downarrow_{\mathbb{P}}}$ the projection of C onto $\mathbb{P}$, i. e., obtained by eliminating the variables not in $\mathbb{P}$

(e. g., using Fourier-Motzkin [Sch86]). $\bot$ denotes the constraint over $\mathbb{P}$ representing the empty set of parameter valuations.

3.1.1 Parametric timed automata

Parametric timed automata (*PTA*) extend timed automata with parameters within guards in place of integer constants [AHV93].

Definition 3.1 (PTA)**.** A PTA $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, L, \ell_0, L_F, \mathbb{X}, \mathbb{P}, E)$, where:

1. Σ is a finite set of actions,
2. L is a finite set of locations,
3. $L_0 \subseteq L$ is the set of initial locations,
4. $L_F \subseteq L$ is the set of accepting locations,
5. $\mathbb{X}$ is the finite set of clock variables,
6. $\mathbb{P}$ is the finite set of parameters,
7. E is the finite set of edges $e = (\ell, g, a, R, \ell')$ where $\ell, \ell' \in L$ are the source and target locations, $a \in \Sigma$, $R \subseteq \mathbb{X}$ is a set of clock variables to be reset, and g is a guard.

$\diamond$

See Fig. 1.5 for an example of PTAs.

Given a parameter valuation v, we denote by $v(\mathcal{A})$ the non-parametric structure where all occurrences of a parameter p_i have been replaced by $v(p_i)$. We refer as a timed automaton to any structure $v(\mathcal{A})$, by assuming a rescaling of the constants: by multiplying all constants in $v(\mathcal{A})$ by the least common multiple of their denominators, we obtain an equivalent (integer-valued) TA, as defined in Section 2.2.

The synchronous product (using strong broadcast, i. e., synchronization on shared actions) of several PTAs gives a PTA.

Definition 3.2 (synchronized product of PTAs)**.** Let $N \in \mathbb{N}$. Given a set of PTAs $\mathcal{A}^i = (\Sigma^i, L^i, L^i_0, L^i_F, \mathbb{X}^i, \mathbb{P}^i, E^i)$, $1 \leq i \leq N$, the *synchronized product* of $\mathcal{A}^i$, $1 \leq i \leq N$, denoted by $\mathcal{A}^1 \parallel \mathcal{A}^2 \parallel \cdots \parallel \mathcal{A}^N$, is the tuple $(\Sigma, L, L_0, L_F, \mathbb{X}, \mathbb{P}, E)$, where:

1. $\Sigma = \bigcup_{i=1}^{N} \Sigma^i$,
2. $L = \prod_{i=1}^{N} L^i$,
3. $L_0 = L^1_0 \times \cdots \times L^N_0$,
4. $L_F = \{(\ell_1, \ldots, \ell_N) \in L \mid \exists i \in [1, N] \text{ s.t. } \ell^i \in L^i_F\}$,
5. $\mathbb{X} = \bigcup_{1 \leq i \leq N} \mathbb{X}^i$,
6. $\mathbb{P} = \bigcup_{1 \leq i \leq N} \mathbb{P}^i$,

and E is defined as follows. For all $a \in \Sigma$, let ζ_a be the subset of indices $i \in 1, \ldots, N$ such that $a \in \Sigma^i$. For any $a \in \Sigma$, for any $(\ell_1, \ldots, \ell_N) \in L$, and for any $(\ell'_1, \ldots, \ell'_N) \in L$, we have $((\ell_1, \ldots, \ell_N), g, a, R, (\ell'_1, \ldots, \ell'_N)) \in E$ if:

- for any $i \in \zeta_a$, there exist g_i, R_i such that $(\ell_i, g_i, a, R_i, \ell'_i) \in E^i$, $g = \bigwedge_{i \in \zeta_a} g_i$, $R = \bigcup_{i \in \zeta_a} R_i$, and,
- for all $i \notin \zeta_a$, $\ell'_i = \ell_i$.

◇

For a PTA $\mathcal{A}$ and a parameter valuation v, the semantics and the language of the instantiated TA $v(\mathcal{A})$ is the same as the usual TAs. See Definition 2.6.

3.1.2 Reachability synthesis

We use here reachability synthesis for the following two purposes.

- In Section 3.3: to solve parametric timed pattern matching
- In Section 3.4: to improve the dedicated parametric timed pattern matching algorithm (Algorithm 4) with a skipping optimization

This procedure, called EFsynth, takes as input a PTA $\mathcal{A}$ and a set of target locations K, and attempts to synthesize all parameter valuations v for which K is reachable in $v(\mathcal{A})$. EFsynth was formalized in e. g., [JLR15] and is a procedure that may not terminate, but that computes an exact result (sound and complete) if it terminates. EFsynth traverses the *parametric zone graph* of $\mathcal{A}$, which is a potentially infinite extension of the zone construction in Section 2.3 (see, e. g., [ACEF09, JLR15] for a formal definition).

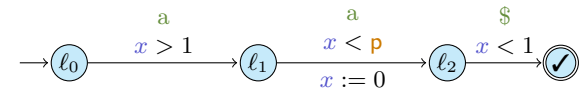
3.2 Parametric timed pattern matching

We extend the timed pattern matching problem in Section 2.4 to parameters by allowing a specification expressed using PTAs. The problem now requires not only the start and end dates for which the property holds, but also the associated parameter valuations.

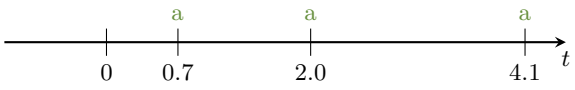
Parametric timed pattern matching problem:
INPUT: a timed word w over an alphabet Σ and a PTA $\mathcal{A}$ over the augmented alphabet $\Sigma \uplus \{\$\}$
PROBLEM: compute all the triples (t, t', v) for which the segment $w|_{(t,t')}$ is accepted by $v(\mathcal{A})$. That is, it requires the *match set* $\mathcal{M}(w, \mathcal{A}) = \{(t, t', v) \mid w|_{(t,t')} \in \mathcal{L}(v(\mathcal{A}))\}$.

The match set $\mathcal{M}(w, \mathcal{A})$ is in general uncountable; however, we will see that it can still be represented as a finite union of polyhedra, but in more dimensions, viz., $|\mathbb{P}| + 2$, i. e., the number of parameters + 2 further dimensions for t and t'. In addition, the form of the obtained polyhedra is more general than zones, as parameters may "accumulate" to produce sums of parameters with coefficients (e. g., $3 \times p_1 < p_2 + 2 \times p_3$).

Example 3.3. Fig. 3.1 shows an example of parametric timed pattern matching: Given the PTA $\mathcal{A}$ and the timed word w, the parametric timed pattern matching problem asks for the



(a) the input PTA $\mathcal{A}$



(b) the input timed word w

$$\begin{cases} 0.7 \le t < 1.0 \\ 4.1 < p + t \\ 4.1 < t' \le 5.1 \end{cases}$$

(c) the convex polyhedron representing the match set $\mathcal{M}(w, \mathcal{A})$

Figure 3.1: Example of parametric timed pattern matching

match set $\mathcal{M}(w, \mathcal{A})$ in Fig. 3.1c. We observe that the convex polyhedron in the match set has three dimensions for the parameter p and for t and t'. We also observe that this convex polyhedron is not a zone because it contains the constraint $4.1 < p + t$. ◇

In the rest of this chapter, we make the following assumption on the PTA in parametric timed pattern matching as in Section 2.4.

Assumption 3.4. *We assume that all transitions to the accepting locations are labeled with* $.
◇

3.3 Algorithm I: via reduction to PTA reachability analysis

3.3.1 General approach

In addition to Assumption 3.4, we make the following two assumptions, that do not impact the correctness of our method, but simplify the subsequent reasoning.

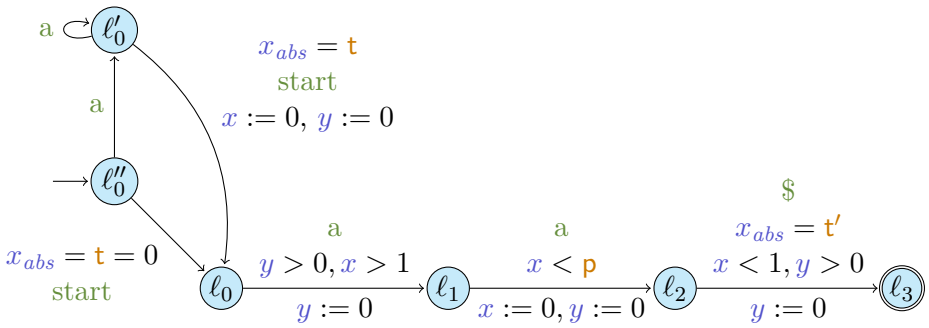
Assumption 3.5. *We assume that the pattern automaton contains a single accepting location.*
◇

Assumption 3.6. *We assume that the initial locations L_0 of $\mathcal{A}$ is a singleton, i. e., $L_0 = \{\ell_0\}$ and there is no incoming edge to ℓ_0.* ◇

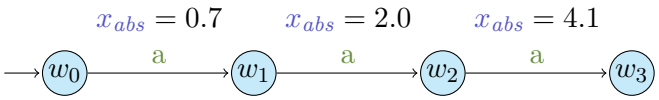
All the assumptions are easy to remove in practice: for Assumption 3.5, if the pattern PTA contains more than one accepting location, they can be merged into a single accepting location. For Assumption 3.6, if the pattern PTA contains incoming edges to the initial location, we can split the initial location into the initial location without incoming edges and the non-initial location with incoming edges; if the pattern PTA contains more than one initial location, they can be merged into a single initial location.

We show using the following approach that parametric timed pattern matching can reduce to parametric reachability analysis.

1. We turn the pattern into a symbolic pattern, by allowing it to start anytime. In addition, we use two parameters to measure the (symbolic) starting time and the (symbolic) ending time of the pattern.
2. We turn the timed word into a (non-parametric) timed automaton that uses a single clock x_{abs} measuring the absolute time.
3. We consider the synchronized product of the symbolic pattern PTA and the timed word (P)TA.



(a) MakeSymbolic applied to the PTA in Fig. 3.1a



(b) TW2PTA applied to the timed word in Fig. 3.1b

Figure 3.2: Our transformations exemplified on Fig. 3.1

4. We run the reachability synthesis algorithm to derive all possible parameter valuations for which the accepting location of the pattern automaton is reachable.

3.3.2 Our approach step by step

3.3.2.1 Making the pattern symbolic

In this first step, we first add two parameters t and t', which encode the (symbolic) start and end time where the pattern holds on the input timed word. This way, we will obtain a result in the form of a finite union of polyhedra in $M+2$ dimensions, where the 2 additional dimensions come from the addition of t and t'. We also add a clock x_{abs} initially 0 and never reset (this clock is shared by the pattern PTA and the subsequent timed word TA). Then, we modify the pattern PTA as follows:

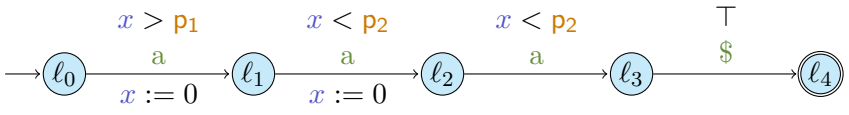
1. we add two fresh locations (say ℓ'_0 and ℓ''_0) prior to the initial location ℓ_0;
2. we add a fresh clock (say y), which is reset at each edge;
3. we add a guard $y > 0$ on all the transitions from ℓ_0;
4. we add an unguarded self-loop allowing any action of the timed word on ℓ'_0;
5. we add an unguarded transition from ℓ''_0 to ℓ'_0 allowing any action of the timed word;
6. we add a transition from ℓ'_0 to ℓ_0 guarded by $x_{abs} = t$, labeled with a fresh action start and resetting all clock variables of the pattern (except x_{abs});
7. we add a transition from ℓ''_0 to ℓ_0 guarded by $x_{abs} = t \wedge x_{abs} = 0$, labeled with start;
8. we add a guard $x_{abs} = t' \wedge y > 0$ on the accepting transitions labeled with \$;
9. the initial location of the modified PTA becomes ℓ''_0.

Let us give the intuition behind our transformation. First, the two guards $x_{abs} = t$ and $x_{abs} = t'$ allow recording symbolically the value of the starting and ending dates. Second, the

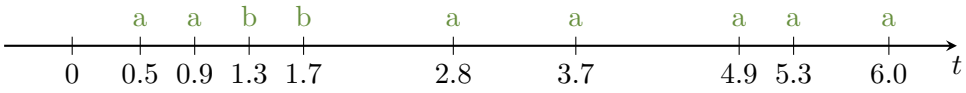
Algorithm 3: PTPM$(\mathcal{A}, w)$

input : A pattern PTA $\mathcal{A}$ with accepting locations L_F, a timed word w
output: Constraint K over the parameters

1 $\mathcal{A}' \leftarrow$ MakeSymbolic$(\mathcal{A})$
2 $\mathcal{A}_w \leftarrow$ TW2PTA(w)
3 **return** EFsynth$(\mathcal{A}' \parallel \mathcal{A}_w, L_F)$



(a) A parametric timed automaton



(b) A timed word

Figure 3.3: An example of parametric timed pattern matching

new locations ℓ_0'' and ℓ_0' allow the pattern to "start anytime"; that is, it can synchronize with the timed word TA for an arbitrary long time while staying in the initial location ℓ_0'' (and therefore *not* matching the pattern), and start (using the transition from ℓ_0' to ℓ_0) anytime. Third, due to the constraint $y > 0$, a non-zero time must elapse between the beginning of the pattern and the first action in the actual pattern. The distinction between ℓ_0'' and ℓ_0' is necessary to also allow starting the pattern at $x_{abs} = 0$ if no action occurred before. Finally, the guard $y > 0$ just before the accepting location ensures that a non-zero time must elapse between the last action in the pattern and the end of the pattern matching. Let MakeSymbolic denote this procedure.

Consider the pattern PTA $\mathcal{A}$ in Fig. 3.1a. The result of MakeSymbolic$(\mathcal{A})$ is given in Fig. 3.2a.

3.3.2.2 Converting the timed word into a (P)TA

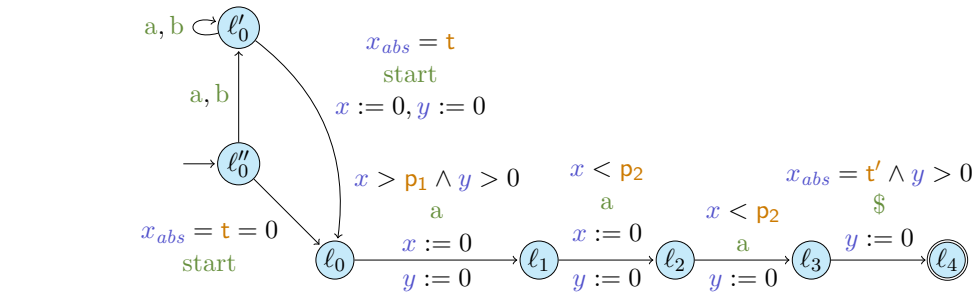
In this second step, we convert the timed word into a (non-parametric) timed automaton. This is very straightforward, and simply consists in converting a timed word of the form $(a_1, \tau_1), \cdots, (a_n, \tau_n)$ into a sequence of transitions labeled with a_i and guarded with $x_{abs} = \tau_i$ (recall that x_{abs} measures the absolute time and is shared by the timed word automaton and the pattern automaton). Let TW2PTA denote this procedure.

Consider the timed word w in Fig. 3.1b. The result of TW2PTA(w) is given in Fig. 3.2b.

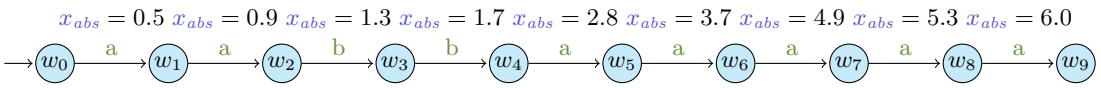
3.3.2.3 Synchronized product

The last part of the method consists in performing the synchronized product of MakeSymbolic$(\mathcal{A})$ and TW2PTA(w), and calling EFsynth on the resulting PTA.

We summarize our method PTPM$(\mathcal{A}, w)$ in Algorithm 3.

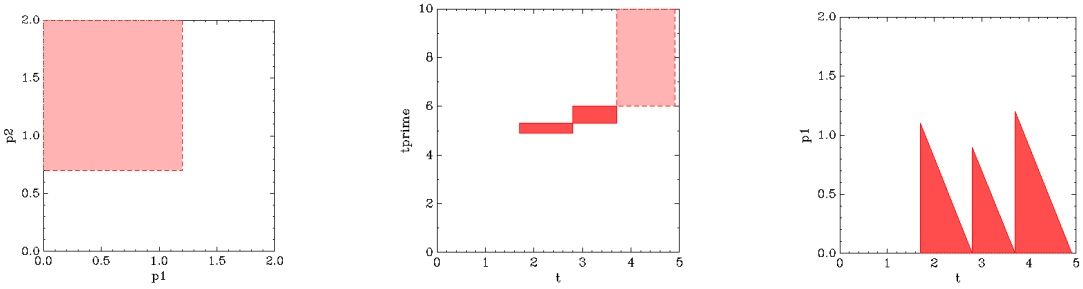


(a) MakeSymbolic applied to the PTA in Fig. 3.3a



(b) TW2PTA applied to the timed word in Fig. 3.3b

Figure 3.4: Our transformations exemplified on Fig. 3.3



(a) On p_1 and p_2 (b) On t and t' (c) On t and p_1

Figure 3.5: Projections of the result of parametric timed pattern matching on Fig. 3.3

Example 3.7. Consider another example: the timed word w and the PTA pattern $\mathcal{A}$ in Fig. 3.3. Fig. 3.4 shows the result of MakeSymbolic and TW2PTA. The result of $\mathsf{PTPM}(\mathcal{A}, w)$ is as follows:

$$\begin{aligned} & 1.7 < \mathsf{t} < 2.8 - \mathsf{p}_1 \wedge 4.9 \leq \mathsf{t}' < 5.3 \wedge \mathsf{p}_2 > 1.2 \\ \vee\ & 2.8 < \mathsf{t} < 3.7 - \mathsf{p}_1 \wedge 5.3 \leq \mathsf{t}' < 6 \wedge \mathsf{p}_2 > 1.2 \\ \vee\ & 3.7 < \mathsf{t} < 4.9 - \mathsf{p}_1 \wedge \mathsf{t}' \geq 6 \wedge \mathsf{p}_2 > 0.7 \end{aligned}$$

Observe that, for the parameter valuation v with $v(p_1) = v(p_2) = 1$, only the pattern corresponding to the last disjunct could be obtained, i. e., the pattern that matches the last three a of the timed word in Fig. 3.3b. In contrast, the first disjunct can match the first three a coming after the two bs, while the second disjunct allows to match the three as in the middle of the last five as in Fig. 3.3b.

We give various projections of this constraint onto two dimensions in Fig. 3.5 (the difference between plain red and light red is not significant—light red constraints denote unbounded constraints towards at least one dimension). ◇

3.3.3 Termination

We state below the termination of our procedure.

Lemma 3.8 (termination)**.** *Let $\mathcal{A}$ be a PTA encoding a parametric pattern, and w be a timed word. Then $\mathsf{PTPM}(\mathcal{A}, w)$ terminates.*

Proof. First, observe that there may be non-determinism in the pattern PTA, i. e., the timed word can potentially synchronize with two transitions labeled with the same action from a given location. Even if there is no syntactic nondeterminism, nondeterminism can appear due to the interleaving of the initial start action: in Fig. 3.4, the first a of the timed word can either synchronize with the edge from ℓ_0'' to ℓ_0', or the start action can first occur, and then the first a synchronizes of the timed word with the a labeling the transition from ℓ_0 to ℓ_1 of the pattern PTA. Second, the pattern PTA may well have loops (and this is the case in our experiments in Section 3.3.5.3), which yields an infinite parametric zone graph (for the pattern automaton not synchronized with the word automaton). However, let us show that only a finite part of the parametric zone graph is explored by EFsynth: indeed, since $\mathsf{TW2PTA}(w)$ is only a finite sequence, and thanks to the strong synchronization between the pattern PTA and the timed word PTA, only a finite number of finite discrete paths in the synchronized product will be explored. The only interleaving is due to the initial start action (which appears twice in the pattern PTA but can only be taken once at most due to the mutually exclusive guards $x = 0$ and $x > 0$), and due to the final \$ action, that only appear on the last transition to the last-but-one accepting location. As the pattern PTA is finitely branching, this gives a finite number of finite paths. The length of each path is clearly bounded by $|w| + 3$. Let us now consider the maximal number of such paths: given a location in $\mathsf{TW2PTA}(w)$, the choice of the action (say a) is entirely deterministic. However, the pattern PTA may be non-deterministic, and can synchronize with B outgoing transitions labeled with a, which gives $B^{|w|}$ combinations. In addition, the start action can be inserted exactly once, at any position in the timed word (from before the first action to after the last action of the word—in the case of an empty pattern): this gives therefore $(|w| + 1) \times B^{|w|}$ different runs. (The \$ is necessarily the last-but-one action, and does not impact the number of runs, as the (potential) outgoing transitions from the accepting location are not explored.) Altogether, a total number of at most $(|w| + 3) \times (|w| + 1) \times B^{|w|}$ symbolic states is explored by EFsynth in the worst case. □

Lemma 3.8 may not come as a surprise, as the input timed word is finite. But it is worth noting that it comes in contrast with the fact that the wide majority of decision problems are undecidable for parametric timed automata, including the emptiness of the valuation set for which a given location is reachable both, for integer- and rational-valued parameters [AHV93, Mil00] (see [And19] for a survey).

3.3.4 Pattern matching with optimization

We also address the following optimization problem: given a timed word and a pattern containing parameters, what is the minimum or maximum value of a given parameter such that the pattern is matched by the timed word? That is, we are only interested in the optimal value of the given parameter, and not in the full list of matches as in PTPM.

While this problem can be solved using our solution from Section 3.3.1 (by computing the multidimensional constraint, and then eliminating all parameters but the target parameter, using variable elimination techniques), we use here a dedicated approach, with the hope it is more efficient. Instead of managing all symbolic matches (i. e., a finite union of polyhedra), we simply manage the current optimum; in addition, we cut branches that cannot improve the optimum, with the hope to reduce the number of states explored. For example, assume

parameter p is to be minimized; if the current minimum is $p > 2$, and if a branch is such that $p \geq 3$, then this branch will not improve the minimum, and can safely be discarded. Let $\mathsf{PTPM}_{\mathsf{opt}}$ denote this procedure.

Example 3.9. Consider again the timed word w and the PTA pattern $\mathcal{A}$ in Fig. 3.3. Minimizing p_2 so that the pattern matches the timed word for at least one position gives $\mathsf{p}_2 > 0.7$, while maximizing p_1 gives $\mathsf{p}_1 < 1.2$. ◇

3.3.5 Experiments

We evaluated our approach against two standard benchmarks from [HAF14], already used in [WHS17], as well as a third benchmark specifically designed to test the limits of parametric timed pattern matching. We fixed no bounds for our parameters.

We used IMITATOR [AFKS12] to perform the parameter synthesis (algorithm EFsynth). IMITATOR relies on the Parma Polyhedra Library (PPL) [BHZ08] to compute symbolic states. It was shown in [BFMU17] that polyhedra may be dozens of times slower than more efficient data structures such as DBMs (difference bound matrices); however, for parametric analyses, DBMs are not suitable, and parameterized extensions (e. g., in [HRSV02]) still need polyhedra in their representation.

We used a slightly modified version of IMITATOR for technical reasons: IMITATOR handles non-convex constraints (finite unions of polyhedra); while most case studies solved by IMITATOR in the past handle simple constraints (made of a few disjuncts), the experiments in this chapter may handle up to *dozens of thousands* of such polyhedra. We therefore had to disable an inclusion test of a newly computed state into the already computed constraint: this test usually has a very interesting gain but, on our complex polyhedra, it had disastrous impact on the performance, due to the inclusion check of a (simple) new convex polyhedron into a disjunction of dozens of thousands of convex polyhedra. To disable this check, we added a new option (not set by default) to the master branch of IMITATOR, and used it in all our experiments.

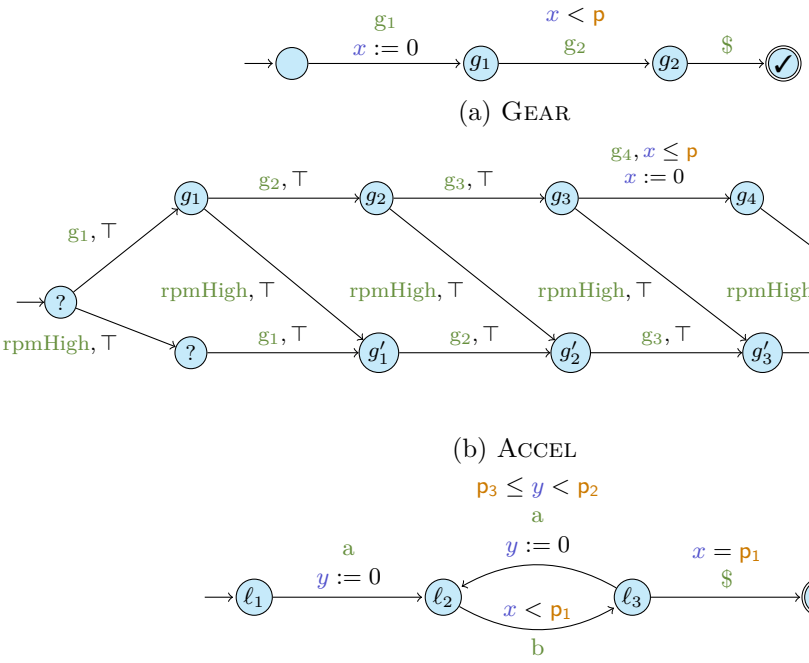
We wrote a simple Python script to implement the TW2PTA procedure; the patterns (Fig. 4.3) were manually transformed following the MakeSymbolic procedure, and converted into the input language of IMITATOR.

We ran experiments using IMITATOR 2.10.4 "Butter Jellyfish" on a Dell Precision 3620 i7-7700 3.60 GHz with 64 GiB memory running Linux Mint 19 beta 64 bits. Sources, binaries, models, logs can be found at `www.imitator.fr/static/ICECCS18`.

3.3.5.1 Gear

Benchmark Gear is inspired by the scenario of monitoring the gear change of an automatic transmission system. We conducted simulation of the model of an automatic transmission system [HAF14]. We used S-TaLiRo [ALFS11] to generate an input to this model; it generates a gear change signal that is fed to the model. A gear is chosen from $\{g_1, g_2, g_3, g_4\}$. The generated gear change is recorded in a timed word. The set W consists of 10 timed words; the length of each word is 1,467 to 14,657.

The pattern PTA $\mathcal{A}$, shown in Fig. 3.6a, detects the violation of the following condition: If the gear is changed to 1, it should not be changed to 2 within p seconds. This condition is related to the requirement ϕ_5^{AT} proposed in [HAF14] (the nominal value for p in [HAF14] is 2).



(a) Gear

(b) Accel

(c) Blowup

Figure 3.6: Experiments: patterns

Model		PTPM				PTPM$_{opt}$	
Length	Time frame	States	Matches	Parsing (s)	Comp. (s)	States	Comp. (s)
1,467	1,000	4,453	379	0.02	1.60	3,322	0.94
2,837	2,000	8,633	739	0.33	2.14	6,422	1.70
4,595	3,000	14,181	1,247	0.77	3.63	10,448	2.85
5,839	4,000	17,865	1,546	1.23	4.68	13,233	3.74
7,301	5,000	22,501	1,974	1.94	5.88	16,585	4.79
8,995	6,000	27,609	2,404	2.96	7.28	20,413	5.76
10,316	7,000	31,753	2,780	4.00	8.38	23,419	6.86
11,831	8,000	36,301	3,159	5.39	9.75	26,832	7.87
13,183	9,000	40,025	3,414	6.86	10.89	29,791	8.61
14,657	10,000	44,581	3,816	8.70	12.15	33,141	9.89

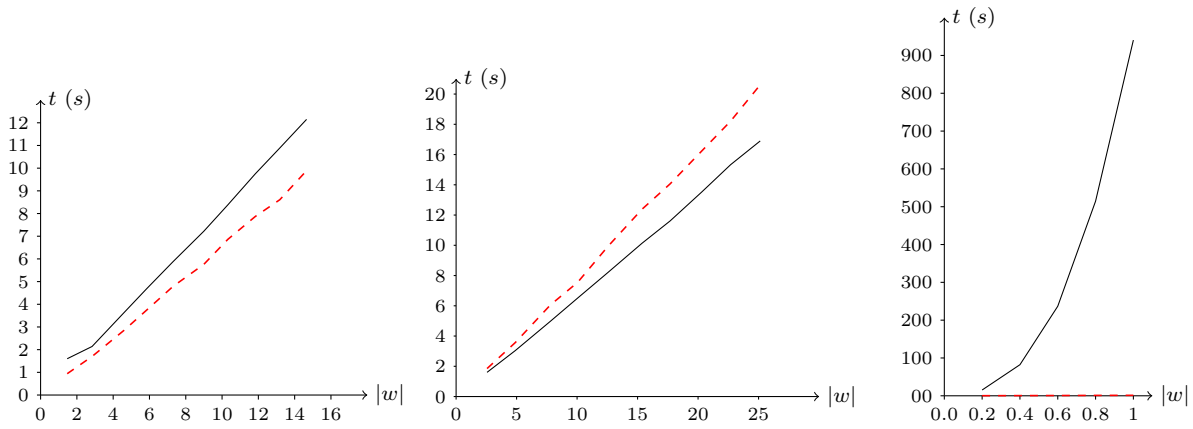
Table 3.1: Experiments: Gear

We tabulate our experiments in Table 3.1. We give from left to right the length of the timed word in terms of actions and time, then the data for PTPM (the number of symbolic states explored, the number of (symbolic) matches found, the parsing time and the computation time excluding parsing) and for PTPM$_{opt}$ (number of symbolic states explored and computation time) using IMITATOR. The parsing time for PTPM$_{opt}$ is almost identical to PTPM and is therefore omitted.

The corresponding chart is given in Fig. 3.7a (PTPM is given in plain black, and PTPM$_{opt}$ in red dashed). PTPM$_{opt}$ brings a gain in terms of memory (symbolic states) of about 25 %, while the gain in time is about 20 %.

3.3.5.2 Accel

The W of benchmark Accel is also constructed from the Simulink model of the automated transmission system [HAF14]. For this benchmark, the (discretized) value of three state variables are recorded in W: engine RPM (discretized to "high" and "low" with a certain thresh-



(a) Gear (b) Accel (c) Blowup

Figure 3.7: Experiments: charts (x-scale $\times 1,000$)

Model		PTPM				PTPM_{opt}	
Length	Time frame	States	Matches	Parsing (s)	Comp. (s)	States	Comp. (s)
2,559	1,000	6,504	2	0.27	1.60	6,502	1.85
4,894	2,000	12,429	2	0.86	3.04	12,426	3.57
7,799	3,000	19,922	7	2.21	4.98	19,908	6.06
10,045	4,000	25,520	3	3.74	6.51	25,514	7.55
12,531	5,000	31,951	9	6.01	8.19	31,926	9.91
15,375	6,000	39,152	7	9.68	10.14	39,129	12.39
17,688	7,000	45,065	9	13.40	11.61	45,039	14.06
20,299	8,000	51,660	10	18.45	13.52	51,629	16.23
22,691	9,000	57,534	11	24.33	15.33	57,506	18.21
25,137	10,000	63,773	13	31.35	16.90	63,739	20.61

Table 3.2: Experiments: Accel

old), velocity (discretized to "high" and "low" with a certain threshold), and 4 gear positions. We used Breach [Don10] to generate an input sequence of gear change. Our set W consists of 10 timed words; the length of each word is 2,559 to 25,137.

The pattern PTA $\mathcal{A}$ of this benchmark is shown in Fig. 3.6b. This pattern matches a part of a timed word that violates the following condition: If a gear changes from 1 to 2, 3, and 4 in this order in p seconds and engine RPM becomes large during this gear change, then the velocity of the car must be sufficiently large in one second. This condition models the requirement ϕ_8^{AT} proposed in [HAF14] (the nominal value for p in [HAF14] is 10).

Experiments are tabulated in Table 3.2. The corresponding chart is given in Fig. 3.7b. This time, PTPM_{opt} brings almost no gain in terms of states, and a loss of speed of about 15 to 20 %, which may come from the additional polyhedra inclusion checks to test whether a branch is less good than the current optimum.

3.3.5.3 Blowup

As a third experiment, we considered an original (toy) benchmark that acts as the worst case situation for parametric timed pattern matching. Consider the PTA pattern in Fig. 3.6c, and assume a timed word consisting in an alternating sequence of "a" and "b". Observe that the time from the pattern beginning (that resets x) to the end is exactly p_1 time units. Also observe that the duration of the loop through ℓ_2 and ℓ_3 has a duration in $[\mathsf{p}_3, \mathsf{p}_2)$; therefore, for values sufficiently small of $\mathsf{p}_2, \mathsf{p}_3$, one can always match a larger number of loops. That is, for a timed word of length $2n$ alternating between "a" and "b", there will be n possible

Model		PTPM				PTPM$_{opt}$	
Length	Time frame	States	Matches	Parsing (s)	Comp. (s)	States	Comp. (s)
200	101	20,602	5,050	0.01	15.31	515	0.24
400	202	81,202	20,100	0.02	82.19	1,015	0.49
600	301	181,802	45,150	0.03	236.80	1,515	0.71
800	405	322,402	80,200	0.05	514.57	2,015	1.05
1,000	503	503,002	125,250	0.06	940.74	2,515	1.24

Table 3.3: Experiments: Blowup

matches from position 0 (with n different parameter constraints), $n-1$ from position 1, and so on, giving a total number of $\frac{n(n+1)}{2}$ matches with different constraints in 5 dimensions.

Note that this worst case situation is not specific to our approach, but would appear independently of the approach chosen for parametric timed pattern matching.

We generated random timed words of various sizes, all alternating exactly between "a" and "b". Our set W consists of 5 timed words of length from 200 to 1,000.

Experiments are tabulated in Table 3.3. The corresponding chart is given in Fig. 3.7c. PTPM becomes clearly non-linear as expected. This time, PTPM$_{opt}$ brings a dramatic gain in both memory and time; even more interesting, PTPM$_{opt}$ remains perfectly linear.

3.3.6 Discussion

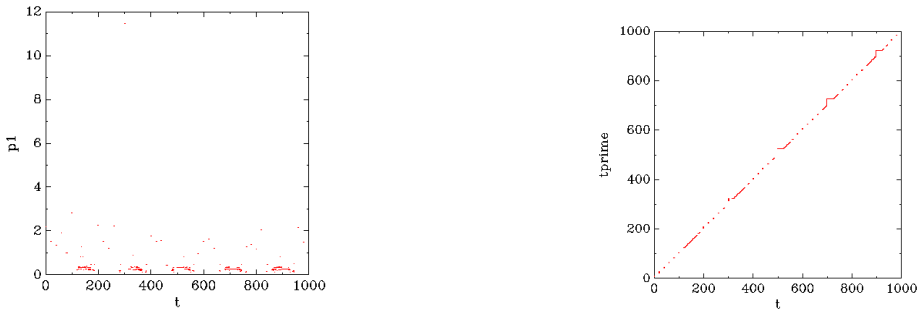
A first positive outcome is that our method is effectively able to perform parametric pattern matching on words of length up to several dozens of thousands, and is able to output results in the form of several dozens of thousands of symbolic matches in several dimensions, in just a few seconds. Another positive outcome is that PTPM is perfectly linear in the size of the input word for Gear and Accel: this was expected as these examples are linear, in the sense that the number of states explored by PTPM is linear as these patterns feature no loops.

Note that the parsing time is not linear, but it could be highly improved: due to the relatively small size of the models usually treated by IMITATOR, this part was never properly optimized, and it contains several quadratic syntax checking functions that could easily be avoided.

The performances do not completely allow yet for an *online* usage in the current version of our algorithm and implementation (in [WHS17], we pushed the Accel case study for timed words of length up to 17,280,002). A possible direction is to perform and on-the-fly computation of the parametric zone graph, more precisely to do an on-the-fly parsing of the timed word automaton; this will allow IMITATOR to keep in memory a single location at a time (instead of up to 25,137 in our experiments).

Finally, although this is not our original motivation, we believe that, if we are only interested in *robust* pattern matching, i. e., non-parametric pattern matching but with an allowed deviation ("guard enlargement") of the pattern automaton, then using the efficient 1-dimensional parameterized DBMs of [San15] would probably be an interesting alternative: indeed, in contrast to classical parameterized DBMs [HRSV02] (that are made of a matrix and a parametric polyhedron), the structure of [San15] only needs an $H \times H$ matrix with a single parameter, and seems particularly efficient.

Remark 3.10. In the conference version of this work [AHW18], we describe this approach as an *offline* algorithm. In fact, it is essentially *online* in the sense that it can potentially run with only a portion of the log: it relies on parallel composition of a specification automaton and a log automaton, and this parallel composition can be achieved on-the-fly. However, as mentioned in [BDD+18], "a good online monitoring algorithm must: *i*) be able to generate intermediate



(a) Projection onto t and p (b) Projection onto t and t'

Figure 3.8: Visualizing many matches for GEAR ($|w = 1467|$)

estimates of property satisfaction based on partial signals, *ii*) use minimal amount of data storage, and *iii*) be able to run fast enough in a real-time setting." So, at least for point iii, our algorithm may not really run in a real-time setting.

In contrast, we will present in the next section a contribution fast enough to run in a real-time setting, with runs of dozens of thousands of events being analyzable in less than a second.

3.4 Algorithm II: Direct method by polyhedra computation

In this section, we present a *dedicated* online algorithm for parametric timed pattern matching. We will then enhance it with *skipping* in Section 3.4.1, and experimentally evaluate both versions with and without skipping in Section 3.4.2.

Similarly to the online algorithm for timed pattern matching in Algorithm 2 (Section 2.4), our algorithm finds all the matching triples $(t, t', v) \in \mathcal{M}(w, \mathcal{A})$ by a breadth-first search. Our algorithm is online in the following sense: after reading the i-th element (a_i, τ_i) of the timed word $w = (\overline{a}, \overline{\tau})$, it immediately outputs all the matching triples (t, t', v) over the available prefix $(a_1, \tau_1), (a_2, \tau_2), \dots, (a_i, \tau_i)$ of w. See Definition 2.17 for the auxiliaries $\mathrm{eval}(\rho, \tau)$, $\mathrm{reset}(\rho, R, \tau)$, and $\rho_\emptyset$ as well as the usage of $\rho \colon \mathbb{X} \to (\mathbb{R}_{\geq 0} \amalg \{t\})$ to represent the latest reset time of each clock variable.

Algorithm 4 shows our online algorithm for parametric timed pattern matching. In the pseudocode, we used *CurrConf*, *PrevConf*, and Z. *CurrConf* and *PrevConf* are finite sets of triples $(\ell, \rho, \mathcal{C})$ made of a location $\ell \in L$, a mapping $\rho \colon \mathbb{X} \to (\mathbb{R}_{\geq 0} \amalg \{t\})$ denoting the latest reset of each clock, and a constraint $\mathcal{C}$ over $\mathbb{P}\amalg\{t\}$. Z is a finite set of constraints over $\mathbb{P}\amalg\{t, t'\}$. As a running example, we use the PTA and the timed word in Fig. 3.1.

At first, the counter i is 1 (line 2), and we start the matching trial from $t \in [\tau_0, \tau_1)$. At line 4, we add the new configuration $(\ell_0, \rho_\emptyset, (\tau_0 \leq t < \tau_1))$ to *CurrConf*, which means we are at the initial location ℓ_0, we have no reset of the clock variables yet, and we can potentially start the matching from any $t \in [\tau_0, \tau_1)$. In lines 5 to 8, we try to insert \$(i. e., the end of the matching) in $(\tau_0, \tau_1]$; in our running example in Fig. 3.1, since there is no edge from ℓ_0 to the accepting state, we immediately jump to line 9. Then, in lines 10 to 14, we consume $(a_1, \tau_1) = (a, 0.7)$ and try to transit from ℓ_0 to ℓ_1. The guard $x > 1$ at the edge from ℓ_0 to ℓ_1 is examined at line 12. We take the conjunction of the current constraint $\mathcal{C}$, the guard g, and the constraints $\mathrm{eval}(\rho, \tau_i)$ on the clock valuations. We take the projection to $\mathbb{P} \amalg \{t\}$ because the constraint on the clock variables changes after time passing. Since no clock variable is reset so far, the constraint on the clock valuation is $x = \tau_1 - t$. The constraint

Algorithm 4: Online parametric timed pattern matching without skipping

Input: A timed word $w = (\overline{a}, \overline{\tau})$, and a PTA $\mathcal{A} = (\Sigma, L, L_0, L_F, \mathbb{X}, \mathbb{P}, E)$.
Output: $\bigvee Z$ is the match set $\mathcal{M}(w, \mathcal{A})$

1 $\mathit{CurrConf} \leftarrow \emptyset$; $Z \leftarrow \emptyset$
2 **for** $i \leftarrow 1$ **to** $|w|$ **do**
3 **for** $\ell \in L_0$ **do**
4 **push** $(\ell, \rho_\emptyset, (\tau_{i-1} \le t < \tau_i))$ **to** $\mathit{CurrConf}$
5 **for** $(\ell, \rho, \mathcal{C}) \in \mathit{CurrConf}$ **do** // lines 5 to 8 try to insert $ in $(\tau_{i-1}, \tau_i]$.
6 **for** $\ell_f \in L_F$ **do**
7 **for** $(\ell, g, \$, R, \ell_f) \in E$ **do**
8 **push** $(\mathcal{C} \wedge (\tau_{i-1} < t' \le \tau_i) \wedge g \wedge \mathrm{eval}(\rho, t'))\downarrow_{\mathbb{PI}\{t,t'\}}$ **to** Z
9 $(\mathit{PrevConf}, \mathit{CurrConf}) \leftarrow (\mathit{CurrConf}, \emptyset)$
10 **for** $(\ell, \rho, \mathcal{C}) \in \mathit{PrevConf}$ **do** // lines 10 to 14 try to go forward using (a_i, τ_i).
11 **for** $(\ell, g, a_i, R, \ell') \in E$ **do**
12 $\mathcal{C}' \leftarrow (\mathcal{C} \wedge g \wedge \mathrm{eval}(\rho, \tau_i))\downarrow_{\mathbb{PI}\{t\}}$
13 **if** $\mathcal{C}' \neq \bot$ **then**
14 **push** $(\ell', \mathrm{reset}(\rho, R, \tau), \mathcal{C}')$ **to** $\mathit{CurrConf}$
15 **push** $(\ell_0, \rho_\emptyset, \{\tau_{|w|} \le t < \infty\})$ **to** $\mathit{CurrConf}$ // for the trimming after the final event
16 **for** $(\ell, \rho, \mathcal{C}) \in \mathit{CurrConf}$ **do** // lines 16 to 19 try to insert $ in $(\tau_{|w|}, \infty)$.
17 **for** $\ell_f \in L_F$ **do**
18 **for** $(\ell, g, \$, R, \ell_f) \in E$ **do**
19 **push** $(\mathcal{C} \wedge (\tau_{|w|} < t' < \infty) \wedge g \wedge \mathrm{eval}(\rho, t'))\downarrow_{\mathbb{PI}\{t,t'\}}$ **to** Z

$\mathcal{C} \wedge g \wedge \mathrm{eval}(\rho, \tau_1) = (0 \le t < 0.7) \wedge (x > 1) \wedge (x = 0.7 - t)$ is unsatisfiable, and we go back to line 4.

At line 4, we add the new configuration $(\ell_0, \rho_\emptyset, (\tau_1 \le t < \tau_2))$ to $\mathit{CurrConf}$. Similarly, we immediately jump to line 9, and we try the edge from ℓ_0 to ℓ_1 in lines 10 to 14. This time, the constraint $\mathcal{C} \wedge g \wedge \mathrm{eval}(\rho, \tau_2) = (0.7 \le t < 2.0) \wedge (x > 1) \wedge (x = 2.0 - t)$ is satisfiable at line 13, and we push the next configuration $(\ell_1, \rho_\emptyset, \mathcal{C}')$ to $\mathit{CurrConf}$ at line 14.

Similarly, we keep adding and updating configurations until the end of the input timed word w. Finally, in lines 16 to 19, we try to insert \$in $(\tau_3, \infty) = (4.1, \infty)$. We can use the edge from ℓ_2 to the accepting state, and we add the constraint at the right of Fig. 3.1 to Z.

Algorithm 4 terminates because the size of $\mathit{CurrConf}$ is always finite. Algorithm 4 is correct because it symbolically keeps track of all the runs of $v(\mathcal{A})$ over $w|_{(t,t')}$ for any $v \in (\mathbb{Q}_+)^{\mathbb{P}}$ and $(t, t') \subseteq \mathbb{R}_{\ge 0}$.

3.4.1 Skipping enhanced parametric timed pattern matching

We now enhance Algorithm 4 with automata-based skipping. By using skipping, at line 2 of Algorithm 4, we can increase the counter i by the *skip value*. The skip value can be more than 1, and some matching trials may be prevented. A large part of the skip value computation can be reused and the whole algorithm can become faster. Following [WHS17], we employ *FJS-style* skipping [FJS07]. An FJS-style skipping consists of two skip value functions: the KMP-style skip value function Δ_{KMP} [KJP77] and the Quick Search-style skip value function Δ_{QS} [Sun90]. We note that this optimization does not change the result thanks to the soundness theorems (Theorems 3.12 and 3.15).

The following are auxiliaries for the skip values. For a PTA $\mathcal{A}$ and a parameter valuation v, the language without the last element is denoted by $\mathcal{L}_{-\$}(v(\mathcal{A})) = \{w(1,|w|-1) \mid w \in \mathcal{L}(v(\mathcal{A}))\}$. For a PTA $\mathcal{A} = (\Sigma, L, L_0, L_F, \mathbb{X}, \mathbb{P}, E)$ and $\ell \in L$, $\mathcal{A}_\ell$ denotes the PTA $\mathcal{A}_\ell = (\Sigma, L, L_0, \{\ell\}, \mathbb{X}, \mathbb{P}, E)$.

3.4.1.1 KMP-style skip values

Given a location $\ell \in L$ and a set $V \subseteq (\mathbb{Q}_+)^{\mathbb{P}}$ of parameter valuations, the *KMP-style skip value function* Δ_{KMP} returns the skip value $\Delta_{\mathrm{KMP}}(\ell, V) \in \mathbb{Z}_{>0}$. The location ℓ and the parameter valuations V present one of the configurations in the previous matching trial. We utilize the pair (ℓ, V) to overapproximate the subsequence $w(i,j)$ of the timed word w examined in the latest matching trial.

Definition 3.11 (Δ_{KMP})**.** Let $\mathcal{A}$ be a PTA $\mathcal{A} = (\Sigma, L, L_0, L_F, \mathbb{X}, \mathbb{P}, E)$. For a location $\ell \in L$ and $n \in \mathbb{Z}_{>0}$, let $V_{\ell,n}$ be the set of parameter valuations v such that there is a parameter valuation $v' \in (\mathbb{Q}_+)^{\mathbb{P}}$ satisfying $\mathcal{L}(v(\mathcal{A}_\ell)) \cdot \mathcal{T}(\Sigma) \cap \mathcal{T}^n(\Sigma) \cdot \{w'' + t \mid w'' \in \mathcal{L}_{-\$}(v'(\mathcal{A})), t \geq 0\} \cdot \mathcal{T}(\Sigma) \neq \emptyset$. The *KMP-style skip value function* $\Delta_{\mathrm{KMP}} \colon L \times \mathcal{P}((\mathbb{Q}_+)^{\mathbb{P}}) \to \mathbb{Z}_{>0}$ is $\Delta_{\mathrm{KMP}}(\ell, V) = \min\{n \in \mathbb{Z}_{>0} \mid V \subseteq V_{\ell,n}\}$. ◇

Intuitively, the soundness of the skipping with Δ_{KMP} is as follows. Let ℓ be a location we reached in the end of the matching trial from $i \in \{1, 2, \ldots, |w|\}$ for the parameter valuation v. We overapproximate the subsequence $w(i,|w|)$ by the language $\mathcal{L}(v(\mathcal{A}_\ell)) \cdot \mathcal{T}(\Sigma)$. For $n \in \mathbb{Z}_{>0}$, we overapproximate the matching from $i+n$ by $\bigcup_{v' \in (\mathbb{Q}_+)^{\mathbb{P}}} \mathcal{T}^n(\Sigma) \cdot \{w'' + t \mid w'' \in \mathcal{L}_{-\$}(v'(\mathcal{A})), t \geq 0\} \cdot \mathcal{T}(\Sigma)$. Because of these overapproximation, we have no matching from $i+n$ if $v \notin V_{l,n}$ holds. Overall, we can skip the matching trials from $i+1, i+2, \ldots, i+\Delta_{\mathrm{KMP}}(\ell, \{v\}) - 1$. We note that if we reached both ℓ and ℓ', the intersection $(\mathcal{L}(v(\mathcal{A}_\ell)) \cdot \mathcal{T}(\Sigma)) \cap (\mathcal{L}(v(\mathcal{A}_{\ell'})) \cdot \mathcal{T}(\Sigma))$ is an overapproximation of $w(i,|w|)$, and therefore, we take the maximum of $\Delta_{\mathrm{KMP}}(\ell, V)$ over the reached configurations.

Since $V_{\ell,n}$ is independent of the timed word w, we can compute it before the matching trials. The computation of $V_{\ell,n}$ is by the reachability synthesis of PTAs constructed from $\mathcal{A}$. See Appendix A for the construction of the PTAs. During the matching trials, only the inclusion checking $V \subseteq V_{\ell,n}$ is necessary. This test can be achieved thanks to convex polyhedra inclusion.

Theorem 3.12 (soundness)**.** *Let $\mathcal{A} = (\Sigma, L, L_0, L_F, \mathbb{X}, \mathbb{P}, E)$ be a PTA and let $w \in \mathcal{T}(\Sigma)$. For any subsequence $w(i,j)$ of w and for any $(\ell, v) \in L \times (\mathbb{Q}_+)^{\mathbb{P}}$, if there exists $t \in \mathbb{R}_{\geq 0}$ satisfying $w(i,j) - t \in \mathcal{L}(v(\mathcal{A}_\ell))$, for any $n \in \{1, 2, \ldots, \Delta_{\mathrm{KMP}}(\ell, \{v\}) - 1\}$, we have $((\tau_{i+n-1}, \tau_{i+n}] \times \mathbb{R}_{\geq 0} \times (\mathbb{Q}_+)^{\mathbb{P}}) \cap \mathcal{M}(w, \mathcal{A}) = \emptyset$.* □

First, we prove the following lemma.

Lemma 3.13. *Let $\mathcal{A}$ be a PTA $\mathcal{A} = (\Sigma, L, L_0, L_F, \mathbb{X}, \mathbb{P}, E)$ and let $w \in \mathcal{T}(\Sigma)$. For a subsequence $w(i,j)$ of w, let $C_{i,j} \subseteq L \times (\mathbb{Q}_+)^{\mathbb{P}}$ be $C_{i,j} = \{(\ell, v) \mid \exists t \in \mathbb{R}_{\geq 0}.\, w(i,j) - t \in \mathcal{L}(v(\mathcal{A}_\ell))\}$. For a subsequence $w(i,j)$ of w and $n \in \mathbb{Z}_{>0}$, if there exists $(\ell, v) \in C_{i,j}$ satisfying $v \notin V_{\ell,n}$, we have $([\tau_{i+n-1}, \tau_{i+n}) \times \mathbb{R}_{\geq 0} \times (\mathbb{Q}_+)^{\mathbb{P}}) \cap \mathcal{M}(w, \mathcal{A}) = \emptyset$.* □

Proof. If there exists $(\ell, v) \in C_{i,j}$ satisfying $v \notin V_{\ell,n}$, by the definition of $C_{i,j}$ and $V_{\ell,n}$, such $(\ell, v) \in L \times (\mathbb{Q}_+)^{\mathbb{P}}$ satisfies the following.

- $\exists t \in \mathbb{R}_{\geq 0}.\, w(i,j) - t \in \mathcal{L}(v(\mathcal{A}_\ell))$

- $\forall v' \in (\mathbb{Q}_+)^{\mathbb{P}}. \mathcal{L}(v(\mathcal{A}_\ell)) \cdot \mathcal{T}(\Sigma) \cap \mathcal{T}^n(\Sigma) \cdot \{w'' + t \mid w'' \in \mathcal{L}_{-\$}(v'(\mathcal{A})), t > 0\} \cdot \mathcal{T}(\Sigma) = \emptyset$

Therefore, we have the following.

$$
\begin{aligned}
&\exists (\ell, v) \in C_{i,j}. v \notin V_{\ell,n} \\
\Rightarrow &\exists t \in \mathbb{R}_{\geq 0}. \forall v' \in (\mathbb{Q}_+)^{\mathbb{P}}. \\
&(w(i,j) - t) \cdot \mathcal{T}(\Sigma) \cap \mathcal{T}^n(\Sigma) \cdot \{w'' + t \mid w'' \in \mathcal{L}_{-\$}(v'(\mathcal{A})), t > 0\} \cdot \mathcal{T}(\Sigma) = \emptyset \\
\Rightarrow &\forall v' \in (\mathbb{Q}_+)^{\mathbb{P}}. \\
&w(i,j) \cdot \mathcal{T}(\Sigma) \cap \mathcal{T}^n(\Sigma) \cdot \{w'' + t \mid w'' \in \mathcal{L}_{-\$}(v'(\mathcal{A})), t > 0\} \cdot \mathcal{T}(\Sigma) = \emptyset \\
\Rightarrow &\forall v' \in (\mathbb{Q}_+)^{\mathbb{P}}. w(i, |w|) \notin \mathcal{T}^n(\Sigma) \cdot \{w'' + t \mid w'' \in \mathcal{L}_{-\$}(v'(\mathcal{A})), t > 0\} \cdot \mathcal{T}(\Sigma) \\
\Rightarrow &\forall v' \in (\mathbb{Q}_+)^{\mathbb{P}}. w(i, |w|) \notin \mathcal{T}^n(\Sigma) \circ \{w'' + t \mid w'' \in \mathcal{L}_{-\$}(v'(\mathcal{A})), t \in [\tau_{i+n-1}, \tau_{i+n})\} \cdot \mathcal{T}(\Sigma) \\
\Rightarrow &\forall v' \in (\mathbb{Q}_+)^{\mathbb{P}}. w(i+n, |w|) \notin \{w'' + t \mid w'' \in \mathcal{L}_{-\$}(v'(\mathcal{A})), t \in [\tau_{i+n-1}, \tau_{i+n})\} \cdot \mathcal{T}(\Sigma) \\
\iff &\forall t \in [\tau_{i+n-1}, \tau_{i+n}), v' \in (\mathbb{Q}_+)^{\mathbb{P}}. w(i+n, |w|) - t \notin \{w'' \mid w'' \in \mathcal{L}_{-\$}(v'(\mathcal{A}))\} \cdot \mathcal{T}(\Sigma) \\
\iff &\forall t \in [\tau_{i+n-1}, \tau_{i+n}), t' > t, v' \in (\mathbb{Q}_+)^{\mathbb{P}}. w|_{(t,t')} \notin \mathcal{L}(v'(\mathcal{A})) \\
\iff &([\tau_{i+n-1}, \tau_{i+n}) \times \mathbb{R}_{\geq 0} \times (\mathbb{Q}_+)^{\mathbb{P}}) \cap \mathcal{M}(w, \mathcal{A}) = \emptyset
\end{aligned}
$$

□

Then, the proof of Theorem 3.12 is as follows.

Proof. By definition of $C_{i,j}$ in Lemma 3.13, for any subsequence $w(i,j)$ of w and $(\ell, v) \in L \times (\mathbb{Q}_+)^{\mathbb{P}}$, $\exists t \in \mathbb{R}_{\geq 0}. w(i,j) - t \in \mathcal{L}(v(\mathcal{A}_\ell))$ implies $(\ell, v) \in C_{i,j}$. By definition of Δ_{KMP}, for any $(\ell, v) \in C_{i,j}$ and $n \in \{1, 2, \ldots, \Delta_{\mathrm{KMP}}(\ell, \{v\}) - 1\}$, we have $v \notin V_{i,n}$. Therefore, Theorem 3.12 holds because of Lemma 3.13. □

Although $V_{l,n}$ can be computed before the matching trials, the KMP-style skip value function Δ_{KMP} requires checking $V \subseteq V_{l,n}$ after each matching trial, which means a polyhedral inclusion test in $|\mathbb{P}| + 2$ dimensions. To reduce this runtime overhead, we define the *non-parametric* KMP-style skip value function $\Delta'_{\mathrm{KMP}}(\ell) = \min_{v \in (\mathbb{Q}_+)^{\mathbb{P}}} \Delta_{\mathrm{KMP}}(\ell, \{v\})$. For comparison, we refer Δ_{KMP} as the *parametric* KMP-style skip value function.

3.4.1.2 Quick Search-style skip values

Given an action $a \in \Sigma$, the *Quick Search-style skip value function* Δ_{QS} returns the skip value $\Delta_{\mathrm{QS}}(a) \in \mathbb{Z}_{>0}$. Before the matching trial from the i-th element (a_i, τ_i), we look ahead the action a_{i+N-1}, where N is the length of the shortest matching. If we observe that there is no potential matching, we also look ahead the action a_{i+N} and skip by $\Delta_{\mathrm{QS}}(a_{i+N})$. The construction of the Quick Search-style skip value function Δ_{QS} is by reachability emptiness of PTAs, i. e., the emptiness of the valuation set reaching a given location.

Definition 3.14 (Δ_{QS})**.** For a PTA $\mathcal{A} = (\Sigma, L, L_0, L_F, \mathbb{X}, \mathbb{P}, E)$, the *Quick-Search-style skip value function* $\Delta_{\mathrm{QS}} \colon \Sigma \to \mathbb{Z}_{>0}$ is as follows, where $\mathrm{Untimed}(\mathcal{L}_{-\$}(v(\mathcal{A})))$ is the untimed projection of the language $\mathcal{L}_{-\$}(v(\mathcal{A}))$ and $N \in \mathbb{Z}_{>0}$ is $N = \min\{|w| \mid w \in \bigcup_{v \in (\mathbb{Q}_+)^{\mathbb{P}}} \mathcal{L}_{-\$}(v(\mathcal{A}))\}$.

$$
\Delta_{\mathrm{QS}}(a) = \min\{n \in \mathbb{Z}_{>0} \mid \exists v \in (\mathbb{Q}_+)^{\mathbb{P}}. \Sigma^N a \Sigma^* \cap \Sigma^n \mathrm{Untimed}(\mathcal{L}_{-\$}(v(\mathcal{A}))) \neq \emptyset\}
$$

◇

The intuition of Δ_{QS} is as follows. For $i \in \{1, 2, \ldots, |w|\}$, the subsequence $w(i, |w|)$ is overapproximated by $\Sigma^N a_{i+N} \Sigma^*$. For $n \in \mathbb{Z}_{>0}$, the matching from $i+n$ is overapproximated by $\bigcup_{v \in (\mathbb{Q}_+)^{\mathbb{P}}} \Sigma^n \mathrm{Untimed}(\mathcal{L}_{-\$}(v(\mathcal{A})))$. Therefore, for any $n \in \{1, 2, \ldots, \Delta_{\mathrm{QS}}(a_{i+N}) - 1\}$, we have no matching from $i+n$, and we can skip these matching trials.

Theorem 3.15. *Let $\mathcal{A}$ be a PTA $\mathcal{A} = (\Sigma, L, L_0, L_F, \mathbb{X}, \mathbb{P}, E)$, let $w = (\overline{a}, \overline{\tau}) \in \mathcal{T}(\Sigma)$, and let $N = \min\{|w| \mid w \in \bigcup_{v \in (\mathbb{Q}_+)^{\mathbb{P}}} \mathcal{L}_{-\$}(v(\mathcal{A}))\}$. For any index $i \in \{1, 2, \ldots, |w|\}$ of w and for any $m \in \{1, 2, \ldots, \Delta_{\mathrm{QS}}(a_{i+N}) - 1\}$, we have $([\tau_{i+m-1}, \tau_{i+m}) \times \mathbb{R}_{\geq 0} \times (\mathbb{Q}_+)^{\mathbb{P}}) \cap \mathcal{M}(w, \mathcal{A}) = \emptyset$.* □

First, we prove the following lemma.

Lemma 3.16. *Let $\mathcal{A}$ be a PTA $\mathcal{A} = (\Sigma, L, L_0, L_F, \mathbb{X}, \mathbb{P}, E)$, let $w = (\overline{a}, \overline{\tau}) \in \mathcal{T}(\Sigma)$, and let $N = \min\{|w| \mid w \in \bigcup_{v \in (\mathbb{Q}_+)^{\mathbb{P}}} \mathcal{L}_{-\$}(v(\mathcal{A}))\}$. For any index $i \in \{1, 2, \ldots, |w|\}$ of w and for any $m \in \{1, 2, \ldots, N\}$, if any $(\overline{a'}, \overline{\tau'}) \in \bigcup_{v \in (\mathbb{Q}_+)^{\mathbb{P}}} \mathcal{L}(v(\mathcal{A}))$ satisfies $a_{i+N} \neq a'_{N-m+1}$, we have $([\tau_{i+m-1}, \tau_{i+m}) \times \mathbb{R}_{\geq 0} \times (\mathbb{Q}_+)^{\mathbb{P}}) \cap \mathcal{M}(w, \mathcal{A}) = \emptyset$.*

Proof. If $a_{i+N} \neq a'_{N-m+1}$ holds for any $(\overline{a'}, \overline{\tau'}) \in \bigcup_{v \in (\mathbb{Q}_+)^{\mathbb{P}}} \mathcal{L}(v(\mathcal{A}))$, we have the following.

$$\mathrm{Untimed}(\{w(i+m, |w|)\}) \not\subseteq \bigcup_{v \in (\mathbb{Q}_+)^{\mathbb{P}}} \mathrm{Untimed}(\mathcal{L}_{-\$}(v(\mathcal{A})))\Sigma^*$$

Lemma 3.16 is proved by the following.

$$\begin{aligned}
&\mathrm{Untimed}(\{w(i+m, |w|)\}) \not\subseteq \bigcup_{v \in (\mathbb{Q}_+)^{\mathbb{P}}} \mathrm{Untimed}(\mathcal{L}_{-\$}(v(\mathcal{A})))\Sigma^* \\
\Rightarrow &\forall t \in [\tau_{i+m-1}, \tau_{i+m}), v \in (\mathbb{Q}_+)^{\mathbb{P}}. w(i+m, |w|) - t \notin \bigcup_{v \in (\mathbb{Q}_+)^{\mathbb{P}}} \mathcal{L}_{-\$}(v(\mathcal{A})) \cdot \mathcal{T}(\Sigma) \\
\Rightarrow &\forall t \in [\tau_{i+m-1}, \tau_{i+m}), t' > t, v \in (\mathbb{Q}_+)^{\mathbb{P}}. w|_{(t,t')} \notin \bigcup_{v \in (\mathbb{Q}_+)^{\mathbb{P}}} \mathcal{L}(v(\mathcal{A})) \\
\iff &([\tau_{i+m-1}, \tau_{i+m}) \times \mathbb{R}_{\geq 0} \times (\mathbb{Q}_+)^{\mathbb{P}}) \cap \mathcal{M}(w, \mathcal{A}) = \emptyset
\end{aligned}$$

□

Then, the proof of Theorem 3.15 is as follows.

Proof. Since $\Sigma^N a \Sigma^* \cap \Sigma^n \mathrm{Untimed}(\mathcal{L}_{-\$}(v(\mathcal{A}))) \neq \emptyset$ holds for any $n \geq N+1$, we have $\Delta_{\mathrm{QS}}(a_{i+n}) - 1 \leq N$. By definition of Δ_{QS}, $m < \Delta_{\mathrm{QS}}(a_{i+N})$ implies the following.

$$\forall v \in (\mathbb{Q}_+)^{\mathbb{P}}. \Sigma^N a_{i+N} \Sigma^* \cap \Sigma^m \mathrm{Untimed}(\mathcal{L}_{-\$}(v(\mathcal{A}))) = \emptyset$$

Therefore, Lemma 3.16 implies Theorem 3.15. □

Algorithm 5 shows an improvement of Algorithm 4 by skipping. After reading the i-th element (a_i, τ_i) of the timed word $w = (\overline{a}, \overline{\tau})$, Algorithm 5 does not immediately output the matching over the available prefix $(a_1, \tau_1), (a_2, \tau_2), \ldots, (a_i, \tau_i)$ of w, but it still outputs the matching before obtaining the entire timed word with some delay. In the loop in lines 4 to 7 of Algorithm 5, we use the Quick-Search-style skip value function Δ_{QS} to avoid unnecessary matching trials. The matching trial in lines 8 to 10 of Algorithm 5 corresponds to the loop in lines 2 to 14 of Algorithm 4. At line 11, it skips using the parametric KMP-style skip value $\Delta_{\mathrm{KMP}}(\ell, V)$. We can employ the non-parametric KMP-style skip value by replacing $\Delta_{\mathrm{KMP}}(\ell, V)$ with $\Delta'_{\mathrm{KMP}}(\ell)$.

Algorithm 5: Parametric timed pattern matching with parametric skipping

Input: A timed word w and a PTA $\mathcal{A} = (\Sigma, L, L_0, L_F, \mathbb{X}, \mathbb{P}, E)$
Output: Z is the match set $\mathcal{M}(w, \mathcal{A})$

1 $i \leftarrow 1$ // i is the position in w of the beginning of the current matching trial
2 $N = \min\{|w| \mid w \in \bigcup_{v \in (\mathbb{Q}_+)^{\mathbb{P}}} \mathcal{L}_{-\$}(v(\mathcal{A}))\}$
3 **while** $i \leq |w| - N + 1$ **do**
4 | **while** $\forall v \in (\mathbb{Q}_+)^{\mathbb{P}}, (\overline{a'}, \overline{\tau'}) \in \mathcal{L}(v(\mathcal{A})).\, a_{i+N-1} \neq a'_N$ **do**
| | // Try matching the n-th action of $\mathcal{L}(v(\mathcal{A}))$
5 | | $i \leftarrow i + \Delta_{\mathrm{QS}}(a_{i+N})$ // Quick Search-style skipping
6 | | **if** $i > |w| - N + 1$ **then**
7 | | | **return**
8 | $Z \leftarrow Z \cup \{(t, t', v) \in [\tau_{i-1}, \tau_i) \times (\tau_{i-1}, \infty) \times (\mathbb{Q}_+)^{\mathbb{P}} \mid w|_{(t,t')} \in \mathcal{L}(v(\mathcal{A}))\}$
| // Try matching
9 | $j \leftarrow \max\{j \in \{i, i+1, \ldots, |w|\} \mid \exists \ell \in L, v \in (\mathbb{Q}_+)^{\mathbb{P}}, t \in \mathbb{R}_{\geq 0}.\, w(i,j) - t \in \mathcal{L}(v(\mathcal{A}_\ell))\}$
10 | $C \leftarrow \{(\ell, V) \in L \times \mathcal{P}((\mathbb{Q}_+)^{\mathbb{P}}) \mid \forall v \in V, \exists t \in \mathbb{R}_{\geq 0}.\, w(i,j) - t \in \mathcal{L}(v(\mathcal{A}_\ell))\}$
11 | $i \leftarrow i + \max_{(\ell,V) \in C} \Delta_{\mathrm{KMP}}(\ell, V)$ // Parametric KMP-style skipping
12 $Z \leftarrow Z \cup \{(t, t', v) \in [\tau_{|w|}, \infty) \times (\tau_{|w|}, \infty) \times (\mathbb{Q}_+)^{\mathbb{P}} \mid w|_{(t,t')} \in \mathcal{L}(v(\mathcal{A}))\}$

3.4.2 Experiments

We implemented our *dedicated* algorithms for parametric timed pattern matching in a tool ParamMONAA. We implemented the following three algorithms: the online algorithm without skipping (Algorithm 4, referred as "no skip"); the online algorithm with parametric skipping (Algorithm 5, referred as "parametric skip"); and the online algorithm with non-parametric skipping (Algorithm 5 where $\Delta_{\mathrm{KMP}}(\ell, V)$ at line 11 is replaced with $\Delta'_{\mathrm{KMP}}(\ell)$, referred as "non-parametric skip"). In the skip value computation, we use reachability synthesis for PTAs. Since reachability synthesis is intractable in general (the emptiness problem, i. e., the (non-)existence of a valuation reaching a given location, is undecidable [AHV93]), we use the following over-approximation: after investigating 100 configurations, we speculate that all the inconclusive parameter valuations are reachable parameter valuations. We remark that this overapproximation does not affect the correctness of parametric timed pattern matching, as it potentially *decreases* the skip value. We conducted experiments to answer the following research questions.

RQ1 Which is the fastest algorithm for parametric timed pattern matching?

RQ2 Why is parametric timed pattern matching slower than non-parametric timed pattern matching? Namely, is it purely because of the difficulty of the problem itself or is it mainly because of the general implementation and data structure required by the general problem setting?

RQ3 How large is the overhead of the skip value computation? Namely, is it small and acceptable?

We implemented ParamMONAA in C++ and we compiled them using GCC 7.3.0. We conducted the 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). Experiment data can be found on https://github.com/MasWag/monaa/blob/PTPM/doc/NFM2019.md.

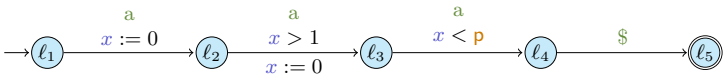


Figure 3.9: OnlyTiming: the parameter p is substituted to 1 in OnlyTiming-np.

Table 3.4: Execution time for Gear [s]

$\|w\|$	No Skip	Non-Param. Skip	Param. Skip	IMITATOR
1467	0.04	0.05	0.05	1.781
2837	0.0725	0.0805	0.09	3.319
4595	0.124	0.13	0.1405	5.512
5839	0.1585	0.156	0.17	7.132
7301	0.201	0.193	0.2115	8.909
8995	0.241	0.2315	0.2505	10.768
10315	0.2815	0.269	0.2875	12.778
11831	0.322	0.301	0.325	14.724
13183	0.3505	0.3245	0.353	16.453
14657	0.392	0.361	0.395	18.319

Table 3.5: Execution time for Accel [s]

$\|w\|$	No Skip	Non-Param. Skip	Param. Skip	IMITATOR
2559	0.03	0.0515	0.06	2.332
4894	0.0605	0.0605	0.0705	4.663
7799	0.1005	0.071	0.08	7.532
10045	0.13	0.08	0.09	9.731
12531	0.161	0.09	0.1	12.503
15375	0.1985	0.1005	0.113	15.583
17688	0.2265	0.1095	0.1215	17.754
20299	0.261	0.115	0.1325	21.040
22691	0.288	0.121	0.143	23.044
25137	0.3205	0.1315	0.159	25.815

Table 3.6: Execution time for Blowup [s]

$\|w\|$	No Skip	Non-Param. Skip	Param. Skip	IMITATOR
2000	66.75	68.0125	67.9735	OutOfMemory
4000	267.795	271.642	269.084	OutOfMemory
6000	601.335	611.782	607.58	OutOfMemory
8000	1081.42	1081.25	1079	OutOfMemory
10000	1678.15	1688.22	1694.53	OutOfMemory

Table 3.7: Execution time for OnlyTiming [s]

$\|w\|$	No Skip	Non-Param. Skip	Param. Skip	IMITATOR
1000	0.0995	0.1305	0.11	1.690
2000	0.191	0.23	0.191	3.518
3000	0.2905	0.3265	0.273	5.499
4000	0.3905	0.426	0.3525	7.396
5000	0.488	0.5225	0.4325	9.123
6000	0.588	0.6235	0.517	11.005

Figure 4.3 shows the pattern PTAs we used in the experiments. We reuse the benchmarks Gear, Accel, and Blowup from Section 3.3 as well as the new benchmark OnlyTiming. The timed words for Gear and Accel are generated by the automatic transmission system model in [HAF14]. Blowup and OnlyTiming are toy examples. Blowup shows the worst case situation for parametric timed pattern matching. In OnlyTiming, the parametric skip values are greater than the non-parametric skip values. In Sections 3.4.2.2 and 3.4.2.3, we also used the non-parametric variants Gear-np, Accel-np, Blowup-np, and OnlyTiming-np where the parameters are substituted to specific concrete values.

3.4.2.1 RQ1: Overall execution time

To answer RQ1, we compared the total execution time of ParamMONAA using Gear, Accel, Blowup, and OnlyTiming. As a baseline, we used our previous implementation of parametric timed pattern matching based on IMITATOR ("Butter Jellyfish", version 2.10.4). Tables 3.4 to 3.7 and Fig. 3.10 show the execution time of our online algorithms compared with the IMITATOR-based implementation.

In Tables 3.4 to 3.7, we observe that our algorithms are faster than the IMITATOR-based implementation by orders of magnitude. Moreover, for Blowup, the IMITATOR-based implementation aborted due to out of memory. This is mainly because ParamMONAA is specific to parametric timed pattern matching while IMITATOR is a general tool for parametric verification. This shows the much better efficiency of our new approach compared to Section 3.3.

In Fig. 3.10, we observe that the curve of "no skip" has the steepest slope and the curves of either "parametric skip" or "non-parametric skip" have the gentlest slope except for Blowup. Blowup is a benchmark designed on purpose to observe exponential blowup of the execution time, and it requires much time for all the implementations.

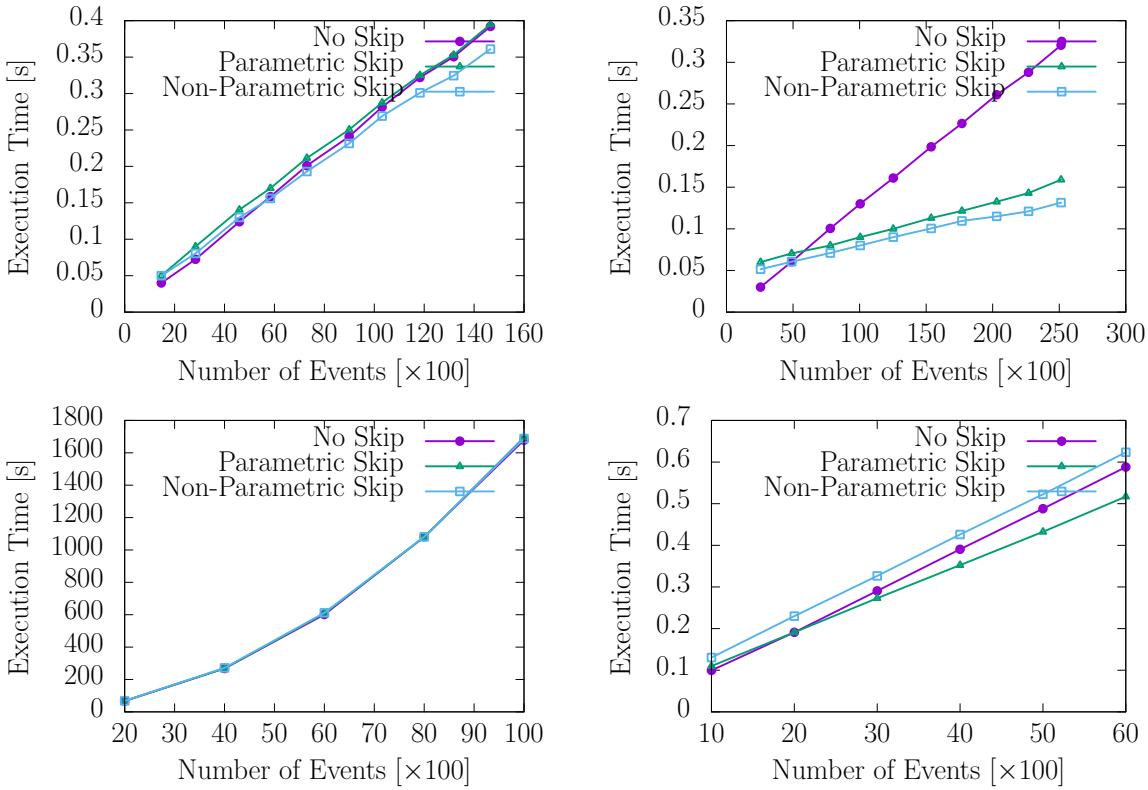


Figure 3.10: Execution time for the benchmarks with parameters which MONAA cannot handle: Gear (above left), Accel (above right), Blowup (below left), and OnlyTiming (below right)

For Gear and Accel, the execution time of "non-parametric skip" increases the most gently. This is because the parametric KMP-style skip value $\Delta_{\mathrm{KMP}}(\ell, V)$ and the non-parametric KMP-style skip value $\Delta'_{\mathrm{KMP}}(\ell)$ are equal for these benchmarks, and "parametric skip" is slower due to the inclusion checking $V \subseteq V_{\ell,n}$.

For OnlyTiming, we observe that the execution time of "parametric skip" increases the most gently because the parametric KMP-style skip value $\Delta_{\mathrm{KMP}}(\ell, V)$ is larger than the non-parametric KMP-style skip value $\Delta'_{\mathrm{KMP}}(\ell)$.

We conclude that skipping usually makes parametric timed pattern matching efficient. The preference between two skipping methods depends on the pattern PTA and it is a future work to investigate the tendency. Since the computation of the skip values does not take much time, the following work flow is reasonable: *i*) compute the skip values for both of them; and *ii*) use "parametric skip" only if its skip values are strictly larger than that of "non-parametric skip".

3.4.2.2 RQ2: Parametric vs. non-parametric timed pattern matching

To answer RQ2, we ran ParamMONAA using the non-parametric benchmarks (Accel-np, Gear-np, Blowup-np, and OnlyTiming-np) and compared the execution time with a tool MONAA [WHS18] for non-parametric timed pattern matching.

In Fig. 3.11, we observe that our algorithms are slower than MONAA by orders of magnitude even though we solve the same problem (non-parametric timed pattern matching). This is presumably because our implementations rely on Parma Polyhedra Library (PPL) [BHZ08] to compute symbolic states, while MONAA utilizes DBMs (difference bound matrices) [Dil89]. It

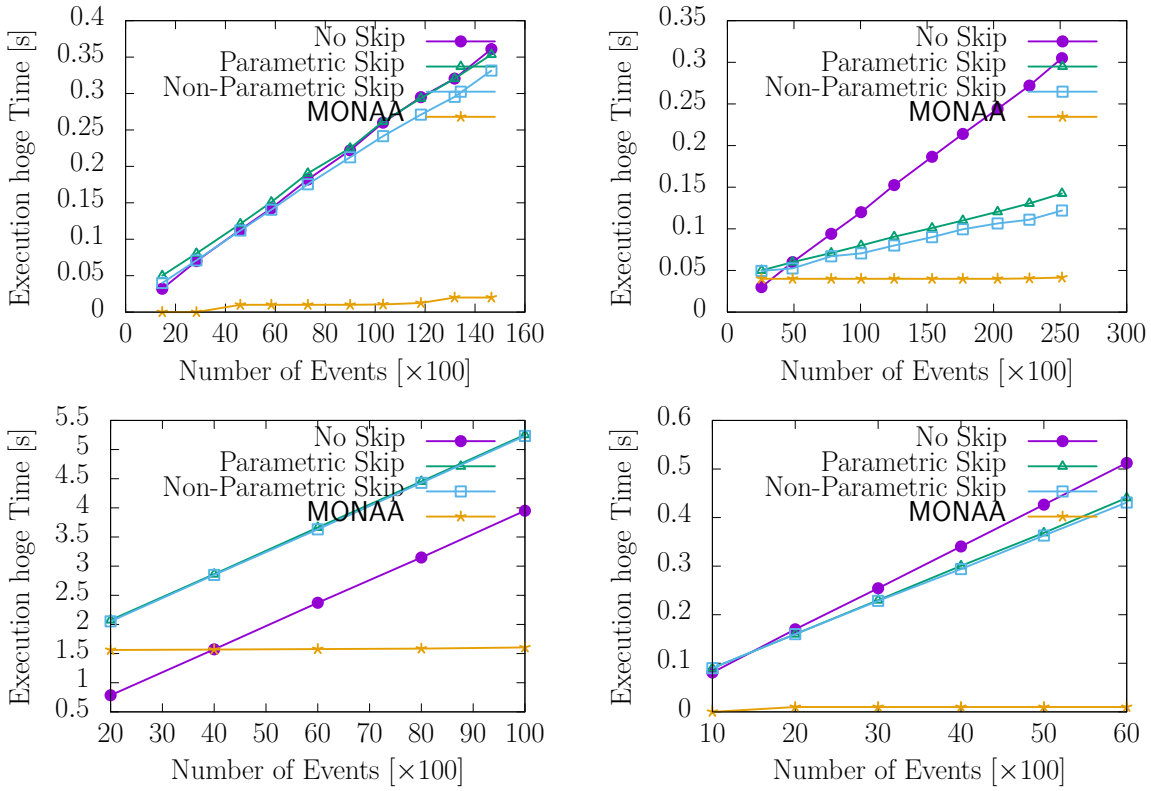


Figure 3.11: Execution time for the benchmarks without parameters: Gear-np (above left), Accel-np (above right), Blowup-np (below left), and OnlyTiming-np (below right)

was shown in [BFMU17] that polyhedra may be dozens of times slower than DBMs; however, for parametric analyses, DBMs are not suitable, and parameterized extensions (e. g., in [HRSV02]) still need polyhedra in their representation.

Moreover, in Figs. 3.10 and 3.11, we observe that the execution time of our algorithms are not much different between each parametric benchmark and its non-parametric variant except Blowup. This observation shows that at least one additional parameter does not make the problem too difficult.

Therefore, we conclude that the lower efficiency of parametric timed pattern matching is mainly because of its general data structure required by the general problem setting.

3.4.2.3 RQ3: Overhead of skip value computation

To answer RQ3, we compared the execution time of our algorithms for an empty timed word using all the benchmarks. As a baseline, we also measured the execution time of MONAA.

In Table 3.8, we observe that the execution time for the skip values is less than 0.05 second except for Blowup and Blowup-np. Even for the slowest pattern PTA Blowup-np, the execution time for the skip values is less than 1.5 second and it is faster than that of MONAA. We conclude that the overhead of the skip value computation is small and acceptable in many usage scenarios.

Table 3.8: Execution time [s] for the skip value computation

	Non-Parametric Skip	Parametric Skip	MONAA
Gear	0.0115	0.0175	n/a
Gear-np	0.01	0.01	< 0.01
Accel	0.042	0.0435	n/a
Accel-np	0.04	0.04	0.0305
OnlyTiming	0.03	0.03	n/a
OnlyTiming-np	0.02	0.02	< 0.01
Blowup	0.3665	0.381	n/a
Blowup-np	1.268	1.2905	1.5455

Figure 3.12: A log of entrance and leaving from a building. Timestamps are omitted for simplicity. We usually know who entered or left the building (e. g., BobEnter) but we sometimes do not know who (e. g., XEnter).

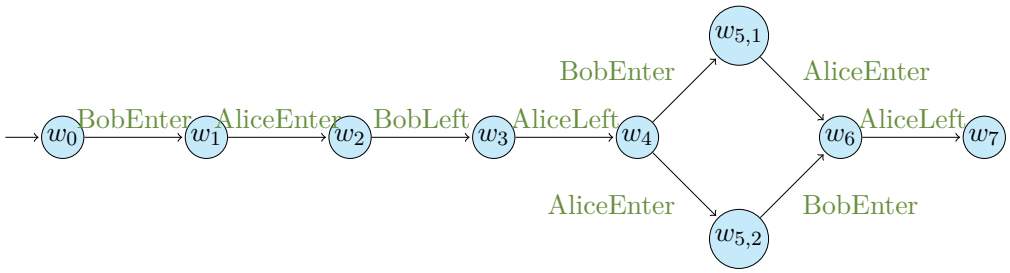


Figure 3.13: The TA constructed from the log in Fig. 3.12 using TW2PTA in Section 3.3.2.2. We assume $\mathsf{X}, \mathsf{Y} \in \{\text{Alice}, \text{Bob}\}$ and $\mathsf{X} \neq \mathsf{Y}$. The timing constraints are omitted for simplicity.

3.5 Comparison between the two approaches

In Sections 3.3 and 3.4, we presented two approaches for parametric timed pattern matching. In Section 3.3, we reduced parametric timed pattern matching to parametric timed model checking of PTAs (Algorithm 3). Parametric timed model checking relies on the reachability analysis with symbolic abstraction by convex polyhedra. In Section 3.4, we solved parametric timed pattern matching by following the transition of the PTA (Algorithm 4). Moreover, we optimized Algorithm 4 using *skipping* (Algorithm 5), which is a technique originally from string matching. Although the relationship between parametric timed pattern matching and parametric timed model checking is implicit in Section 3.4, Algorithms 4 and 5 utilize convex polyhedra to compute the *reachable* concrete states. Moreover, both IMITATOR and Param-MONAA rely on Parma Polyhedra Library (PPL) [BHZ08] for convex polyhedra operation. In this sense, the underlying technique utilized in Section 3.3 is also used in Section 3.4.

For the performance, Section 3.4.2 shows that Algorithms 4 and 5 are much more efficient than Algorithm 3. This efficiency is thanks to the skipping and its direct implementation rather than an indirect approach via parametric timed model checking.

Therefore, someone might consider that the approach in Section 3.4 is strictly superior to the one in Section 3.3. However, we note that the model checking-based framework in

Section 3.3 is generic and it is robust to the modification of the problem setting. For example, consider the ill-shaped log shown in Fig. 3.12: we take a log of the entrance and leavening from a building with the name and the timestamp, but we sometimes fail to identify who passed the gate e. g., due to a sensor error. Since a timed word w is a sequence of pairs (a_i, τ_i) of an event a_i and a timestamp τ_i, it cannot directly represent the log in Fig. 3.12 and therefore, it is not straightforward to handle such a log using Algorithms 4 and 5. However, it is easy to generalize the conversion TW2PTA from a timed word to a (P)TA in Section 3.3.2.2 to a log with branching. An example of the result is shown in Fig. 3.13. Thus, it is rather straightforward to monitor such an ill-shaped log using Algorithm 3. Moreover, the framework in Section 3.3 has a potential to be used for a different monitoring e. g., monitoring of a probabilistic behavior using probabilistic model checking.

In summary, Algorithms 4 and 5 are better at solving the parametric timed pattern matching problem, but the construction used in Algorithm 3 is general and it is potentially applicable to other similar problems.

3.6 Related work

Several algorithms have been proposed for online monitoring of real-time temporal logic specifications. Online monitoring consists in monitoring on-the-fly at runtime, while offline monitoring is performed after the execution is completed, with less hard constraints on the monitoring algorithm performance. An online monitoring algorithm for ptMTL (a past time fragment of MTL [Koy90]) was proposed in [RFB14] and an algorithm for MTL[U,S] (a variant of MTL with both forward and backward temporal modalities) was proposed in [HOW14]. In addition, a case study on an autonomous research vehicle monitoring [KCDK15] shows such procedures can be performed in an actual vehicle.

The approaches most related to ours are [UFAM14, UFAM16, Ulu17]. In that series of works, logs are encoded by *signals*, i. e., values that vary over time. This can be seen as a *state-based* view, while our timed words are *event-based*. The formalism used for specification in [UFAM14, UFAM16] is timed regular expressions (TREs). An offline monitoring algorithm is presented in [UFAM14] and an online one is presented in [UFAM16]. These algorithms are implemented in the tool *Montre* [Ulu17]. In [BFN+18], the setting is signals matched against a temporal pattern; the construction is automata-based as in [WAH16, WHS17].

Some algorithms have also been proposed for parameter identification of a temporal logic specification with uncertainty over a log. In the discrete time setting, an algorithm for an extension of LTL is proposed in [FR08]; and in the real-time setting, algorithms for parametric signal temporal logic (PSTL) are proposed in [ADMN11, JTS+17, BFM18]. Although these works are related to our approach, previous approaches do not focus on segments of a log but on a whole log. In contrast, we exhibit intervals together with their associated parameter valuations, in a fully symbolic fashion. We believe our matching-based setting is advantageous in many usage scenarios e. g., from hours of a log of a car, extracting timing constraints of a certain actions to cause slipping. Also, our setting allows the patterns with complex timing constraints (see the pattern in Fig. 3.6c for example).

Further works attempted to quantify the distance between a specification and a signal temporal logic (STL) specification (e. g., [DFM13, DMP17, JBG+18b]). The main difference with our work is that these works compute a distance w.r.t. to a whole log, while we aim at exhibiting where in the log is the property satisfied; our notion of parameters can also be seen

as a relative time distance. However, our work is closer to the robust satisfaction of guards rather than signal values; in that sense, our contribution is more related to the time robustness in [DM10] or the distance in [ABD18].

Finally, while our work is related to parameter synthesis, in the sense that we identify parameter valuations in the property such that it holds (or not), the term "parameter synthesis" is also used in monitoring with a slightly different meaning: given a *model* with parameters, the goal is to find parameters that maximize the robustness of the specification, i. e., satisfying behaviors for a range of parameters for which the model robustly satisfies the property. A notable tool achieving this is Breach [Don10].

3.7 Conclusion and perspectives

3.7.1 Conclusion

We proposed two approaches to perform timed pattern matching in the presence of an uncertain specification. This allows us to synthesize parameter valuations and intervals for which the specification holds on an input timed word. Our implementation using IMITATOR may not completely allow for *online* timed pattern matching yet, but already gives an interesting feedback in terms of parametric monitoring. Our second algorithm aiming at finding minimal or maximal parameter valuations is less sensitive to state space explosion. While our algorithms should be further optimized, we believe they pave the way for a more precise monitoring of real-time systems with an output richer than just timed intervals.

In a second part, our dedicated method dramatically outperforms the previous approach using parametric timed model checking. In addition, we discussed an optimization using skipping, that brings an interesting speedup.

3.7.2 Perspectives

Exploiting the polarity of parameters, as done in [ADMN11] or in lower-bound/upper-bound parametric timed automata [HRSV02, AL17], may help to improve the efficiency of $\mathsf{PTPM}_{\mathsf{opt}}$.

In addition, natural future works include more expressive specifications than (parametric) timed automata-based specifications, e. g., using more expressive logics such as [BKMZ15b].

Another challenge is the interpretation (and the visualization) of the results of parametric timed pattern matching. While the result of $\mathsf{PTPM}_{\mathsf{opt}}$ is natural, the fully symbolic result of PTPM remains a challenge to be interpreted; for example, the 125,250 matches for Blowup means the union of 125,250 polyhedra in 5 dimensions. We give a possible way to visualize such results in Fig. 3.8 for Gear ($|w| = 1,467$): in particular, observe in Fig. 3.8a that a single point exceeds 3, only a few exceed 2, while the wide majority remain in $[0, 1]$. This helps to visualize how fast the gear is changed from 1 to 2, and at what timestamps.

CHAPTER 4

Symbolic Monitoring against Specifications Parametric in Time and Data

In this chapter, we introduce a new formalism *parametric timed data automata (PTDAs)* and give a *symbolic monitoring* algorithm against a specification in a PTDA. PTDAs extend parametric timed automata with infinite domain data. This chapter is based on joint work [WAH19] with Étienne André and Ichiro Hasuo. Useful comments from the anonymous referees of the paper are gratefully acknowledged.

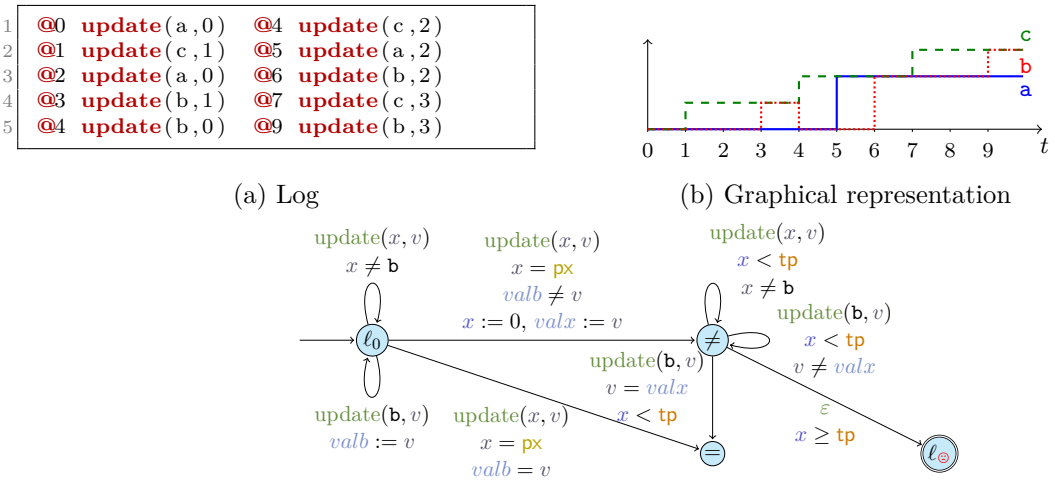
Organization of the chapter Section 4.1 summarizes the technical contribution in this chapter. After reviewing the necessary preliminaries in Section 4.2, We introduce parametric timed data automata in Section 4.3. We present our symbolic monitoring approach in Section 4.4 and show the experimental evaluation in Section 4.5. We review related work in Section 4.6. we conclude in Section 4.7.

4.1 Summary

Here, we summarize our contribution from the technical viewpoint. See Section 1.6.2 for a summary from more application viewpoint.

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 [BKMZ15b]), 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



(a) Log

(b) Graphical representation

(c) Monitoring PTDA

Figure 4.1: Monitoring copy to b within tp time units

of the signal temporal logic (STL) [BDSV14], 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 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 4.1. Consider a system updating three variables a, b and c (i. e., strings) to values (rationals). An example of log is given in Fig. 4.1a. Although our work is event-based, we can give a graphical representation similar to that of signals in Fig. 4.1b. 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". In our formalism, a simple automaton made of 4 locations (given in Fig. 4.1c) can monitor this property. 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 the variable parameter px and the timing parameter 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.

Table 4.1: Comparison of monitoring expressiveness

Work	[ADMN11]	[BDSV14]	[BKMZ15b]	[BKMZ15a]
Timing parameters	√	×	**?**	**?**
Data	√	√	√	√
Parametric data	√	×	√	√
Memory	×	√	√	√
Aggregation	×	×	×	√
Complete parameter identification	√	N/A	√/×	√/×

Work	[RCR15]	[HPU17]	[AHW18]	[BFM18]	This work
Timing parameters	**?**	×	√	×	√
Data	√	√	×	√	√
Parametric data	√	√	×	√	√
Memory	√	√	×	×	√
Aggregation	√	×	×	×	√
Complete parameter identification	N/A	N/A	√	√	√

Our algorithm outputs such violation by a constraint e. g., $\mathsf{px} = c \wedge \mathsf{tp} \leq 2$. 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 $\mathsf{tp} > 2$. ◇

We propose an automata-based approach to perform monitoring parametric in both time and data. We use an extension of both timed automata extended with data, and of parametric timed automata, both extended with parametric data over infinite domains. We implement our work in a dedicated prototype SyMon (relying on polyhedra to encode symbolic parameter relations) 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 4.1 (see Section 4.6 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 reasonable monitoring speed, especially given the degree of parametrization in our formalism. Note that it is not easy to formally claim superiority in expressiveness: proofs would require arguments such as the pumping lemma; and such formal comparison does not seem to be a concern of the existing work. Moreover, such formal comparison bears little importance for industrial practitioners: expressivity via an elaborate encoding is hardly of practical use. We also note that, in the existing work, we often observe gaps between the formalism in a theory and the formalism that the resulting tool actually accepts. This is not the case with the current framework.

4.2 Preliminaries: Clocks, timing parameters and timed guards

Although most of the following concepts are already introduced in the previous chapters, we show them to clarify the notation in this section.

We assume a set $\mathbb{X} = \{x_1, \ldots, x_H\}$ of *clocks*, i. e., real-valued variables that evolve at the same rate. A *clock valuation* is a function $\nu : \mathbb{X} \to \mathbb{R}_{\geq 0}$. We write $\vec{0}$ for the clock valuation assigning 0 to all clocks. Given $d \in \mathbb{R}_{\geq 0}$, $\nu + d$ denotes the valuation s.t. $(\nu + d)(x) = \nu(x) + d$,

for all $x \in \mathbb{X}$. Given $R \subseteq \mathbb{X}$, we define the *reset* of a valuation ν, denoted by $[\nu]_R$, as follows: $[\nu]_R(x) = 0$ if $x \in R$, and $[\nu]_R(x) = \nu(x)$ otherwise.

We assume a set $\mathbb{TP} = \{\mathsf{tp}_1, \ldots, \mathsf{tp}_J\}$ of *timing parameters*, i. e., unknown timing constants. A *timing parameter valuation* γ is a function $\gamma : \mathbb{TP} \to \mathbb{Q}_+$.[1] We assume $\bowtie \in \{<, \leq, =, \geq, >\}$. A *timed guard* tg is a constraint over $\mathbb{X} \cup \mathbb{TP}$ defined by a conjunction of inequalities of the form $x \bowtie d$, or $x \bowtie \mathsf{tp}$ with $d \in \mathbb{N}$ and $\mathsf{tp} \in \mathbb{TP}$. Given tg, we write $\nu \models \gamma(tg)$ if the expression obtained by replacing each x with $\nu(x)$ and each tp with $\gamma(\mathsf{tp})$ in tg evaluates to true.

4.3 Parametric timed data automata

4.3.1 Variables, data parameters and data guards

Here we show our formulation of the data. For sake of simplicity, we assume a *single* infinite domain $\mathbb{D}$ for data. The formalism defined in Section 4.3.2 can be straightforwardly extended to different domains for different variables (and our implementation SyMon does allow for different types). The case of *finite* data domain is immediate too. We however define this formalism in an *abstract* manner, so as to allow a sort of parameterized domain.

We assume a set $\mathbb{V} = \{v_1, \ldots, v_M\}$ of *variables* valued over $\mathbb{D}$. These variables are internal variables, that allow a high expressive power in our framework, as they can be compared or updated to other variables or parameters. We also assume a set $\mathbb{LV} = \{lv_1, \ldots, lv_O\}$ 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 $\mathrm{upate}(x, v)$), and in the associated guard and (right-hand part of) updates. We assume a set $\mathbb{VP} = \{\mathsf{vp}_1, \ldots, \mathsf{vp}_N\}$ 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 4.2. 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 $v \bowtie \mathsf{vp}$ with $v, v' \in \mathbb{V} \cup \mathbb{LV}$, $d \in \mathbb{Q}$ and $\mathsf{vp} \in \mathbb{VP}$. $\mathcal{DE}$ 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 $v \approx \mathsf{vp}$ with $v, v' \in \mathbb{V} \cup \mathbb{LV}$, $s \in \mathbb{S}$, $\mathsf{vp} \in \mathbb{VP}$ 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 a function $\mu : \mathbb{V} \to \mathbb{D}$. A *local variable valuation* is a partial function $\eta : \mathbb{LV} \nrightarrow \mathbb{D}$. A *data parameter valuation* ζ is a function $\zeta : \mathbb{VP} \to \mathbb{D}$. Given a data guard

[1] We choose $\mathbb{Q}_+$ by consistency with most of the PTA literature, but also because, for classical PTAs, choosing $\mathbb{R}_{\geq 0}$ leads to undecidability [Mil00].

Table 4.2: Variables, parameters and valuations used in guards

	timed guards		data guards		
	clock	timing parameter	(data) variable	local variable	data parameter
Variable	x	tp	v	lv	vp
Valuation	ν	γ	μ	η	ζ

$u \in \mathcal{DE}$, a variable valuation μ, a local variable valuation η defined for the local variables in u, and a data parameter valuation ζ, we write $(\mu, \eta) \models \zeta(u)$ if the expression obtained by replacing within u all occurrences of each data parameter vp_i by $\zeta(\mathsf{vp}_i)$ 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 $\mathsf{PDU} : \mathbb{V} \nrightarrow \mathcal{DU}$. 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 PDU, a variable valuation μ, a local variable valuation η (defined for all local variables appearing in PDU), and a data parameter valuation ζ, we define $[\mu]_{\eta(\zeta(\mathsf{PDU}))} : \mathbb{V} \to \mathbb{D}$ as follows:

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

where $\eta(\mu(\zeta(\mathsf{PDU}(v))))$ denotes the replacement within the update expression $\mathsf{PDU}(v)$ of all occurrences of each data parameter vp_i by $\zeta(\mathsf{vp}_i)$, and all occurrences of each variable v_j (resp. local variable lv_k) with its concrete valuation $\mu(v_j)$ (resp. $\eta(lv_k)$). Observe that this replacement gives a value in $\mathbb{D}$, therefore the result of $[\mu]_{\eta(\zeta(\mathsf{PDU}))}$ is indeed a data parameter valuation $\mathbb{V} \to \mathbb{D}$. That is, $[\mu]_{\eta(\zeta(\mathsf{PDU}))}$ computes the new (non-parametric) variable valuation obtained after applying to μ the partial function PDU valuated with ζ.

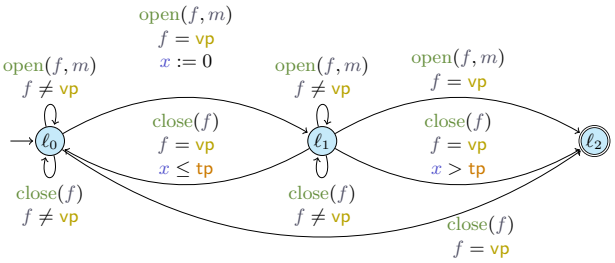
Example 4.3. Consider the data type for rationals, the variables set $\{v_1, v_2\}$, the local variables set $\{lv_1, lv_2\}$ and the parameters set $\{\mathsf{vp}_1\}$. Let μ be the variable valuation such that $\mu(v_1) = 1$ and $\mu(v_2) = 2$, and η be the local variable valuation such that $\eta(lv_1) = 2$ and $\eta(lv_2)$ is not defined. Let ζ be the data parameter valuation such that $\zeta(\mathsf{vp}_1) = 1$. Consider the parametric data update function PDU such that $\mathsf{PDU}(v_1) = 2 \times v_1 + v_2 - lv_1 + \mathsf{vp}_1$, and $\mathsf{PDU}(v_2)$ is undefined. Then the result of $[\mu]_{\eta(\zeta(\mathsf{PDU}))}$ is μ' such that $\mu'(v_1) = 2 \times \mu(v_1) + \mu(v_2) - \eta(lv_1) + \zeta(\mathsf{vp}_1) = 3$ and $\mu'(v_2) = 2$. ◇

4.3.2 Parametric timed data automata

We introduce here parametric timed data automata (PTDAs). They can be seen as an extension of parametric timed automata [AHV93] (that extend timed automata [AD94] 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., [BER94, Dan03, Qua15]), that we again extend with both data parameters and timing parameters. Or as an extension of quantified event automata [BFH+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 4.2.

```
1 @2046 open(Hakuchi.txt,rw)
2 @2136 open(Unagi.mp4,rw)
3 @2166 close(Hakuchi.txt)
```

(a) Example of log



(b) PTDA monitor

Figure 4.2: Monitoring proper file opening and closing

4.3.2.1 Syntax of PTDAs

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

Definition 4.4 (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 $\mathcal{A} = (\Sigma, L, \ell_0, L_F, \mathbb{X}, \mathbb{TP}, \mathbb{V}, \mathbb{LV}, \mu_0, \mathbb{VP}, E)$, where:

1. Σ is a finite set of actions,
2. L is a finite set of locations,
3. $\ell_0 \in L$ is the initial location,
4. $L_F \subseteq L$ is the set of accepting locations,
5. $\mathbb{X}$ is a finite set of clocks,
6. $\mathbb{TP}$ is a finite set of timing parameters,
7. $\mathbb{V}$ (resp. $\mathbb{LV}$) is a finite set of variables (resp. local variables) over $\mathbb{D}$,
8. μ_0 is the initial variable valuation,
9. $\mathbb{VP}$ is a finite set of data parameters,
10. E is a finite set of edges $e = (\ell, tg, u, a, R, \mathsf{PDU}, \ell')$ where *i*) $\ell, \ell' \in L$ are the source and target locations, *ii*) tg is a timed guard, *iii*) $u \in \mathcal{DE}$ is a data guard such as $Var(u) \cap \mathbb{LV} \subseteq Dom(a)$, *iv*) $a \in \Sigma$, *v*) $R \subseteq \mathbb{X}$ is a set of clocks to be reset, and *vi*) $\mathsf{PDU} : \mathbb{V} \nrightarrow \mathcal{DU}$ is the parametric data update function such that $Var(\mathsf{PDU}) \cap \mathbb{LV} \subseteq Dom(a)$.

◇

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

Example 4.5. Consider the PTDA in Fig. 4.2b over the data type for strings. We have $\mathbb{X} = \{x\}$, $\mathbb{TP} = \{\mathsf{tp}\}$, $\mathbb{V} = \emptyset$ and $\mathbb{LV} = \{f, m\}$. $Dom(\text{open}) = \{f, m\}$ while $Dom(\text{close}) = \{f\}$. ℓ_2 is the only accepting location, modeling the violation of the specification.

This PTDA (freely inspired by a formula from [HPU17] 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 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 ℓ_0 and ℓ_1 with data guard $f \neq \mathsf{vp}$). Whenever vp is opened (transition from ℓ_0 to ℓ_1), a clock x is reset. Then, in ℓ_1, if f is closed within tp time units (timed guard "$x \leq \mathsf{tp}$"), then the system goes back to ℓ_0. However, if instead f is opened again, this is an incorrect behavior and the system enters ℓ_2 via the upper transition. The same occurs if f is closed more than tp time units after opening. ◇

Given a data parameter valuation ζ and a timing parameter valuation γ, 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 vp_i (resp. tp_j) have been replaced by $\zeta(\mathsf{vp}_i)$ (resp. $\gamma(\mathsf{tp}_j)$).

Note that, if $\mathbb{V} = \mathbb{LV} = \emptyset$, then $\mathcal{A}$ is a *parametric timed automaton* [AHV93] and $\gamma|\zeta(\mathcal{A})$ is a *timed automaton* [AD94].[2]

4.3.2.2 Semantics of PTDAs

We now equip our TDAs with a concrete semantics.

Definition 4.6 (Semantics of a TDA)**.** Given a PTDA $\mathcal{A} = (\Sigma, L, \ell_0, L_F, \mathbb{X}, \mathbb{TP}, \mathbb{V}, \mathbb{LV}, \mu_0, \mathbb{VP}, E)$ over a data type $(\mathbb{D}, \mathcal{DE}, \mathcal{DU})$, a data parameter valuation ζ and a timing parameter valuation γ, the semantics of $\gamma|\zeta(\mathcal{A})$ is given by the timed transition system (TTS) $(S, s_0, \rightarrow)$, with

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

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 $(\ell_0, \mu_0, \nu_0), (e_0, \eta, d_0), (\ell_1, \mu_1, \nu_1), \cdots$ with $i = 0, 1, \ldots$, $e_i \in E$, $d_i \in \mathbb{R}_{\geq 0}$ 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 $(a_1, \tau_1, \eta_1), (a_2, \tau_2, \eta_2), \cdots$, where a_i is the action of edge e_{i-1}, η_i is the local variable valuation

[2] We may need to multiply all timing constants in $\gamma|\zeta(\mathcal{A})$ by the least common multiple of their denominators, so as to obtain an equivalent (integer-valued) TA, as defined in [AD94].

associated with that transition, and $\tau_i = \sum_{0 \leq j \leq i-1} d_j$, for $i = 1, 2 \cdots$.[3] For a timed data word w and a concrete state (ℓ, μ, ν) of a TDA $\gamma|\zeta(\mathcal{A})$, we write $(\ell_0, \mu_0, \vec{0}) \xrightarrow{w} (\ell, \mu, \nu)$ in $\gamma|\zeta(\mathcal{A})$ if w is associated with a run of $\gamma|\zeta(\mathcal{A})$ of the form $(\ell_0, \mu_0, \vec{0}), \ldots, (\ell_n, \mu_n, \nu_n)$ with $(\ell_n, \mu_n, \nu_n) = (\ell, \mu, \nu)$. For a timed data word $w = (a_1, \tau_1, \eta_1), (a_2, \tau_2, \eta_2), \ldots, (a_n, \tau_n, \eta_n)$, we denote $|w| = n$ and for any $i \in \{1, 2, \ldots, n\}$, we denote $w(1, i) = (a_1, \tau_1, \eta_1), (a_2, \tau_2, \eta_2), \ldots, (a_i, \tau_i, \eta_i)$.

A finite run is *accepting* if its last state (ℓ, μ, ν) is such that $\ell \in L_F$. 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 4.7. Consider again the PTDA in Fig. 4.2b over the data type for strings. Let $\gamma(\texttt{tp}) = 100$ and $\zeta(\texttt{vp}) = \texttt{Hakuchi.txt}$. 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. 4.2b), $\nu_0(x) = 0$, $\nu_1(x) = 0$, $\nu_2(x) = 90$, $\nu_3(x) = 120$, e_0 is the upper edge from ℓ_0 to ℓ_1, e_1 is the self-loop above ℓ_1, e_2 is the lower edge from ℓ_1 to ℓ_2, $\eta_0(f) = \eta_2(f) = \texttt{Hakuchi.txt}$, $\eta_1(f) = \texttt{Unagi.mp4}$, $\eta_0(m) = \eta_1(m) = \texttt{rw}$, and $\eta_2(m)$ is undefined (because $Dom(\text{close}) = \{f\}$).

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 η_0 can be represented as open(Hakuchi.txt, rw). Using this convention, the log in Fig. 4.2a corresponds exactly to this timed data word. ◇

4.4 Symbolic monitoring against PTDA specifications

In symbolic monitoring, in addition to the (observable) actions in Σ, we employ the *unobservable* action denoted by ε with $Dom(\varepsilon) = \emptyset$. We write Σ_ε for $\Sigma \amalg \{\varepsilon\}$. We let η_ε be the local variable valuation such that $\eta_\varepsilon(lv)$ is undefined for any $lv \in \mathbb{LV}$. For a timed data word $w = (a_1, \tau_1, \eta_1), (a_2, \tau_2, \eta_2), \ldots, (a_n, \tau_n, \eta_n)$ over Σ_ε, the projection $w{\downarrow_\Sigma}$ is the timed data word over Σ obtained from w by removing any triple (a_i, τ_i, η_i) where $a_i = \varepsilon$. An edge $e = (\ell, tg, u, a, R, \mathsf{PDU}, \ell') \in E$ is *unobservable* if $a = \varepsilon$, and *observable* otherwise. The use of unobservable actions makes symbolic monitoring more general, and allows us in particular to encode parametric timed pattern matching (see Section 4.4.3).

Example 4.8. Let $\Sigma = \{\text{wd}\}$, $\mathbb{LV} = \{a\}$, and $Dom(\text{wd}) = \{a\}$. For a timed data word $w = (\text{wd}, 0.1, \eta_1), (\text{wd}, 0.3, \eta_2), (\varepsilon, 0.7, \eta_3), (\text{wd}, 0.9, \eta_4)$ over Σ_ε, $w{\downarrow_\Sigma}$ is the timed data word $w{\downarrow_\Sigma} = (\text{wd}, 0.1, \eta_1), (\text{wd}, 0.3, \eta_2), (\text{wd}, 0.9, \eta_4)$ over Σ. ◇

We make the following assumption on the PTDAs in symbolic monitoring, which is necessary for the termination of our algorithm.

Assumption 4.9. *The PTDA $\mathcal{A}$ does not contain any loop of unobservable edges.* ◇

4.4.1 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 $(\gamma, \zeta) \in (\mathbb{Q}_+)^{\mathbb{P}} \times \mathbb{D}^{\mathbb{VP}}$ satisfying $w(1, i) \in \gamma|\zeta(\mathcal{A})$, where $w(1, i)$ is a

[3] The "−1" in indices comes from the fact that, following usual conventions in the literature, states are numbered starting from 0 while words are numbered from 1.

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

Example 4.10. Consider the PTDA $\mathcal{A}$ and the timed data word w shown in Fig. 4.1. The validity domain $D(w, \mathcal{A})$ is $D(w, \mathcal{A}) = D_1 \cup D_2$, where

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

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 γ and ζ are such that $\gamma(\mathsf{tp}) = 1.8$ and $\zeta(\mathsf{xp}) = \mathrm{c}$, and $w(1, 3) \cdot (\varepsilon, \eta_\varepsilon, 2.9)$ denotes the juxtaposition. ◇

For the data types in Example 4.2, 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.

4.4.2 Summary of our 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.

Outline Let $w = (a_1, \tau_1, \eta_1), (a_2, \tau_2, \eta_2), \ldots (a_n, \tau_n, \eta_n)$ and $\mathcal{A}$ be the timed data word and PTDA given in symbolic monitoring, respectively. Intuitively, after reading (a_i, τ_i, η_i), our algorithm symbolically computes for all parameter valuations $(\gamma, \zeta) \in (\mathbb{Q}_+)^{\mathbb{P}} \times \mathbb{D}^{\mathbb{VP}}$ the concrete states (ℓ, ν, μ) satisfying $(\ell_0, \mu_0, \vec{0}) \xrightarrow{w(1,i)} (\ell, \mu, \nu)$ in $\gamma|\zeta(\mathcal{A})$. Since $\mathcal{A}$ has unobservable edges as well as observable edges, we insert unobservable actions between observable actions in w. By $Conf_i^o$, we denote the configurations after reading (a_i, τ_i, η_i) and no unobservable actions are appended after (a_i, τ_i, η_i). By $Conf_i^u$, we denote the configurations after reading (a_i, τ_i, η_i) and at least one unobservable action is appended after (a_i, τ_i, η_i).

Definition 4.11 ($Conf_i^o$, $Conf_i^u$)**.** For a PTDA $\mathcal{A}$ over actions Σ_ε, a timed data word w over Σ, and $i \in \{0, 1, \ldots, |w|\}$ (resp. $i \in \{-1, 0, \ldots, |w|\}$), $Conf_i^o$ (resp. $Conf_i^u$) is the set of 5-tuples $(\ell, \nu, \gamma, \mu, \zeta)$ such that there is a timed data word w' over Σ_ε satisfying the following: *i*) $(\ell_0, \mu_0, \vec{0}) \xrightarrow{w'} (\ell, \mu, \nu)$ 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 τ_{i+1}). ◇

Algorithm 6 shows an outline of our algorithm for symbolic monitoring (see Section 4.4.4 for the full version). Our algorithm incrementally computes $Conf_{i-1}^u$ and $Conf_i^o$ (line 3). After reading (a_i, τ_i, η_i), our algorithm stores the partial results $(\gamma, \zeta) \in D(w, \mathcal{A})$ witnessed from the accepting configurations in $Conf_{i-1}^u$ and $Conf_i^o$ (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).)

Algorithm 6: Outline of our algorithm for symbolic monitoring

Input: A PTDA $\mathcal{A} = (\Sigma_\varepsilon, L, \ell_0, L_F, \mathbb{X}, \mathbb{TP}, \mathbb{V}, \mathbb{LV}, \mu_0, \mathbb{VP}, E)$ over a data type $(\mathbb{D}, \mathcal{DE}, \mathcal{DU})$ and actions Σ_ε, and a timed data word $w = (a_1, \tau_1, \eta_1), (a_2, \tau_2, \eta_2), \ldots, (a_n, \tau_n, \eta_n)$ over Σ

Output: $\bigcup_{i \in \{1,2,\ldots,n+1\}} Result_i$ is the validity domain $D(w, \mathcal{A})$

$Conf^u_{-1} \leftarrow \emptyset$; $Conf^o_0 \leftarrow \{(\ell_0, \vec{0}, \gamma, \mu_0, \zeta) \mid \gamma \in (\mathbb{Q}_+)^{\mathbb{P}}, \zeta \in \mathbb{D}^{\mathbb{VP}}\}$

for $i \leftarrow 1$ **to** n **do**

  compute $(Conf^u_{i-1}, Conf^o_i)$ **from** $(Conf^u_{i-2}, Conf^o_{i-1})$

  $Result_i \leftarrow \{(\gamma, \zeta) \mid \exists (\ell, \nu, \gamma, \mu, \zeta) \in Conf^u_{i-1} \cup Conf^o_i . \ell \in L_F\}$

compute $Conf^u_n$ **from** $(Conf^u_{n-1}, Conf^o_n)$

$Result_{n+1} \leftarrow \{(\gamma, \zeta) \mid \exists (\ell, \nu, \gamma, \mu, \zeta) \in Conf^u_n . \ell \in L_F\}$

Since $(\mathbb{Q}_+)^{\mathbb{P}} \times \mathbb{D}^{\mathbb{VP}}$ is an infinite set, we cannot try each $(\gamma, \zeta) \in (\mathbb{Q}_+)^{\mathbb{P}} \times \mathbb{D}^{\mathbb{VP}}$ and we use a symbolic representation for parameter valuations. Similarly to the reachability synthesis of parametric timed automata [JLR15], 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 6, some operations on the symbolic representation are required.

Theorem 4.12 (termination)**.** *For any PTDA $\mathcal{A}$ over a data type $(\mathbb{D}, \mathcal{DE}, \mathcal{DU})$ and actions Σ_ε, and for any timed data word w over Σ, Algorithm 6 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 $\{([\mu]_{\eta(\zeta(\mathsf{PDU}))}, \zeta) \mid \exists (\mu, \zeta) \in V_d . (\mu, \eta) \models \zeta(u)\}$, where η is a local variable valuation, PDU is a parametric data update function, and u 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 4.13. For the data type for rationals in Example 4.2, 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 4.2, 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 $\mathbb{S}$. ◇

4.4.3 Encoding parametric timed pattern matching

The symbolic monitoring problem is a generalization of the parametric timed pattern matching problem shown in Chapter 3. 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. We note that even with Assumption 4.9, symbolic monitoring is still a generalization of parametric timed pattern matching.

Consider the DOMINANT PTDA (left of Fig. 4.3). It is inspired by a monitoring of withdrawals from bank accounts of various users [BKZ17]. 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

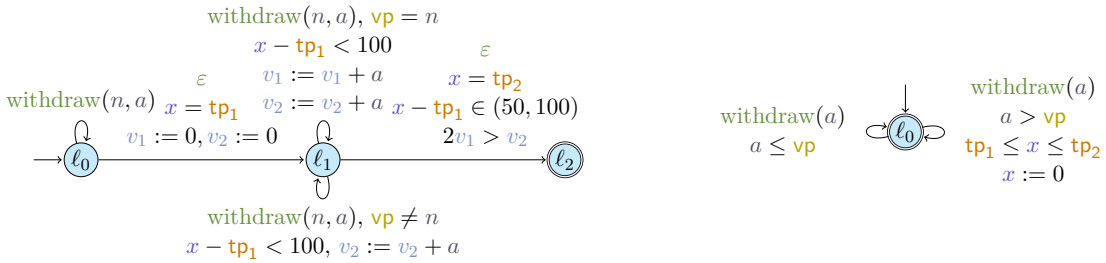


Figure 4.3: PTDAs in DOMINANT (left) and PERIODIC (right)

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 x is never reset, and therefore measures absolute time. The automaton can non-deterministically remain in ℓ_0, or start to measure a log by taking the ε-transition to ℓ_1 checking $x = \mathsf{tp}_1$, and therefore "remembering" the start time using timing parameter tp_1. Then, whenever a user vp has withdrawn more than half of the accumulated withdrawals (data guard $2v_1 > v_2$) in a $(50, 100)$ time window (timed guard $x - \mathsf{tp}_1 \in (50, 100)$), the automaton takes a ε-transition to the accepting location, checking $x = \mathsf{tp}_2$, and therefore remembering the end time using timing parameter tp_2.

4.4.4 Details on our algorithm for symbolic monitoring

Notations In the pseudocode, we use V_t, V_{t+}, and V_d for symbolic representation of valuations: V_t is a set of pairs $(\nu, \gamma) \in (\mathbb{R}_{\geq 0})^{\mathbb{X}} \times (\mathbb{Q}_+)^{\mathbb{P}}$ of a clock valuation and a time parameter valuation; V_{t+} is a set of triples $(\nu, \gamma, t) \in (\mathbb{R}_{\geq 0})^{\mathbb{X}} \times (\mathbb{Q}_+)^{\mathbb{P}} \times \mathbb{R}_{\geq 0}$ of a clock valuation, a time parameter valuation, and an elapsed time; and V_d is a set of pairs $(\mu, \zeta) \in \mathbb{D}^{\mathbb{V}} \times \mathbb{D}^{\mathbb{VP}}$ of a variable valuation and a data parameter valuation. We also use *CurrConf*, *NextConf*, and *CurrUConf*: *CurrConf* and *NextConf* are finite sets of triples (ℓ, V_t, V_d) and *CurrUConf* is a finite set of triples (ℓ, V_{t+}, V_d), where $\ell \in L$ is a location and V_t, V_d, and V_{t+} are as shown in the above. For $V_t \subseteq (\mathbb{R}_{\geq 0})^{\mathbb{X}} \times (\mathbb{Q}_+)^{\mathbb{P}}$ and $t \in \mathbb{R}_{\geq 0}$, we let $V_t + t = \{(\nu + t, \gamma) \mid (\nu, \gamma) \in V_t\}$. For $V_t \subseteq (\mathbb{R}_{\geq 0})^{\mathbb{X}} \times (\mathbb{Q}_+)^{\mathbb{P}}$, $V_{t+} \subseteq (\mathbb{R}_{\geq 0})^{\mathbb{X}} \times (\mathbb{Q}_+)^{\mathbb{P}} \times \mathbb{R}_{\geq 0}$, and $V_d \subseteq \mathbb{D}^{\mathbb{V}} \times \mathbb{D}^{\mathbb{VP}}$, we denote $V_t{\downarrow_{\mathbb{TP}}} = \{\gamma \mid \exists (\nu, \gamma) \in V_t\}$, $V_{t+}{\downarrow_{\mathbb{TP}}} = \{\gamma \mid \exists (\nu, \gamma, t) \in V_{t+}\}$, and $V_d{\downarrow_{\mathbb{VP}}} = \{\zeta \mid \exists (\mu, \zeta) \in V_d\}$. We let $\tau_0 = 0$.

Detail of the algorithm Algorithm 7 is a pseudocode of our algorithm for symbolic monitoring. In line 1 of Algorithm 7, we set the current configurations *CurrConf* to be the triple $(\ell_0, \{\vec{0}\} \times (\mathbb{Q}_+)^{\mathbb{P}}, \{\mu_0\} \times \mathbb{D}^{\mathbb{VP}})$, which means we are at the initial location ℓ_0, the clock (resp. variable) valuation is the initial valuation $\vec{0}$ (resp. μ_0), and the timing (resp. data) parameter valuations can be any valuations $(\mathbb{Q}_+)^{\mathbb{P}}$ (resp. $\mathbb{D}^{\mathbb{VP}}$). In lines 3 to 15, we try unobservable transitions. In line 3, we set the current configurations *CurrUConf* for the unobservable transitions, which is essentially the same as *CurrConf*, but each V_t is equipped with the time elapse after the latest observable transition. The elapsed time t is used *i*) to restrict the unobservable transitions between the last observable action a_{i-1} and the next observable action a_i (line 7) and *ii*) to make the time elapse to τ_i (line 13). For $(\ell, V_{t+}, V_d) \in$ *CurrUConf*, after time elapse in line 7, we try unobservable edges from ℓ (lines 8 to 15). We constrain the valuations (V_{t+}, V_d) by the guards (tg and u) and conduct the reset and update in lines 9 and 10. If (V_{t+}, V_d) satisfies the guards, we add the valuations (V'_{t+}, V'_d) and the valuations after time elapse to *CurrUConf* and *NextConf*, respectively. Moreover, if $\ell' \in L_F$, we add the parameter valuations $(V'_{t+}{\downarrow_{\mathbb{TP}}}, V'_d{\downarrow_{\mathbb{VP}}})$ to *Result*. After trying the unobservable edges, in lines 16 to 25,

Algorithm 7: Algorithm for symbolic monitoring

Input: A PTDA $\mathcal{A} = (\Sigma_\varepsilon, L, \ell_0, L_F, \mathbb{X}, \mathbb{TP}, \mathbb{V}, \mathbb{LV}, \mu_0, \mathbb{VP}, E)$ over a data type $(\mathbb{D}, \mathcal{DE}, \mathcal{DU})$ and actions Σ_ε, and a timed data word $w = (a_1, \tau_1, \eta_1), (a_2, \tau_2, \eta_2), \ldots, (a_n, \tau_n, \eta_n)$ over Σ

Output: *Result* is the validity domain $D(w, \mathcal{A})$

$\mathit{CurrConf} \leftarrow \{(\ell_0, \{\vec{0}\} \times (\mathbb{Q}_+)^{\mathbb{P}}, \{\mu_0\} \times \mathbb{D}^{\mathbb{VP}})\}$; $\mathit{Result} \leftarrow \emptyset$
for $i \leftarrow 1$ **to** n **do**
  $\mathit{CurrUConf} \leftarrow \{(\ell, V_t \times \{0\}, V_d) \mid (\ell, V_t, V_d) \in \mathit{CurrConf}\}$
    // append the elapsed time from τ_{i-1}
  $\mathit{NextConf} \leftarrow \emptyset$
  while $\mathit{CurrUConf} \neq \emptyset$ **do** // insert ε before (a_i, τ_i, η_i)
    pop (ℓ, V_{t+}, V_d) **from** $\mathit{CurrUConf}$
    $V_{t+} \leftarrow \{(\nu + d, \gamma, t + d) \mid (\nu, \gamma, t) \in V_{t+}, d \in \mathbb{R}_{>0}. t + d < \tau_i - \tau_{i-1}\}$ // time elapse
    for $e = (\ell, tg, u, \varepsilon, R, \mathsf{PDU}, \ell') \in E$ **do** // try unobservable edges
      $V'_{t+} \leftarrow \{([\nu]_R, \gamma, t) \mid \exists (\nu, \gamma, t) \in V_{t+}. \nu \models \gamma(tg)\}$
        // constrain and reset the clock
      $V'_d \leftarrow \{([\mu]_{\eta_\varepsilon(\zeta(\mathsf{PDU}))}, \zeta) \mid \exists (\mu, \zeta) \in V_d. (\mu, \eta_\varepsilon) \models \zeta(u)\}$
        // constrain and update the data variables
      if $V'_{t+} \neq \emptyset$ & $V'_d \neq \emptyset$ **then**
        push (ℓ', V'_{t+}, V'_d) **to** $\mathit{CurrUConf}$
        push $(\ell', \{(\nu + \tau_i - \tau_{i-1} - t, \gamma) \mid \exists (\nu, \gamma, t) \in V'_{t+}\}, V'_d)$ **to** $\mathit{NextConf}$
        if $\ell' \in L_F$ **then** // found an accepting run
          $\mathit{Result} \leftarrow \mathit{Result} \cup (V'_{t+}{\downarrow_{\mathbb{TP}}} \times V'_d{\downarrow_{\mathbb{VP}}})$
  $\mathit{NextConf} \leftarrow \mathit{NextConf} \cup \{(\ell, V_t + (\tau_i - \tau_{i-1}), V_d) \mid (\ell, V_t, V_d) \in \mathit{CurrConf}\}$ // time elapse
  $(\mathit{CurrConf}, \mathit{NextConf}) \leftarrow (\mathit{NextConf}, \emptyset)$
  for $(\ell, V_t, V_d) \in \mathit{CurrConf}$ **do** // use (a_i, τ_i, η_i) for transition
    for $e = (\ell, tg, u, a_i, R, \mathsf{PDU}, \ell') \in E$ **do** // try observable edges
      $V'_t \leftarrow \{([\nu]_R, \gamma) \mid \exists (\nu, \gamma) \in V_t. \nu \models \gamma(tg)\}$ // constrain and reset the clock
      $V'_d \leftarrow \{([\mu]_{\eta_i(\zeta(\mathsf{PDU}))}, \zeta) \mid \exists (\mu, \zeta) \in V_d. (\mu, \eta_i) \models \zeta(u)\}$
        // constrain and update the data variables
      if $V'_t \neq \emptyset$ & $V'_d \neq \emptyset$ **then**
        push (ℓ', V'_t, V'_d) **to** $\mathit{NextConf}$
        if $\ell' \in L_F$ **then** // found an accepting run
          $\mathit{Result} \leftarrow \mathit{Result} \cup (V'_t{\downarrow_{\mathbb{TP}}} \times V'_d{\downarrow_{\mathbb{VP}}})$
  $(\mathit{CurrConf}, \mathit{NextConf}) \leftarrow (\mathit{NextConf}, \emptyset)$
while $\mathit{CurrConf} \neq \emptyset$ **do** // append ε after (a_n, τ_n, η_n)
  pop (ℓ, V_t, V_d) **from** $\mathit{CurrConf}$
  $V_t \leftarrow \{(\nu + d, \gamma) \mid (\nu, \gamma) \in V_t, d \in \mathbb{R}_{>0}\}$ // time elapse
  for $e = (\ell, tg, u, \varepsilon, R, \mathsf{PDU}, \ell') \in E$ **do**
    $V'_t \leftarrow \{([\nu]_R, \gamma) \mid \exists (\nu, \gamma) \in V_t. \nu \models \gamma(tg)\}$ // constrain and reset the clock
    $V'_d \leftarrow \{([\mu]_{\eta_\varepsilon(\zeta(\mathsf{PDU}))}, \zeta) \mid \exists (\mu, \zeta) \in V_d. (\mu, \eta_\varepsilon) \models \zeta(u)\}$
      // constrain and update the data variables
    if $V'_t \neq \emptyset$ & $V'_d \neq \emptyset$ **then**
      push (ℓ', V'_t, V'_d) **to** $\mathit{CurrConf}$
      if $\ell' \in L_F$ **then** // found an accepting run
        $\mathit{Result} \leftarrow \mathit{Result} \cup (V'_t{\downarrow_{\mathbb{TP}}} \times V'_d{\downarrow_{\mathbb{VP}}})$

we try observable edges. Finally, we try unobservable edges after the whole timed data word in lines 27 to 36. The explanation of lines 16 to 25 and lines 27 to 36 is essentially similar to that of lines 3 to 15.

Termination Since $\mathcal{A}$ does not have any loop of unobservable edges, *CurrConf* and *CurrUConf* are always finite sets. The valuations V_t, V'_t, V_{t+}, and V'_{t+} can be represented by convex polyhedra. The time elapse (e. g., in line 7), restriction and reset (e. g., in line 9), and projection (e. g., in line 15) are standard operations on convex polyhedra, and they terminate. Therefore, if the operations on variable and data parameter valuations V_d and V'_d terminate, Algorithm 7 terminates.

Algorithm 7 is correct because for each $w' \in \{w''(1,i) \mid w''{\downarrow_\Sigma} = w, i \in \{1,2,\dots,|w''|\}\}$, it adds $\{(\gamma,\zeta) \mid w' \in \mathcal{L}(\gamma|\zeta(\mathcal{A}))\}$ to *Result*.

Theorem 4.14 (correctness)**.** *For any PTDA $\mathcal{A}$ over a data type $(\mathbb{D}, \mathcal{DE}, \mathcal{DU})$ and actions Σ_ε, and for any timed data word w over Σ, if Algorithm 7 terminates, we have Result $= D(w,\mathcal{A})$ after the execution of Algorithm 7.* □

4.5 Experiments

We implemented our symbolic monitoring algorithm in a tool SyMon in C++ (compiled using GCC 7.3.0), where the domain for data is the strings and the integers.[4] For the strings, we used the data type in Example 4.2 and for integers, we used the data type for the rationals in Example 4.2, where any occurrences of $\mathbb{Q}$ are replaced by $\mathbb{Z}$. Our tool SyMon is distributed at `https://github.com/MasWag/symon`. We use the Parma Polyhedra Library (PPL) [BHZ08] for the symbolic representation of the valuations. We note that we employ an optimization to merge adjacent polyhedra in the configurations if possible.[5] See the following paragraph for the detail of the optimization. We evaluated our monitor algorithm against three original benchmarks: the PTDA in Copy is in Fig. 4.1c; and the PTDAs in Dominant and Periodic are shown in Fig. 4.3.

We conducted the 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).

Optimization In our implementation, we also employ an optimization to merge adjacent polyhedra in the configurations *NextConf* if possible. Precisely, we merge (ℓ, V_t, V_d) and (ℓ', V'_t, V'_d) in *NextConf* whenever we have the following:

- ℓ and ℓ' are the same.
- V_t and V'_t are the same.
- The projection of V_d and V'_d to the valuations on strings are the same.
- The projection of V_d and V'_d to the valuations on integers are adjacent convex polyhedra.

Such a merge is conducted after consuming each entry (a_i, τ_i, η_i) of the timed word w i. e., in line 26 of Algorithm 7.

[4] The use of integers is not an essential limitation. We may scale any rational number to an integer.

[5] After consuming each entry (a_i, τ_i, η_i) of the timed word w (i. e., in line 5 of Algorithm 6), we use PPL's `Pointset_Powerset::pairwise_reduce` function.

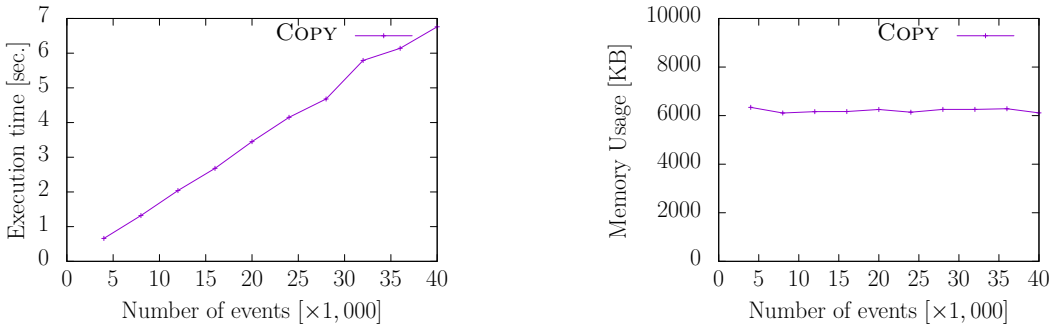


Figure 4.4: Execution time (left) and memory usage (right) of Copy

4.5.1 Benchmark 1: Copy

Table 4.3: Experiment results: each cell consists of a pair (T, M) of the execution time T [sec.] and the memory usage M [KiB] in the experiment setting.

(a) Results of Copy

$\|w\|$	**Copy**
4,000	(0.66,6340)
8,000	(1.32,6108)
12,000	(2.04,6164)
16,000	(2.68,6168)
20,000	(3.45,6252)
24,000	(4.15,6140)
28,000	(4.68,6256)
32,000	(5.79,6256)
36,000	(6.14,6284)
40,000	(6.76,6112)

(b) Results of Dominant and Periodic

$\|w\|$	**Dominant**	**Periodic**
2,000	(14.65,6928)	(6.66,6396)
4,000	(29.22,6964)	(14.91,6472)
6,000	(44.41,6964)	(16.82,6332)
8,000	(61.78,6956)	(27.85,6384)
10,000	(75.95,6936)	(36.64,6568)
12,000	(87.63,7032)	(37.59,6564)
14,000	(106.93,6984)	(55.93,6372)
16,000	(121.71,6948)	(57.09,6492)
18,000	(132.45,6952)	(61.53,6440)
20,000	(148.22,7236)	(69.59,6384)

Our first benchmark Copy is a monitoring of variable updates much like the scenario in [BDSV14]. 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. We generated random timed data words of various sizes. Our set W consists of 10 timed data words of length 4,000 to 40,000.

The PTDA in Copy is shown in Fig. 4.1c, where we give an additional constraint $3 < \mathtt{tp} < 10$ on $\mathtt{tp}$. The property encoded in Fig. 4.1c is "for any variable px, whenever an update of that variable occurs, then within $\mathtt{tp}$ time units, the value of $\mathtt{b}$ must be equal to that update".

The experiment result is in Fig. 4.4 and Table 4.3. 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.

4.5.2 Benchmark 2: Dominant

Our second benchmark is Dominant (Fig. 4.3 left). We generated random timed data words of various sizes, where the number of users is 3 and the duration between each withdrawal follows the uniform distribution on $\{1, 2, \ldots, 10\}$. Our set W consists of 10 timed data words of length 2,000 to 20,000. Recall that this PTDA matches a situation when the amount of the withdrawal by the user $\mathtt{vp}$ in a certain time window is more than the half of the withdrawals by all the users in the same time window. The time window must be between 50 and 100. The parameters $\mathtt{tp}_1$ and $\mathtt{tp}_2$ show the beginning and the end of the time window respectively.

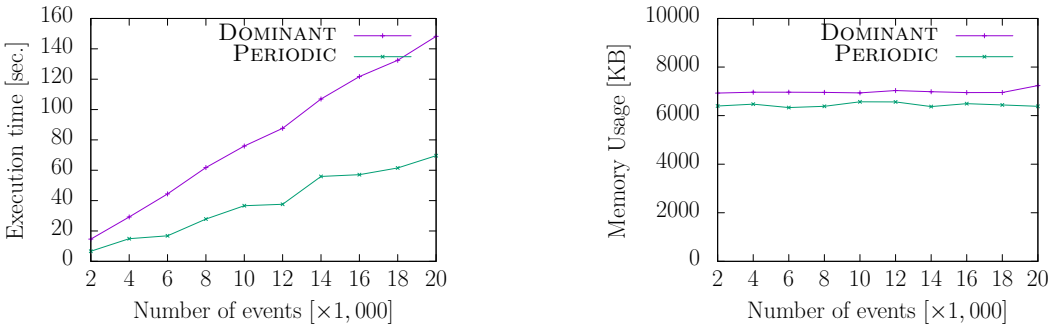


Figure 4.5: Execution time (left) and memory usage (right) of Dominant and Periodic

The experiment result is in Fig. 4.5 and Table 4.3. 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.

4.5.3 Benchmark 3: Periodic

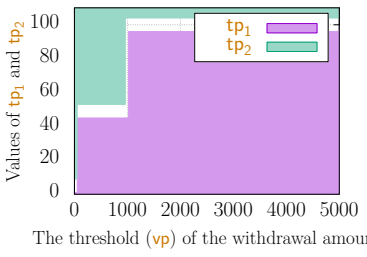
Our third benchmark Periodic is inspired by a parameter identification of periodic withdrawals from one bank account. The actions are $\Sigma = \{\text{withdraw}\}$ and $Dom(\text{withdraw}) = \{a\}$, where a has an integer value representing the amount of the withdrawal. We randomly generated a set W 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:

short period One withdrawal occurs every 5 ± 1 time units. The amount of the withdrawal is 50 ± 3.

middle period One withdrawal occurs every 50 ± 3 time units. The amount of the withdrawal is 1000 ± 40.

long period One withdrawal occurs every 100 ± 5 time units. The amount of the withdrawal is 5000 ± 20.

The PTDA in Periodic is shown in the right of Fig. 4.3. The PTDA matches situations where, for any two successive withdrawals of amount more than vp, the duration between them is within $[\mathsf{tp}_1, \mathsf{tp}_2]$. By the symbolic monitoring, one can identify the period of the periodic withdrawals of amount greater than vp is in $[\mathsf{tp}_1, \mathsf{tp}_2]$. An example of the validity domain is shown in the right figure.



The experiment result is in Fig. 4.5 and Table 4.3. 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.

4.5.4 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 $3 < \mathsf{tp} < 10$ in Copy is strict and the size of the

configurations (i. e., $Conf_i^o$ and $Conf_i^u$ in Algorithm 6) is small. Another positive result is that in all the benchmarks (Copy, Dominant, and Periodic), 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., $Conf_i^o$ and $Conf_i^u$ in Algorithm 6) is bounded due to the following reason. In Dominant, the loop in ℓ_1 of the PTDA is deterministic, and because of the guard $x - \mathsf{tp}_1 \in (50, 100)$ in the edge from ℓ_1 to ℓ_2, the number of the loop edges at ℓ_1 in an accepting run is bounded (if the duration between two continuing actions are bounded as in the current setting). Therefore, $|Conf_i^o|$ and $|Conf_i^u|$ in Algorithm 6 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, $|Conf_i^o|$ and $|Conf_i^u|$ in Algorithm 6 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 4.1 in Section 4.6, no other monitoring tool can compute the valuations satisfying the specification. We however used the parametric timed model checker IMITATOR [AFKS12] 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 14 s with only 6.9 MiB for the same benchmark. Concerning Periodic, the only existing work that can possibly accommodate this specification is [ADMN11]. 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 [ADMN11], 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 [HPU17], it is reported that they can process a trace of length 1,100,004 in 30.3 seconds. 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.

4.6 Related work

4.6.1 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., [FP09, DM10, Don10, DFM13, DMP17, JBG+18b] (or in a slightly different setting in [ALFS11]). The distance can be understood in terms of space ("signals") or time. In [ABD18], the distance also copes for reordering of events. In [BFMU17], 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).

4.6.2 Monitoring with data

The tool MarQ [RCR15] performs monitoring using Quantified Event Automata (QEA) [BFH+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, [RCR15] does not seem to natively support specification parametric in time; in addition, [RCR15] 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 [BKMZ15b]. Aggregation operators are added in [BKMZ15a], allowing the computation of *sums* or *maximums* over data. A fragment of this logic is implemented in MonPoly [BKZ17]. 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 [HPU17], binary decision diagrams (BDDs) are used in order to symbolically represent the observed data in QTL. This can be seen as monitoring data against a parametric specification, with a symbolic internal encoding (the BDDs of [HPU17, HP18] work efficiently for comparing whether a variable is equal or not equal to another, but not for comparing whether a variable is smaller than another one—which suits strings better than rationals). 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 ($\neq$).

4.6.3 Freeze operator

In [BDSV14], the STL logic 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 10 seconds, x copies the values of y within a delay of 4 seconds": $\mathbf{G}_{[0,10]} * (\mathbf{G}_{[0,4]} y^* = x)$. 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 4.1. In contrast, it is not possible to determine using STL* which variables and which delays satisfy or violate the specification.

4.6.4 Monitoring with parameters

In [ADMN11], 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 [BFM18], the focus is only on signal parameters, with an improved efficiency by reusing techniques from the *robust* monitoring.

Whereas [ADMN11, BFM18] fit in the framework of signals and temporal logics while we fit in words and automata, our work shares similarities with [ADMN11, BFM18] in the sense that we can express data parameters; in addition, [BFM18] 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 [FR08], the problem of *inferring* temporal logic formulae with constraints that hold in a given numerical data time series is addressed. The method is applied to biological systems.

4.6.5 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 [UFAM14, UFAM16, Ulu17, BFN+18], or a temporal logic [UM18]. On the other hand, a line of works considers timed words, with specifications in the form of timed automata in [WAH16, WHS17] and also in Chapter 3 of this thesis. 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 (note that we also considered similar timing parameters in Chapter 3) and, second, we add data—themselves extended with parameters. That is, coming back to Example 4.1, [UFAM14, UFAM16, Ulu17, WHS17] 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 parametric timed pattern matching in Chapter 3 could infer both the segments of the log and the timing parameter valuations, but not which variable violates the specification.

4.6.6 Summary

We compare related works with our work in Table 4.1. "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. 4.1a 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. 4.1a, an example of property parametric in data is to synthesize the parameters for which the difference of values between two consecutive updates of variable px is always below pv, where px is a string parameter and 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 [RCR15, BKMZ15b, HPU17]). "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" [BKMZ15a]. This can be supported using dedicated aggregation operators [BKMZ15a] or using variables ([RCR15], 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 [BDSV14], 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 [HPU17]). In contrast, we return in a *symbolic* manner (as in [ADMN11, AHW18]) the exact set of (data and timing) parameters for which a property is satisfied. "$\surd$//$\times$" denotes "yes" in the theory paper, but not in the associated tool.

4.7 Conclusion and perspectives

4.7.1 Conclusion

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.

4.7.2 Perspectives

Combining the BDDs used in [HPU17] 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 [BL09]) could improve further the efficiency.

We only considered *infinite* domains, but the case of *finite* domains raises interesting questions concerning result representation: if the answer to a property on the log of Fig. 4.1a is "neither a nor b", knowing the domain is $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, then the answer should be c.

From a usability point of view, adding some syntactic improvements to the PTDAs will help further the ease of using by non-experts (for example allowing "update($\neg$b, _)" without guard instead of the self-loop over ℓ_0 in Fig. 4.1c).

Investigating further usage of the symbolic monitoring problem is also a future work. For example, it may be useful as a general query language for streams of data.

CHAPTER 5

Online Quantitative Timed Pattern Matching with Semiring-Valued Weighted Automata

In this chapter, we give an *online* algorithm for the quantitative timed pattern matching problem. This chapter is based on work [Wag19]. Useful comments from the anonymous referees are gratefully acknowledged.

Organization of the chapter Section 5.1 summarizes the technical contribution in this chapter. Section 5.2 introduces preliminaries on signals and semirings. Section 5.3 defines timed symbolic weighted automata (TSWAs), and our quantitative semantics of signals over a TSWA. Section 5.4 defines the quantitative timed pattern matching problem. Section 5.5 and Section 5.6 describe our algorithms for computing the quantitative semantics and the quantitative timed pattern matching problem, respectively. Section 5.7 presents our experimental results for the sup-inf and tropical semirings, which confirm the scalability of our algorithm under some reasonable assumptions. After reviewing the related work in Section 5.8, we present conclusions and some future perspectives in Section 5.9.

5.1 Summary

Here, we summarize our contribution from the technical viewpoint.

Quantitative timed pattern matching Among the various problem settings of monitoring, we focus on an *online* algorithm for *quantitative timed pattern matching* [BFMU17] in a *dense* time setting.

Figure 5.1: Piecewise-constant signal σ (left) and an illustration of the quantitative matching function $(\mathcal{M}(\sigma, \mathcal{W}))(t, t')$ for $[t, t') \subseteq [0, 30.5)$ (right). In the right figure, the score in the white areas is $-\infty$. The specification $\mathcal{W}$ is outlined in Example 5.1. In the right figure, the value at $(3, 15)$ is 5. It shows that the score $(\mathcal{M}(\sigma, \mathcal{W}))(3, 15)$, for the restriction $\sigma([3, 15))$ of σ to the interval $[3, 15)$, is 5.

Given a piecewise-constant signal σ and a specification $\mathcal{W}$ expressed by what we call a *timed symbolic weighted automaton*, our algorithm returns the *quantitative matching function* $\mathcal{M}(\sigma, \mathcal{W})$ that maps each interval $[t, t') \subseteq [0, |\sigma|)$ to the (quantitative) semantics $(\mathcal{M}(\sigma, \mathcal{W}))(t, t')$, with respect to $\mathcal{W}$, for the restriction $\sigma([t, t'))$ of σ to the interval $[t, t')$, where $|\sigma|$ is the duration of the signal. An illustration of $\mathcal{M}(\sigma, \mathcal{W})$ is in Fig. 5.1. In [BFMU17], quantitative timed pattern matching was solved by an offline algorithm using a syntax tree of *signal regular expressions*. In this chapter, we propose an *online* algorithm for quantitative timed pattern matching with automata. To the best of our knowledge, this is the first online algorithm for quantitative timed pattern matching. Moreover, our (quantitative) semantics is parameterized by a *semiring* and what we call a *cost function*. This algebraic formulation makes our problem setting general.

Example 5.1. Let σ be the piecewise-constant signal in the left of Fig. 5.1 and $\mathcal{W}$ be the specification meaning the following.

- At first, the value of x stays less than 15, and then the value of x becomes and remains greater than 5 within 5 s.
- We are only interested in the behavior within 10 s after the value of x becomes greater than 5.
- We want the score showing how robustly the above conditions are satisfied (or violated).

The right of Fig. 5.1 illustrates the result of quantitative timed pattern matching. Quantitative timed pattern matching computes the semantics $(\mathcal{M}(\sigma, \mathcal{W}))(t, t')$, with respect to $\mathcal{W}$, for each subsignal $\sigma([t, t'))$ of σ. The current semantics shows how robustly the conditions are satisfied. The semantics $(\mathcal{M}(\sigma, \mathcal{W}))(3, 15)$ for the subsignal $\sigma([3, 15))$ is 5, which is the value at $(3, 15)$ in the right of Fig. 5.1. This is because the distance between the first constraint $x < 15$ and the first valuation $x = 10$ of the subsignal $\sigma([3, 15))$ is 5, and the distance between the second constraint $x > 5$ and the valuations $x = 10$ and $x = 40$ of the subsignal $\sigma([3, 15))$ is not smaller than 5. The semantics $(\mathcal{M}(\sigma, \mathcal{W}))(10, 15)$ for the subsignal $\sigma([10, 15))$ is -25, which is the value at $(10, 15)$ in the right of Fig. 5.1. Thus, the subsignal $\sigma([3, 15))$ satisfies the condition specified in $\mathcal{W}$ more robustly than the subsignal $\sigma([10, 15))$.

Our algorithm is *online* and it starts returning the result before obtaining the entire signal σ. For example, after obtaining the subsignal $\sigma([0, 7.5))$ of the initial 7.5 s, our algorithm returns that for any $[t, t') \subseteq [0, 7.5)$, the score $(\mathcal{M}(\sigma, \mathcal{W}))(t, t')$ is 5. ◇

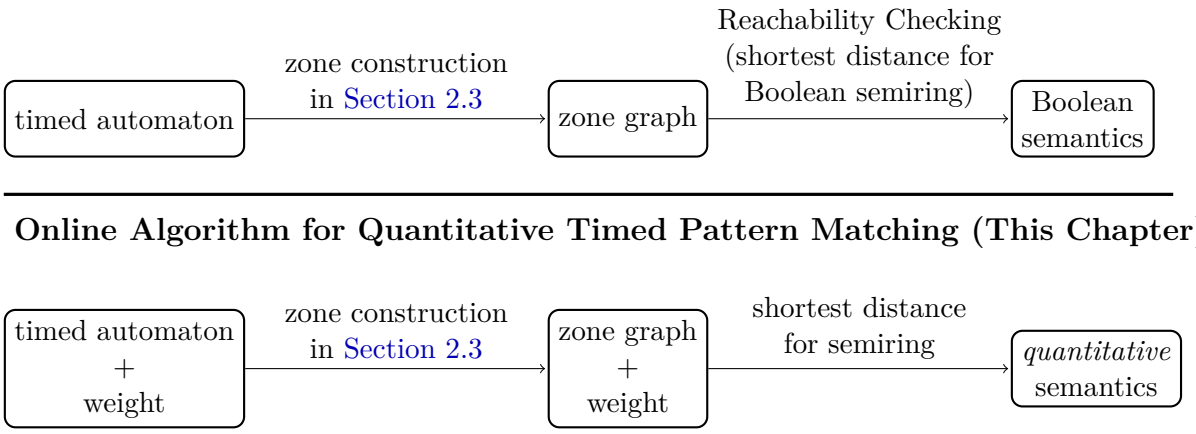
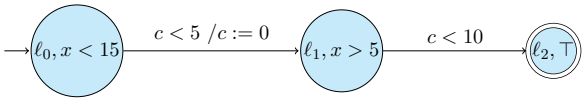


Figure 5.2: High level comparison between [BFN+18] and our contribution: our contribution is a generalization of [BFN+18] from Boolean semiring to semiring in general



$$\kappa_r\big(u, (a_1 a_2 \dots a_m)\big) = \inf_{i \in \{1,2,\dots,n\}} \kappa_r(u, (a_i))$$

$$\kappa_r\big(\bigwedge_{i=1}^{n} (v_i \bowtie_i d_i), (a)\big) = \inf_{i \in \{1,2,\dots,n\}} \kappa_r(v_i \bowtie_i d_i, (a)) \text{ where } \bowtie_i \in \{>, \geq, \leq, <\}$$

$$\kappa_r(v \succ d, (a)) = a(x) - d \quad \text{where } \succ \in \{\geq, >\}$$

$$\kappa_r(v \prec d, (a)) = d - a(x) \quad \text{where } \prec \in \{\leq, <\}$$

Figure 5.3: Example of a TSWA $\mathcal{W} = (\mathcal{A}, \kappa_r)$ which is the pair of the TSA $\mathcal{A}$ (upper) and the cost function κ_r (lower). See Definition 5.5 for the precise definition.

Our solution We formulate quantitative timed pattern matching using the *shortest distance* [Moh09] of semiring-valued (potentially *infinite*) weighted graphs. We reduce it to the shortest distance of *finite* weighed graphs. This is in contrast with the qualitative setting: the semantics is defined by the *reachability* in a (potentially *infinite*) graph and it is reduced to the reachability in a *finite* graph. The following is an overview.

Problem formulation We introduce *timed symbolic weighted automata (TSWAs)* and define the (quantitative) semantics $\alpha(\sigma, \mathcal{W})$ of a signal σ with respect to a TSWA $\mathcal{W}$. Moreover, we define *quantitative timed pattern matching* for a signal and a TSWA. A TSWA $\mathcal{W}$ is a pair $(\mathcal{A}, \kappa)$ of a *timed symbolic automaton (TSA)* $\mathcal{A}$ — that we also introduce in this chapter — and a cost function κ. The cost function κ returns a semiring value at each transition of $\mathcal{A}$, and the semiring operations specify how to accumulate such values over time. This algebraic definition makes our problem general. Fig. 5.3 shows an example of a TSWA.

Algorithm by zones We give an algorithm for computing our semantics $\alpha(\sigma, \mathcal{W})$ of a signal σ by the shortest distance of a *finite* weighted graph. The constructed weighted graph is

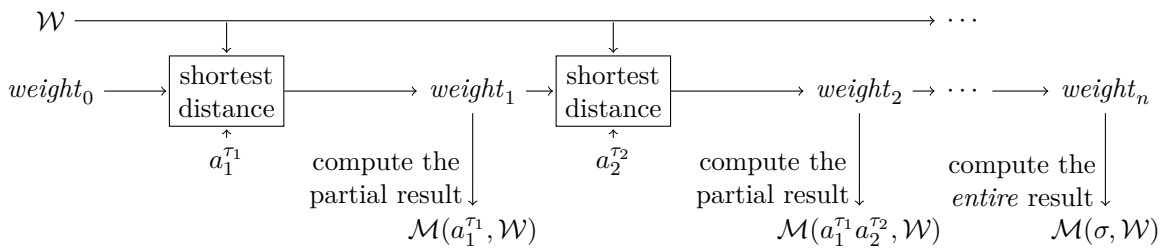


Figure 5.4: Illustration of our online algorithm for quantitative timed pattern matching of a signal $\sigma = a_1^{\tau_1} a_2^{\tau_2} \dots a_n^{\tau_n}$ meaning "the signal value is a_1 for τ_1, the signal value is a_2 for the next τ_2, ..." and a TSWA $\mathcal{W}$. The intermediate data $weight_i$ for the weight computation is represented by zones. The precise definition of the $weight_i$ is introduced later in Definition 5.14.

much like the *zone graph* [BY03b] for reachability analysis of timed automata. See also Section 2.3. Our algorithm is general and works for any semantics defined on an idempotent and complete semiring. (See Example 5.4 later for examples of such semirings.)

Incremental and online algorithms We present an incremental algorithm for computing the semantics $\alpha(\sigma, \mathcal{W})$ of a signal σ with respect to the TSWA $\mathcal{W}$. Based on this incremental algorithm for computing $\alpha(\sigma, \mathcal{W})$, we present an online algorithm for quantitative timed pattern matching. To the best of our knowledge, this is the first online algorithm for quantitative timed pattern matching. Our online algorithm for quantitative timed pattern matching works incrementally, much like in *dynamic programming*. Fig. 5.4 shows an illustration.

Contribution We summarize our contributions as follows.

- We formulate the semantics of a signal with respect to a TSWA by the shortest distance of a potentially *infinite* weighted graph.
- We reduce the above graph to a *finite* weighted graph.
- We give an online algorithm for quantitative timed pattern matching.

5.2 Preliminary

For a set X, its powerset is denoted by $\mathcal{P}(X)$. We use ε to represent the empty sequence. All the signals in this chapter are piecewise-constant, which is one of the most common interpolation methods of sampled signals.

Definition 5.2 (signal)**.** Let X be a finite set of variables defined over a data domain $\mathbb{D}$. A (piecewise-constant) *signal* σ is a sequence $\sigma = a_1^{\tau_1} a_2^{\tau_2} \dots a_n^{\tau_n}$, where for each $i \in \{1, 2, \dots, n\}$, $a_i \in \mathbb{D}^{\mathbb{V}}$ and $\tau_i \in \mathbb{R}_{>0}$. The set of signals over $\mathbb{D}^{\mathbb{V}}$ is denoted by $\mathcal{T}(\mathbb{D}^{\mathbb{V}})$. The *duration* $\sum_{i=1}^{n} \tau_i$ of a signal σ is denoted by $|\sigma|$. The sequence $a_1 \circ a_2 \circ \dots \circ a_n$ of the values of a signal σ is denoted by $\mathit{Values}(\sigma)$, where $a \circ a'$ is the absorbing concatenation

$$a \circ a' = \begin{cases} aa' & \text{if } a \neq a' \\ a & \text{if } a = a' \end{cases}.$$

We denote the set $\{a_1 \circ a_2 \circ \ldots \circ a_n \mid n \in \mathbb{Z}_{\geq 0}, a_1, a_2, \ldots, a_n \in \mathbb{D}^{\mathbb{V}}\}$ by $(\mathbb{D}^{\mathbb{V}})^{\circledast}$. For $t \in [0, |\sigma|)$, we define $\sigma(t) = a_k$, where k is such that $\sum_{i=1}^{k-1} \tau_i \leq t < \sum_{i=1}^{k} \tau_i$. For an interval $[t, t') \subseteq [0, |\sigma|)$, we define $\sigma([t, t')) = a_k^{\sum_{i=1}^{k} \tau_i - t} a_{k+1}^{\tau_{k+1}} \ldots a_{l-1}^{\tau_{l-1}} \ldots a_l^{t' - \sum_{i=1}^{l-1} \tau_i}$, where k and l are such that $\sum_{i=1}^{k-1} \tau_i \leq t < \sum_{i=1}^{k} \tau_i$ and $\sum_{i=1}^{l-1} \tau_i < t' \leq \sum_{i=1}^{l} \tau_i$.

◇

Definition 5.3 (semiring)**.** A system $\mathbb{S} = (S, \oplus, \otimes, e_\oplus, e_\otimes)$ is a *semiring* if we have the following.

- $(S, \oplus, e_\oplus)$ is a commutative monoid with identity element $e_\oplus$.
- $(S, \otimes, e_\otimes)$ is a monoid with identity element $e_\otimes$.
- For any $s, s', s'' \in S$, we have $(s \oplus s') \otimes s'' = (s \otimes s'') \oplus (s' \otimes s'')$ and $s \otimes (s' \oplus s'') = (s \otimes s') \oplus (s \otimes s'')$.
- For any $s \in S$, we have $e_\oplus \otimes s = s \otimes e_\oplus = e_\oplus$.

◇

A semiring $(S, \oplus, \otimes, e_\oplus, e_\otimes)$ is *complete* if for any $S' \subseteq S$, $\bigoplus_{s \in S'} s$ is an element of S satisfying the following.

$$\bigoplus_{s \in S'} s = e_\oplus \quad \text{if } S' = \emptyset \qquad \bigoplus_{s \in S'} s = s \quad \text{if } S' = \{s\}$$

$$\bigoplus_{s \in S'} s = \bigoplus_{i \in I} \Big(\bigoplus_{s \in S'_i} s \Big) \qquad \text{for any partition } S' = \coprod_{i \in I} S'_i$$

$$s \otimes \Big(\bigoplus_{s' \in S'} s' \Big) = \bigoplus_{s' \in S'} (s \otimes s') \quad \text{and} \quad \Big(\bigoplus_{s \in S'} s \Big) \otimes s' = \bigoplus_{s \in S'} (s \otimes s') \qquad \text{for any } s \in S$$

A semiring $\mathbb{S} = (S, \oplus, \otimes, e_\oplus, e_\otimes)$ is *idempotent* if for any $s \in S$, $s \oplus s = s$ holds. For a semiring $(S, \oplus, \otimes, e_\oplus, e_\otimes)$ and $s_1, s_2, \ldots, s_n \in S$, we denote $\bigoplus_{i=1}^{n} s_i = s_1 \oplus s_2 \oplus \cdots \oplus s_n$ and $\bigotimes_{i=1}^{n} s_i = s_1 \otimes s_2 \otimes \cdots \otimes s_n$.

Example 5.4. All the following four semirings are complete and idempotent.

***Boolean* semiring** $(\{\top, \bot\}, \vee, \wedge, \bot, \top)$

***Sup-inf* semiring** $(\mathbb{R} \amalg \{\pm\infty\}, \sup, \inf, -\infty, +\infty)$

***Tropical (inf-plus)* semiring** $(\mathbb{R} \amalg \{+\infty\}, \inf, +, +\infty, 0)$

***Sup-plus* semiring** $(\mathbb{R} \amalg \{-\infty\}, \sup, +, -\infty, 0)$

◇

Let $\mathbb{S} = (S, \oplus, \otimes, e_\oplus, e_\otimes)$ be a semiring and $G = (V, E, W)$ be a weighted graph over $\mathbb{S}$, i. e., V is the finite set of vertices, $E \subseteq V \times V$ is the finite set of edges, and $W \colon V \times V \to \mathbb{S}$ is the weight function. For $V_{\mathrm{from}}, V_{\mathrm{to}} \subseteq V$, the *shortest distance* from V_{from} to V_{to} is

$$\mathrm{Dist}(V_{\mathrm{from}}, V_{\mathrm{to}}, V, E, W) = \bigoplus_{v \in V_{\mathrm{from}}, v' \in V_{\mathrm{to}}} . \bigoplus_{v = v_1 v_2 \ldots v_n = v' \in Paths(G)} \bigotimes_{i=1}^{n-1} W(v_i, v_{i+1}) \,,$$

where $Paths(G)$ is the set of paths in the directed graph G, i. e.,

$$Paths(G) = \{v_1 v_2 \dots v_n \mid \forall i \in \{1, 2, \dots, n-1\}., (v_i, v_{i+1}) \in E\} \ .$$

For any complete semiring, the shortest distance problem can be solved by a generalization of the Floyd-Warshall algorithm [Moh09]. Under some conditions, the shortest distance problem can be solved more efficiently by a generalization of the Bellman-Ford algorithm [Moh09].

5.3 Timed symbolic weighted automata

We propose timed symbolic automata (TSAs), timed symbolic weighted automata (TSWAs), and the (quantitative) semantics of TSWAs. TSAs are an adaptation of timed automata [AD94] for handling signals over $\mathbb{D}$ rather than signals over a finite alphabet. In the remainder of this chapter, we assume that the data domain $\mathbb{D}$ is equipped with a partial order $\leq$. A typical example of the data domain $\mathbb{D}$ is the reals $\mathbb{R}$ with the usual order. We note that TSAs are much like the state-based variant of timed automata [ACM97, BFN+18] rather than the original, event-based definition [AD94] in Section 2.2.

For a finite set $\mathbb{V}$ of variables and a poset $(\mathbb{D}, \leq)$, we denote by $\Phi(\mathbb{V}, \mathbb{D})$ the set of constraints defined by a finite conjunction of inequalities $v \bowtie d$, where $v \in \mathbb{V}$, $d \in \mathbb{D}$, and $\bowtie \in \{>, \geq, <, \leq\}$. We denote $\bigwedge \emptyset \in \Phi(\mathbb{V}, \mathbb{D})$ by $\top$. For a finite set $\mathbb{X}$ of clock variables, a *clock valuation* is a function $\nu \in (\mathbb{R}_{\geq 0})^{\mathbb{X}}$. For a clock valuation $\nu \in (\mathbb{R}_{\geq 0})^{\mathbb{X}}$ over $\mathbb{X}$ and $\mathbb{X}' \subseteq \mathbb{X}$, we let $\nu{\downarrow_{\mathbb{X}'}} \in (\mathbb{R}_{\geq 0})^{\mathbb{X}'}$ be the clock valuation over $\mathbb{X}'$ satisfying $\nu{\downarrow_{\mathbb{X}'}}(x) = \nu(x)$ for any $x \in \mathbb{X}'$. For a finite set $\mathbb{X}$ of clock variables, let $\mathbf{0}_{\mathbb{X}}$ be the clock valuation $\mathbf{0}_{\mathbb{X}} \in (\mathbb{R}_{\geq 0})^{\mathbb{X}}$ satisfying $\mathbf{0}_{\mathbb{X}}(x) = 0$ for any $x \in \mathbb{X}$. For a clock valuation ν over $\mathbb{X}$ and $\tau \in \mathbb{R}_{\geq 0}$, we denote by $\nu + \tau$ the valuation satisfying $(\nu + \tau)(x) = \nu(x) + \tau$ for any $x \in \mathbb{X}$. For a clock valuation $\nu \in (\mathbb{R}_{\geq 0})^{\mathbb{X}}$ and $R \subseteq \mathbb{X}$, we denote by $[\nu]_R$ the valuation such that $([\nu]_R)(x) = 0$ for $x \in R$ and $([\nu]_R(x) = \nu(x)$ for $x \notin R$.

The definitions of TSAs and TSWAs are as follows. As shown in Fig. 5.3, TSAs are similar to the timed automata in [ACM97, BFN+18], but the locations are labeled with a constraint on the signal values $\mathbb{D}^{\mathbb{V}}$ instead of a character in a finite alphabet.

Definition 5.5 (timed symbolic, timed symbolic weighted automata)**.** For a poset $(\mathbb{D}, \leq)$, a *timed symbolic automaton* (*TSA*) over $\mathbb{D}$ is a 7-tuple $\mathcal{A} = (\mathbb{V}, L, L_0, L_F, \mathbb{X}, \Delta, \Lambda)$, where:

- $\mathbb{V}$ is a finite set of variables over $\mathbb{D}$;
- L is the finite set of locations;
- $L_0 \subseteq L$ is the set of initial locations;
- $L_F \subseteq L$ is the set of accepting locations;
- $\mathbb{X}$ is the finite set of clock variables;
- $\Delta \subseteq L \times \Phi(\mathbb{X}, \mathbb{Z}_{\geq 0}) \times \mathcal{P}(\mathbb{X}) \times L$ is the set of transitions; and
- Λ is the labeling function $\Lambda : L \to \Phi(\mathbb{V}, \mathbb{D})$.

For a poset $(\mathbb{D}, \leq)$ and a complete semiring $\mathbb{S} = (S, \oplus, \otimes, e_{\oplus}, e_{\otimes})$, a *timed symbolic weighted automaton* (*TSWA*) over $\mathbb{D}$ and $\mathbb{S}$ is a pair $\mathcal{W} = (\mathcal{A}, \kappa)$ of a TSA $\mathcal{A}$ over $\mathbb{D}$ and a *cost function* $\kappa \colon \Phi(\mathbb{V}, \mathbb{D}) \times (\mathbb{D}^{\mathbb{V}})^{\circledast} \to S$ over $\mathbb{S}$. $\diamond$

The semantics of a TSWA $\mathcal{W} = (\mathcal{A}, \kappa)$ on a signal σ is defined by the *trace value* $\alpha(\mathcal{S})$ of the *weighted timed transition systems* (*WTTS*) $\mathcal{S}$ of σ and $\mathcal{W}$. The trace value $\alpha(\mathcal{S})$ depends on the cost function κ and implicitly on its range semiring $\mathbb{S}$ as well as the signal σ and the TSA $\mathcal{A}$. As shown below, the state space of a WTTS $\mathcal{S}$ is $Q = L \times (\mathbb{R}_{\geq 0})^{\mathbb{X}} \times [0, |\sigma|] \times (\mathbb{D}^{\mathbb{V}})^{\circledast}$. Intuitively, a state $(\ell, \nu, t, \overline{a}) \in Q$ of $\mathcal{S}$ consists of: the current location ℓ; the current clock valuation ν; the current absolute time t; and the observed signal value $\overline{a}$ after the latest transition. The transition $\to$ of $\mathcal{S}$ is for a transition of $\mathcal{A}$ or time elapse.

Definition 5.6 (weighted timed transition systems)**.** For a signal $\sigma \in \mathcal{T}(\mathbb{D}^{\mathbb{V}})$ and a TSWA $\mathcal{W} = (\mathcal{A}, \kappa)$ over the data domain $\mathbb{D}$ and semiring $\mathbb{S}$, the *weighted timed transition system* (*WTTS*) $\mathcal{S} = (Q, Q_0, Q_F, \to, W)$ is as follows, where $\mathcal{A} = (\mathbb{V}, L, L_0, L_F, \mathbb{X}, \Delta, \Lambda)$ is a TSA over $\mathbb{D}$ and κ is a cost function over $\mathbb{S}$.

- $Q = L \times (\mathbb{R}_{\geq 0})^{\mathbb{X}} \times [0, |\sigma|] \times (\mathbb{D}^{\mathbb{V}})^{\circledast}$
- $Q_0 = \{(\ell_0, \mathbf{0}_C, 0, \varepsilon) \mid \ell_0 \in L_0\}$
- $Q_F = \{(\ell_F, \nu, |\sigma|, \varepsilon) \mid \ell_F \in L_F, \nu \in (\mathbb{R}_{\geq 0})^{\mathbb{X}}\}$
- $\to \subseteq Q \times Q$ is the relation such that $((\ell, \nu, t, \overline{a}), (\ell', \nu', t', \overline{a'})) \in \to$ if and only if either of the following holds.

  (transition of $\mathcal{A}$) $\exists (\ell, g, R, \ell') \in \Delta$ satisfying $\nu \models g$, $\nu' = [\nu]_R$, $t' = t$, $\overline{a'} = \varepsilon$, and $\overline{a} \neq \varepsilon$

  (time elapse) $\exists \tau \in \mathbb{R}_{>0}$ satisfying $\ell = \ell'$, $\nu' = \nu + \tau$, $t' = t + \tau$, and $\overline{a'} = \overline{a} \circ \mathit{Values}(\sigma([t, t + \tau)))$
- $W((\ell, \nu, t, \overline{a}), (\ell', \nu', t', \overline{a'}))$ is $\kappa(\Lambda(\ell), \overline{a})$ if $\overline{a'} = \varepsilon$; and $e_{\otimes}$ if $\overline{a'} \neq \varepsilon$

$\diamond$

Definition 5.7 (trace value)**.** For a WTTS $\mathcal{S} = (Q, Q_0, Q_F, \to, W)$, the *trace value* $\alpha(\mathcal{S})$ is the shortest distance $\mathrm{Dist}(Q_0, Q_F, Q, \to, W)$ from Q_0 to Q_F. $\diamond$

For a signal σ and a TSWA $\mathcal{W}$, by $\alpha(\sigma, \mathcal{W})$, we denote the trace value $\alpha(\mathcal{S})$ of the WTTS $\mathcal{S}$ of σ and $\mathcal{W}$.

Example 5.8. By changing the semiring $\mathbb{S}$ and the cost function κ, various semantics can be defined by the trace value. Let $\mathbb{D} = \mathbb{R}$. For the Boolean semiring $(\{\top, \bot\}, \vee, \wedge, \bot, \top)$ in Example 5.4, the following function κ_b is a prototypical example of a cost function, where $u \in \Phi(\mathbb{V}, \mathbb{D})$ and $(a_1 a_2 \dots a_m) \in (\mathbb{D}^{\mathbb{V}})^{\circledast}$.

$$\kappa_b(u, (a_1 a_2 \dots a_m)) = \bigwedge_{i=1}^{m} \kappa_b(u, (a_i))$$

$$\kappa_b(\bigwedge_{i=1}^{n} (v_i \bowtie_i d_i), (a)) = \bigwedge_{i=1}^{n} \kappa_b(v_i \bowtie_i d_i, (a)) \quad \text{where } \bowtie_i \in \{>, \geq, \leq, <\}$$

$$\kappa_b(v \bowtie d, (a)) = \begin{cases} \top & \text{if } a \models v \bowtie d \\ \bot & \text{if } a \not\models v \bowtie d \end{cases}$$

For the sup-inf semiring $(\mathbb{R} \amalg \{\pm\infty\}, \sup, \inf, -\infty, +\infty)$ in Example 5.4, the trace value defined by the cost function κ_r in Fig. 5.3 captures the essence of the so-called space robustness [FP09,

BFMU17]. For the tropical semiring $(\mathbb{R} \amalg \{+\infty\}, \inf, +, +\infty, 0)$ in Example 5.4, an example cost function κ_t is as follows.

$$\begin{aligned}
\kappa_t(u, (a_1 a_2 \dots a_m)) &= \sum_{i=1}^{n} \kappa_r(u, (a_i)) \\
\kappa_t(\bigwedge_{i=1}^{n} (v_i \bowtie_i d_i), (a)) &= \sum_{i=1}^{n} \kappa_t(v_i \bowtie_i d_i, (a)) \quad \text{where } \bowtie_i \in \{>, \geq, \leq, <\} \\
\kappa_t(v \succ d, (a)) &= a(v) - d \quad \text{where } \succ \in \{\geq, >\} \\
\kappa_t(v \prec d, (a)) &= d - a(v) \quad \text{where } \prec \in \{\leq, <\}
\end{aligned}$$

◇

The preference between these semantics depends on the usage scenario. For example, as shown in Example 1.13, the quantitative semantics with sup-plus semiring compensates the "unsafe" values by "safe" values. This semantics may be an option, e. g., for costs or energy consumption, but not suitable, e. g., for physical distance because we cannot compensate a collision of cars with very long inter-vehicular distance. Nevertheless, the quantitative semantics by the sup-inf semantics would be the standard semantics thanks to the intuition in Fig. 1.16.

Example 5.9. Let $\mathcal{W} = (\mathcal{A}, \kappa)$ be a TSWA over $\mathbb{R}$ and $\mathbb{S}$, where $\mathcal{A}$ is the TSA over $\mathbb{R}$ in Fig. 5.3, σ be the signal $\sigma = \{x = 10\}^{2.5}\{x = 40\}^{1.0}\{x = 60\}^{3.0}$. When $\mathbb{S} = (\mathbb{R}\amalg\{\pm\infty\}, \sup, \inf, -\infty, +\infty)$ and κ is the cost function κ_r in Example 5.8, we have $\alpha(\sigma, \mathcal{W}) = 5$. When $\mathbb{S} = (\mathbb{R} \amalg \{+\infty\}, \inf, +, +\infty, 0)$ and κ is the cost function κ_t in Example 5.8, we have $\alpha(\sigma, \mathcal{W}) = 35$. ◇

5.4 Quantitative timed pattern matching

Using TSWAs, we formulate *quantitative timed pattern matching* as follows.

Quantitative timed pattern matching problem:
INPUT: a TSWA $\mathcal{W}$ over the data domain $\mathbb{D}$ and complete semiring $\mathbb{S}$, and a signal $\sigma \in \mathcal{T}(\mathbb{D}^{\mathbb{V}})$
PROBLEM: compute the *quantitative matching function* $\mathcal{M}(\sigma, \mathcal{W})\colon \mathrm{dom}(\sigma) \to S$ such that $(\mathcal{M}(\sigma, \mathcal{W}))(t, t') = \alpha(\sigma([t, t)), \mathcal{W})$, where $\mathrm{dom}(\sigma) = \{(t, t') \mid 0 \leq t < t' \leq |\sigma|\}$ and S is the underlying set of $\mathbb{S}$.

Example 5.10. Let $\mathcal{W}$ be the TSWA shown in Fig. 5.3, which is defined over $\mathbb{R}$ and the sup-inf semiring $(\mathbb{R}\amalg\{\pm\infty\}, \sup, \inf, -\infty, +\infty)$, and σ be the signal $\sigma = \{x = 10\}^{7.5}\{x = 40\}^{10.0}\{x = 60\}^{13.0}$. The quantitative matching function $\mathcal{M}(\sigma, \mathcal{W})$ is as follows. Fig. 5.1 shows an illustration.

$$(\mathcal{M}(\sigma, \mathcal{W}))(t, t') = \begin{cases} 5 & \text{when } \begin{array}{l} t \in [0, 7.5), t' \in (0, 17.5], t' - t < 10 \text{ or} \\ t \in [0, 7.5), t' \in (10, 17.5], t' - t \in [10, 15) \end{array} \\ -25 & \text{when } \begin{array}{l} t \in [7.5, 17.5), t' \in (7.5, 27.5], t' - t < 10 \text{ or} \\ t \in [2.5, 17.5), t' \in (17.5, 27.5], t' - t \in [10, 15) \end{array} \\ -45 & \text{when } \begin{array}{l} t \in [17.5, 30.5), t' \in (17.5, 30.5], t' - t < 10 \text{ or} \\ t \in [12.5, 30.5), t' \in (27.5, 30.5], t' - t \in [10, 15) \end{array} \end{cases}$$

◇

Although the domain $\{(t,t') \mid 0 \le t < t' \le |\sigma|\}$ of the quantitative matching function $\mathcal{M}(\sigma,\mathcal{W})$ is an infinite set, $\mathcal{M}(\sigma,\mathcal{W})$ is a piecewise-constant function with finitely many pieces. Moreover, each piece of $\mathcal{M}(\sigma,\mathcal{W})$ can be represented by zones in Section 2.3, much like the case of timed pattern matching in Section 2.4. We denote the set of zones over $\mathbb{X}$ by $\mathcal{Z}(\mathbb{X})$.

Theorem 5.11. *For any TSWA $\mathcal{W}$ over $\mathbb{D}$ and $\mathbb{S}$ and for any signal $\sigma \in \mathcal{T}(\mathbb{D}^{\mathbb{V}})$, there is a finite set $\{(Z_1,s_1),(Z_2,s_2),\dots,(Z_n,s_n)\} \subseteq \mathcal{Z}(\{x_{\mathrm{begin}},x_{\mathrm{end}}\}) \times S$ such that $Z_1,Z_2,\dots,Z_n$ is a partition of the domain $\{(t,t') \mid 0 \le t < t' \le |\sigma|\}$, and for any $[t,t') \subseteq \mathbb{R}_{\ge 0}$ satisfying $0 \le t < t' \le |\sigma|$, there exists $i \in \{1,2,\dots,n\}$ and $\nu \in Z_i$ satisfying $\nu(x_{\mathrm{begin}}) = t$, $\nu(x_{\mathrm{end}}) = t'$, and $(\mathcal{M}(\sigma,\mathcal{W}))(t,t') = s_i$.* □

5.5 Trace value computation by shortest distance

We present an algorithm to compute the trace values $\alpha(\mathcal{S})$. Since a WTTS possibly has infinitely many states and transitions (see Definition 5.6), we need a finite abstraction of it. We use zone-based abstraction for what we call *weighted symbolic timed transition systems* (*WSTTSs*). In addition to the clock variables in the TSA, we introduce a fresh clock variable T to represent the absolute time.

Definition 5.12 (weighted symbolic timed transition system)**.** For a TSWA $\mathcal{W} = (\mathcal{A},\kappa)$ over the data domain $\mathbb{D}$ and complete semiring $\mathbb{S}$, and a signal $\sigma = a_1^{\tau_1}a_2^{\tau_2}\dots a_n^{\tau_n} \in \mathcal{T}(\mathbb{D}^{\mathbb{V}})$, where $\mathcal{A} = (\mathbb{V},L,L_0,L_F,\mathbb{X},\Delta,\Lambda)$, the *weighted symbolic timed transition system* (*WSTTS*) is the 5-tuple $\mathcal{S}^{\mathrm{sym}} = (Q^{\mathrm{sym}},Q_0^{\mathrm{sym}},Q_F^{\mathrm{sym}},\to^{\mathrm{sym}},W^{\mathrm{sym}})$ defined as follows.

- $Q^{\mathrm{sym}} = \{(\ell,Z,\overline{a}) \in L \times \mathcal{Z}(\mathbb{X} \amalg \{T\}) \times (\mathbb{D}^{\mathbb{V}})^{\circledast} \mid Z \neq \emptyset, \forall \nu \in Z.\, \nu(T) \le |\sigma|, \overline{a} = \varepsilon \text{ or } \overline{a} \circ \sigma(\nu(T)) = \overline{a}\}$
- $Q_0^{\mathrm{sym}} = \{(\ell_0,\{\mathbf{0}_{\mathbb{X}\amalg\{T\}}\},\varepsilon) \mid \ell_0 \in L_0\}$
- $Q_F^{\mathrm{sym}} = \{(\ell_F,Z,\varepsilon) \mid \ell_F \in L_F, \exists \nu \in Z.\, \nu(T) = |\sigma|\}$
- $\to^{\mathrm{sym}} \subseteq Q^{\mathrm{sym}} \times Q^{\mathrm{sym}}$ is the relation such that $((\ell,Z,\overline{a}),(\ell',Z',\overline{a'})) \in \to^{\mathrm{sym}}$ if and only if one of the following holds.

  (transition of $\mathcal{A}$) there exists $(\ell,g,R,\ell') \in \Delta$, satisfying $Z' = \{[\nu]_R \mid \nu \in Z, \nu \models g\}$, $\overline{a} \neq \varepsilon$, and $\overline{a'} = \varepsilon$.

  (punctual time elapse) $\ell = \ell'$, $\overline{a'} = \overline{a} \circ \mathit{Values}(\sigma([\tilde{\nu}(T),\tilde{\nu}'(T)])))$, and there is $i \in \{1,2,\dots,n\}$ satisfying $Z' = \{\nu+\tau \mid \nu \in Z, \tau \in \mathbb{R}_{>0}, \nu(T)+\tau = \sum_{j=0}^{i}\tau_j\}$, where $\tilde{\nu} \in Z, \tilde{\nu}' \in Z'$[1].

  (non-punctual time elapse) $\ell = \ell'$, $\overline{a'} = \overline{a} \circ \mathit{Values}(\sigma([\tilde{\nu}(T),\tilde{\nu}'(T)])))$, and there is $i \in \{1,2,\dots,n\}$ satisfying $Z' = \{\nu+\tau \mid \nu \in Z, \tau \in \mathbb{R}_{>0}, \sum_{j=0}^{i-1}\tau_j < \nu(T)+\tau < \sum_{j=0}^{i}\tau_j\}$, where $\tilde{\nu} \in Z, \tilde{\nu}' \in Z'$.

- $W^{\mathrm{sym}}((\ell,Z,\overline{a}),(\ell',Z',\overline{a'}))$ is $\kappa(\Lambda(\ell),\overline{a})$ if $\overline{a'} = \varepsilon$; and $e_{\otimes}$ if $\overline{a'} \neq \varepsilon$

◇

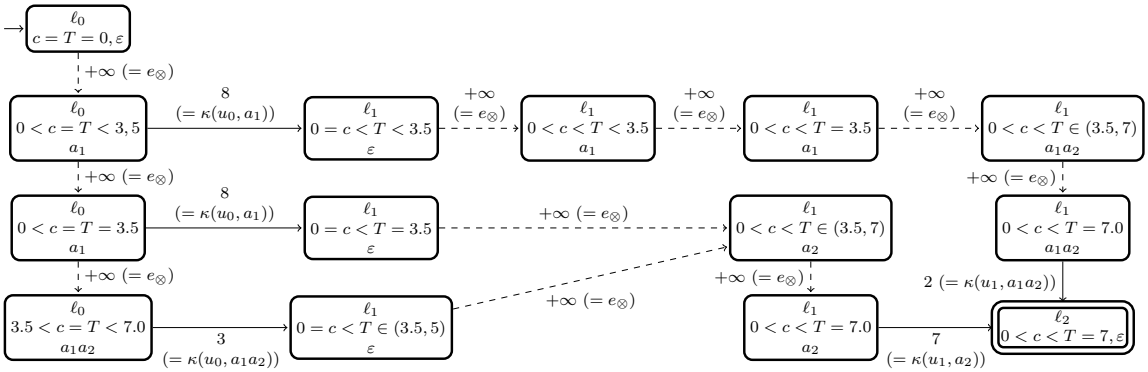


Figure 5.5: WSTTS $\mathcal{S}^{\mathrm{sym}}$ of the TSWA $\mathcal{W}$ in Fig. 5.3 and the signal $\sigma = a_1^{3.5}a_2^{3.5}$, where $u_0 = x < 15$, $u_1 = x > 5$, $a_1 = \{x = 7\}$, and $a_2 = \{x = 12\}$. The states unreachable from the initial state or unreachable to the accepting state are omitted. The transition for time elapse which can be represented by the composition of other transitions are also omitted. A dashed transition is for the time elapse and a solid transition is for a transition of $\mathcal{A}$.

Although the state space Q^{sym} of the WSTTS $\mathcal{S}^{\mathrm{sym}}$ may be infinite, there are only finitely many states reachable from Q_0^{sym} and therefore, we can construct the reachable part of $\mathcal{S}^{\mathrm{sym}}$. See Appendix B.1 for the proof. An example of a WSTTS is shown in Fig. 5.5. For a WSTTS $\mathcal{S}^{\mathrm{sym}}$, we define the *symbolic trace value* $\alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}})$ as the following shortest distance from Q_0^{sym} to Q_F^{sym}.

$$\mathrm{Dist}(Q_0^{\mathrm{sym}}, Q_F^{\mathrm{sym}}, Q^{\mathrm{sym}}, \to^{\mathrm{sym}}, W^{\mathrm{sym}})$$

Theorem 5.13. *Let $\mathcal{W}$ be a TSWA over $\mathbb{D}$ and $\mathbb{S}$, and $\sigma \in \mathcal{T}(\mathbb{D}^{\mathbb{V}})$ be a signal. Let $\mathcal{S}$ and $\mathcal{S}^{\mathrm{sym}}$ be the WTTS (in Definition 5.6) and WSTTS of $\mathcal{W}$ and σ, respectively. If $\mathbb{S}$ is idempotent, we have $\alpha(\mathcal{S}) = \alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}})$.* □

Because of Theorem 5.13, we can compute the trace value $\alpha(\mathcal{S})$ by *i*) constructing the reachable part of $\mathcal{S}^{\mathrm{sym}}$; and *ii*) computing the symbolic trace value $\alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}})$ using an algorithm for the shortest distance problem. For example, the symbolic trace value of the WSTTS in Fig. 5.5 is $\alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}}) = \max\{\min\{8, 2\}, \min\{8, 7\}, \min\{3, 7\}\} = 7$. However, this method requires the whole signal to compute the trace value, and it does not suit for the use in online quantitative timed pattern matching. Instead, we define the *intermediate weight* $weight_i$ and give an incremental algorithm to compute the trace value $\alpha(\mathcal{S})$. Intuitively, for each state $(\ell, Z, \overline{a}) \in Q^{\mathrm{sym}}$ of the WSTTS $\mathcal{S}^{\mathrm{sym}}$, the intermediate weight $weight_i$ assign the shortest distance to reach $(\ell, Z, \overline{a})$ by reading the subsignal $a_1^{\tau_1}a_2^{\tau_2}\dots a_i^{\tau_i}$ of $\sigma = a_1^{\tau_1}a_2^{\tau_2}\dots a_n^{\tau_n}$.

Definition 5.14 ($incr$, $weight_i$)**.** For a TSWA $\mathcal{W} = (\mathcal{A}, \kappa)$ over the data domain $\mathbb{D}$ and complete semiring $\mathbb{S}$, $a \in \mathbb{D}^{\mathbb{V}}$, and $t \in \mathbb{R}_{>0}$, the *increment function*

$$incr(a, t) \colon \mathcal{P}(L \times \mathcal{Z}(\mathbb{X} \amalg \{T\}) \times (\mathbb{D}^{\mathbb{V}})^{\circledast} \times S) \to \mathcal{P}(L \times \mathcal{Z}(\mathbb{X} \amalg \{T\}) \times (\mathbb{D}^{\mathbb{V}})^{\circledast} \times S)$$

is as follows, where $\mathcal{A} = (\mathbb{V}, L, L_0, L_F, \mathbb{X}, \Delta, \Lambda)$ and $(Q_{a,t}^{\mathrm{sym}}, Q_{a,t,0}^{\mathrm{sym}}, Q_{a,t,F}^{\mathrm{sym}}, \to_{a,t}^{\mathrm{sym}}, W_{a,t}^{\mathrm{sym}})$ is the WSTTS of a^t and $\mathcal{W}$.

$$incr(a, t)(w) = \{(\ell', Z', \overline{a'}, s') \in L \times \mathcal{Z}(\mathbb{X} \amalg \{T\}) \times (\mathbb{D}^{\mathbb{V}})^{\circledast} \times S \mid \forall \nu' \in Z'.\, \nu'(T) = t,$$
$$s' = \bigoplus_{(\ell, Z, \overline{a}, s) \in w} s \otimes \mathrm{Dist}(\{(\ell, Z, \overline{a})\}, \{(\ell', Z', \overline{a'})\}, Q_{a,t}^{\mathrm{sym}}, \to_{a,t}^{\mathrm{sym}}, W_{a,t}^{\mathrm{sym}})\}$$

[1]The choice of $\tilde{\nu}$ and $\tilde{\nu}'$ does not change $\sigma(\tilde{\nu}(T))$ and $\sigma(\tilde{\nu}'(T))$ due to the definition of Q^{sym}.

Algorithm 8: Incremental algorithm for trace value computation

Input: A WSTTS $\mathcal{S}^{\mathrm{sym}} = (Q^{\mathrm{sym}}, Q_0^{\mathrm{sym}}, Q_F^{\mathrm{sym}}, \to^{\mathrm{sym}}, W^{\mathrm{sym}})$ of $\sigma = a_1^{\tau_1} a_2^{\tau_2} \dots a_n^{\tau_n}$ and $\mathcal{W}$
Output: R is the symbolic trace value $\alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}})$

$weight \leftarrow \{(\ell_0, \{\mathbf{0}_{\mathbb{X}\amalg\{T\}}\}, \varepsilon, e_\otimes) \mid \ell_0 \in L_0\}$; $R \leftarrow e_\oplus$ `// initialize`
for $i \in \{1, 2, \dots, n\}$ **do**
  $weight \leftarrow incr(a_i, T_i)$, where $T_i = \sum_{k=1}^{i} \tau_k$
  `// We have` $weight = weight_i$.
for $(\ell, Z, \overline{a}, s) \in weight$ **do**
  if $(\ell, Z, \overline{a}) \in Q_F^{\mathrm{sym}}$ **then**
    $R \leftarrow R \oplus s$

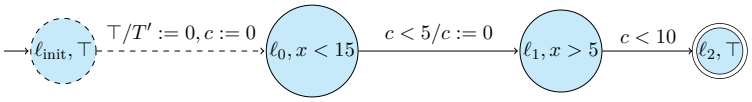


Figure 5.6: Matching automaton $\mathcal{A}_{\mathrm{match}}$ for the TSA $\mathcal{A}$ shown in Fig. 5.3. The fresh initial location ℓ_{init} and the transition to the original initial location ℓ_0 are added.

For a TSWA $\mathcal{W}$ over $\mathbb{D}$ and $\mathbb{S}$, a signal $\sigma = a_1^{\tau_1} a_2^{\tau_2} \dots a_n^{\tau_n}$, and $i \in \{1, 2, \dots, n\}$, the *intermediate weight* $weight_i$ is defined as follows, where $T_j = \sum_{k=1}^{j} \tau_k$.

$$weight_i = (incr(a_i, T_i) \circ \dots \circ incr(a_1, T_1))(\{(\ell_0, \{\mathbf{0}_{\mathbb{X}\amalg\{T\}}\}, \varepsilon, e_\otimes) \mid \ell_0 \in L_0\})$$

◇

Because of the following theorem, we can incrementally compute the symbolic trace value $\alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}})$, which is equal to the trace value $\alpha(\sigma, \mathcal{W})$, by Algorithm 8.

Theorem 5.15. *For any WSTTS $\mathcal{S}^{\mathrm{sym}}$ of a signal $\sigma = a_1^{\tau_1} a_2^{\tau_2} \dots a_n^{\tau_n}$ and a TSWA $\mathcal{W}$ over $\mathbb{D}$ and $\mathbb{S}$, we have the following, where Q_F^{sym} is the accepting states of $\mathcal{S}^{\mathrm{sym}}$.*

$$\alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}}) = \bigoplus_{(\ell, Z, \overline{a}) \in Q_F^{\mathrm{sym}}} \bigoplus_{(\ell, Z, \overline{a}, s) \in weight_n} s$$

□

5.6 Online algorithm for quantitative timed pattern matching

In quantitative timed pattern matching, we compute the trace value $\alpha(\sigma([t, t')), \mathcal{W})$ for each subsignal $\sigma([t, t'))$. In order to try matching for each subsignal $\sigma([t, t'))$, we construct the *matching automaton* [BFN+18] $\mathcal{A}_{\mathrm{match}}$ from the TSA $\mathcal{A}$. The matching automaton $\mathcal{A}_{\mathrm{match}}$ is constructed by adding a new clock variable T' and a new initial location ℓ_{init} to the TSA $\mathcal{A}$. The new clock variable T' represents the duration from the beginning t of the subsignal $\sigma([t, t'))$. The new location ℓ_{init} is used to start the subsignal in the middle of the signal. We add transitions from ℓ_{init} to each initial location ℓ_{init} of $\mathcal{A}$, resetting all the clock variables. Fig. 5.6 shows an example of $\mathcal{A}_{\mathrm{match}}$. We also define the auxiliary $incr_<$ for our online algorithm for quantitative timed pattern matching.

Definition 5.16 (matching automaton [BFN+18] $\mathcal{A}_{\mathrm{match}}$)**.** For a TSA $\mathcal{A} = (\mathbb{V}, L, L_0, L_F, \mathbb{X}, \Delta, \Lambda)$ over $\mathbb{D}$, the *matching automaton* is the TSA $\mathcal{A}_{\mathrm{match}} = (\mathbb{V}, L\amalg\{\ell_{\mathrm{init}}\}, \{\ell_{\mathrm{init}}\}, L_F, \mathbb{X}\amalg\{T'\}, \Delta', \Lambda')$

Algorithm 9: Online algorithm for quantitative timed pattern matching

Input: A signal $\sigma = a_1^{\tau_1} a_2^{\tau_2} \dots a_n^{\tau_n}$ and a TSWA $\mathcal{W} = (\mathcal{A}, \kappa)$
Output: M is the quantitative matching function $\mathcal{M}(\sigma, \mathcal{W})$.

1 $\mathcal{A}_{\mathrm{match}} \leftarrow$ the matching automaton of $\mathcal{A}$;
2 $\mathit{weight} \leftarrow \{(\ell_0, \{\mathbf{0}_{\mathbb{X} \amalg \{T,T'\}}\}, \varepsilon, e_\otimes) \mid \ell_0 \in L_0\}$; for each $[t,t') \subseteq [0, |\sigma|)$, $M(t,t') \leftarrow e_\oplus$;
3 **for** $i \in \{1, 2, \dots, n\}$ **do**
4 $\quad \mathit{weight} \leftarrow (\mathit{incr}_<(a_i, T_i))(\mathit{weight})$, where $T_i = \sum_{k=1}^{i} \tau_k$;
5 $\quad$ **for** $(\ell, Z, \varepsilon, s) \in \mathit{weight}, \nu \in Z$ **do**
6 $\quad\quad$ **if** $\ell \in L_F$ **then**
7 $\quad\quad\quad M(\nu(T') - \nu(T), \nu(T')) \leftarrow M(\nu(T') - \nu(T), \nu(T')) \oplus s.$;
8 $\quad \mathit{weight} \leftarrow (\mathit{incr}(a_i, T_i))(\mathit{weight})$, where $T_i = \sum_{k=1}^{i} \tau_k$;

over $\mathbb{D}$, where the transition is $\Delta' = \Delta \amalg \{(\ell_{\mathrm{init}}, \top, \mathbb{X} \amalg \{T'\}, \ell_{\mathrm{init}}) \mid \ell_{\mathrm{init}} \in L_0\}$, the labeling function is $\Lambda'(\ell_{\mathrm{init}}) = \top$, and $\Lambda'(\ell) = \Lambda(\ell)$ for $\ell \in L$. ◇

Definition 5.17 ($\mathit{incr}_<$)**.** For a TSWA $\mathcal{W} = (\mathcal{A}, \kappa)$ over the data domain $\mathbb{D}$ and complete semiring $\mathbb{S}$, $a \in \mathbb{D}^{\mathbb{V}}$, and $t \in \mathbb{R}_{>0}$, the partial increment function

$$\mathit{incr}_<(a,t) \colon \mathcal{P}(L \times \mathcal{Z}(\mathbb{X} \amalg \{T\}) \times (\mathbb{D}^{\mathbb{V}})^{\circledast} \times S) \to \mathcal{P}(L \times \mathcal{Z}(\mathbb{X} \amalg \{T\}) \times (\mathbb{D}^{\mathbb{V}})^{\circledast} \times S)$$

is as follows, where $\mathcal{A} = (\mathbb{V}, L, L_0, L_F, \mathbb{X}, \Delta, \Lambda)$ and $(Q^{\mathrm{sym}}_{a,t}, Q^{\mathrm{sym}}_{a,t,0}, Q^{\mathrm{sym}}_{a,t,F}, \to^{\mathrm{sym}}_{a,t}, W^{\mathrm{sym}}_{a,t})$ is the WSTTS of the TSWA $\mathcal{W}$ and the constant signal a^t.

$$\mathit{incr}_<(a,t)(w) = \{(\ell', Z', \overline{a'}, s') \in L \times \mathcal{Z}(\mathbb{X} \amalg \{T\}) \times (\mathbb{D}^{\mathbb{V}})^{\circledast} \times S \mid \forall \nu' \in Z'.\, \nu'(T) < t,$$
$$s' = \bigoplus_{(\ell, Z, \overline{a}, s) \in w} s \otimes \mathrm{Dist}(\{(\ell, Z, \overline{a})\}, \{(\ell', Z', \overline{a'})\}, Q^{\mathrm{sym}}_{a,t}, \to^{\mathrm{sym}}_{a,t}, W^{\mathrm{sym}}_{a,t})\}$$

◇

Algorithm 9 shows our online algorithm for quantitative timed pattern matching. We construct the matching automaton $\mathcal{A}_{\mathrm{match}}$ from the TSA $\mathcal{A}$ (line 1), and we try matching by reading each constant subsignal $a_i^{\tau_i}$ of the signal $\sigma = a_1^{\tau_1} a_2^{\tau_2} \dots a_n^{\tau_n}$ much like the illustration in Fig. 5.4. For each i, first, we consume a prefix $a_i^{\tau_i'}$ of $a_i^{\tau_i} = a_i^{\tau_i'} a_i^{\tau_i''}$ and update the intermediate weight *weight* (line 4). Then, we update the result M for each weight $(\ell, Z, \varepsilon, s) \in \mathit{weight}$ labelled with an accepting location (line 7). Finally, we consume the remaining part $a_i^{\tau_i''}$ and update the intermediate weight *weight* (line 8).

Complexity discussion In general, the time and space complexities of Algorithm 9 are polynomial to the length n of the signal $\sigma = a_1^{\tau_1} a_2^{\tau_2} \dots a_n^{\tau_n}$ due to the bound of the size of the reachability part of the WSTTS. On the other hand, if the TSWA has a time-bound and the sampling frequency of the signal is also bounded (such as in Figs. 5.7 and 5.8), time and space complexities are linear and constant to the length n of the signal, respectively.

5.7 Experiments

We implemented our online algorithm for quantitative timed pattern matching in C++ and conducted experiments to answer the following research questions. We suppose that the input

Table 5.1: Execution time and memory usage under long signals for Overshoot and Ringing for sup-inf semiring

$\lvert\sigma\rvert$	Execution Time [s] (Overshoot)	Memory Usage [KiB] (Overshoot)	Execution Time [s] (Ringing)	Memory Usage [KiB] (Ringing)
60,000	1.59	7,196.20	13.06	7,949.20
120,000	3.19	7,143.60	26.17	7,924.00
180,000	4.85	7,197.80	39.03	7,960.00
240,000	6.44	7,160.60	52.06	7,977.60
300,000	8.07	7,165.60	65.19	7,912.20
360,000	9.71	7,147.20	78.13	7,953.60
420,000	11.40	7,197.80	91.37	7,961.20
480,000	13.01	7,195.60	104.12	7,933.20
540,000	14.60	7,160.00	117.49	7,966.40
600,000	16.28	7,202.80	131.00	7,972.40

piecewise-constant signals are interpolations of the actual signals by sampling. Our implementation QTPM is in https://github.com/MasWag/qtpm.

RQ1 Is the practical performance of Algorithm 9 realistic?

RQ2 Is Algorithm 9 online capable, i. e., does it perform in linear time and constant space, with respect to the number of the entries in the signal?

RQ3 Can Algorithm 9 handle denser logs, i. e., what is the performance with respect to the sampling frequency of the signal?

RQ4 What is the shortest sampling interval QTPM can handle?

We conducted the experiments on an Amazon EC2 c4.large instance (2 vCPUs and 3.75 GiB RAM) running Ubuntu 18.04 LTS (64 bit). We compiled QTPM by GCC-4.9.3. For the measurement of the execution time and memory usage, we used GNU time and took an average of 20 executions. We could not compare with [BFMU17] because their implementation is not publicly available.

As the complete semiring $\mathbb{S}$, we used the sup-inf semiring $(\mathbb{R} \amalg \{\pm\infty\}, \sup, \inf, -\infty, +\infty)$ and the tropical semiring $(\mathbb{R}\amalg\{+\infty\}, \inf, +, +\infty, 0)$ in Example 5.4. We used the cost functions κ_r in Example 5.8 for the sup-inf semiring, and κ_t in Example 5.8 for the tropical semiring.

Benchmarks We used the automotive benchmark problems shown in Figs. 5.7 to 5.9. A summary of quantitative timed pattern matching is on the right of each figure. The specified behaviors in the TSWAs are taken from ST-Lib [KJD+16] and known to be useful for automotive control applications.

RQ1: practical performance Figs. 5.10 and 5.11 and Tables 5.1 to 5.4 show the execution time and memory usage of our quantitative timed pattern matching for the TSWAs $\mathcal{W}$ and signals σ. Here, we fixed the sampling frequency to be 0.1 Hz and changed the duration $|\sigma|$ of the signal from 60,000 s to 600,000 s in Overshoot and Ringing, and from 1,000 s to 10,000 s in Overshoot (Unbounded).

In Fig. 5.10, we observe that QTPM handles the log with 60,000 entries for Overshoot, in less than 20 s with less than 7.1 MiB of memory usage, and for Ringing, in about 1 or 2 minutes with less than 7.8 MiB of memory usage. In Fig. 5.11, we observe that QTPM handles the log with 10,000 entries in less than 120 s with less than 250 MiB of memory usage for

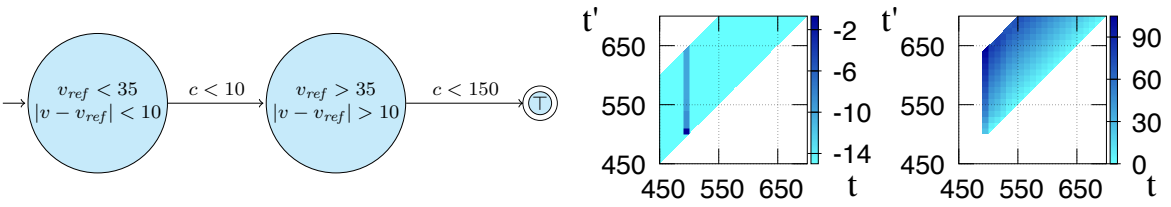


Figure 5.7: Overshoot: The set of input signals is generated by the cruise control model [cru]. The TSA is for the settling when the reference value of the velocity is changed from $v_{ref} < 35$ to $v_{ref} > 35$. The left and right maps are for the sup-inf and tropical semirings, respectively.

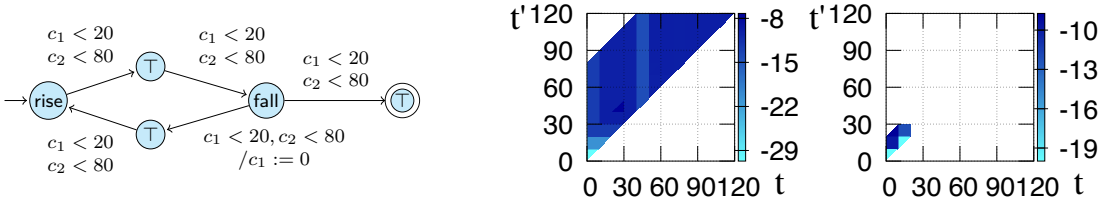


Figure 5.8: Ringing: The set of input signals is generated by the same model [cru] as that in Overshoot. The TSA is for the frequent rise and fall of the signal in 80 s. The constraints rise and fall are $\mathsf{rise} = v(t) - v(t-10) > 10$ and $\mathsf{fall} = v(t) - v(t-10) < -10$. The left and right maps are for the sup-inf and tropical semirings, respectively.

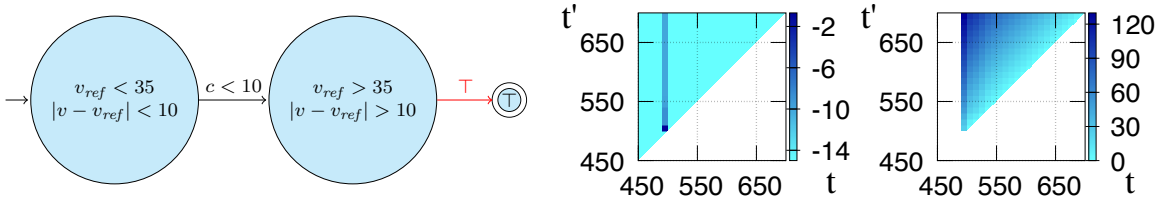


Figure 5.9: Overshoot (Unbounded): The set of input signals is generated by the same model [cru] as that in Overshoot. The TSA is almost the same as that in Overshoot, but the time-bound ($c < 150$) is removed. The left and right maps are for the sup-inf and tropical semirings, respectively.

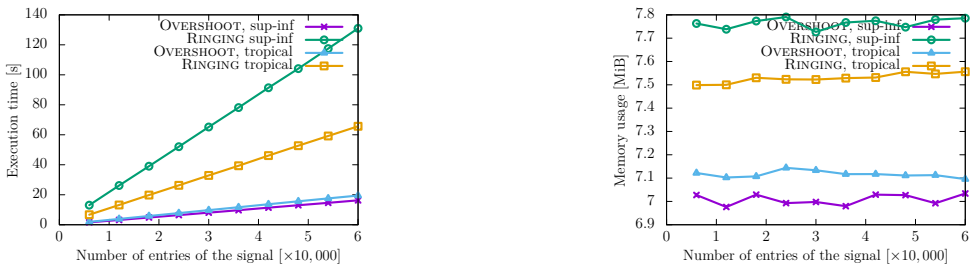


Figure 5.10: Change in execution time (left) and memory usage (right) for Overshoot and Ringing with the number of the entries of the signals

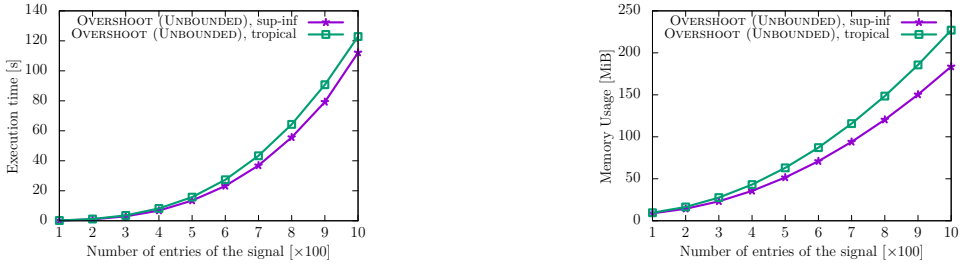


Figure 5.11: Change in execution time (left) and memory usage (right) for Overshoot (Unbounded) with the number of the entries of the signals

Table 5.2: Execution time and memory usage under long signals for Overshoot (Unbounded) for sup-inf semiring

$\|\sigma\|$	Execution Time [s]	Memory Usage [KiB]
1,000	0.14	9,050.60
2,000	0.92	14,633.80
3,000	2.97	23,667.80
4,000	6.79	36,452.00
5,000	13.46	52,806.80
6,000	23.22	72,639.40
7,000	36.86	96,220.80
8,000	55.63	123,263.00
9,000	79.23	153,900.00
10,000	112.15	188,054.00

Table 5.3: Execution time and memory usage under long signals for Overshoot and Ringing for tropical semiring

$\|\sigma\|$	Execution Time [s] (Overshoot)	Memory Usage [KiB] (Overshoot)	Execution Time [s] (Ringing)	Memory Usage [KiB] (Ringing)
60,000	1.92	7,292.80	6.67	7,678.80
120,000	3.86	7,273.00	13.17	7,680.00
180,000	5.83	7,278.00	19.79	7,710.80
240,000	7.76	7,315.40	26.30	7,704.00
300,000	9.75	7,304.80	32.88	7,703.20
360,000	11.69	7,287.80	39.35	7,709.40
420,000	13.67	7,288.00	46.11	7,712.00
480,000	15.64	7,281.80	52.73	7,737.20
540,000	17.57	7,283.40	59.20	7,728.00
600,000	19.48	7,266.00	65.67	7,737.60

Table 5.4: Execution time and memory usage under long signals for Overshoot (Unbounded) for tropical semiring

$\|\sigma\|$	Execution Time [s]	Memory Usage [KiB]
1,000	0.18	9,799.40
2,000	1.16	16,840.40
3,000	3.59	28,256.20
4,000	8.22	44,139.40
5,000	15.75	64,509.40
6,000	27.34	89,218.40
7,000	43.33	118,498.00
8,000	64.29	152,075.00
9,000	90.74	190,065.00
10,000	122.87	232,474.00

Overshoot (Unbounded). Although the quantitative timed pattern matching problem is complex, we conclude that its practical performance is realistic.

RQ2: change in speed and memory usage with signal size Figs. 5.10 and 5.11 and Tables 5.1 to 5.4 show the execution time and memory usage of our quantitative timed pattern matching. See RQ1 for the detail of our experimental setting.

In Fig. 5.10, for the TSAs with time-bound, we observe that the execution time is linear with respect to the duration $|\sigma|$ of the input signals and the memory usage is more or less constant with respect to the duration $|\sigma|$ of the input signals. This performance is essential for a monitor to keep monitoring a running system.

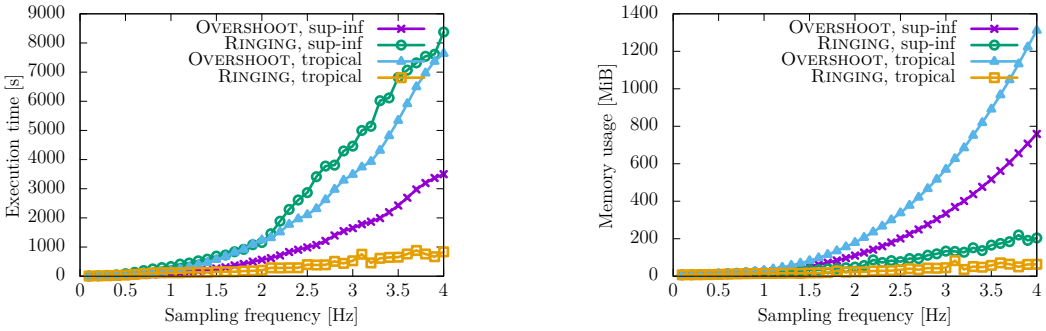


Figure 5.12: Change in execution time (left) and memory usage (right) for Overshoot and Ringing with the sampling frequency

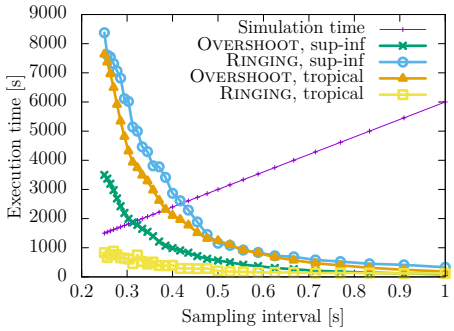


Figure 5.13: Change in execution time for Overshoot and Ringing with the sampling interval

In Fig. 5.11, for the TSA without any time-bound, we observe that the execution time is cubic and the memory usage is quadratic with respect to the number of the entries in $|\sigma|$. The memory usage increases quadratically with the number of the entries because the intermediate weight $weight_j$ has an entry for each initial interval $[\tau_i, \tau_{i+1})$ of the trimming and for each interval $[\tau_k, \tau_{k+1})$ where the transition occurred. The execution time increases cubically with respect to the number of the entries because the shortest distance is computed for each entry of $weight_j$. However, we note that our quantitative timed pattern matching still works when the number of the entries is relatively small.

RQ3: change in speed and memory usage with sampling frequency Fig. 5.12 and Tables 5.5 and 5.6 show the execution time and memory usage of QTPM for each TSWA $\mathcal{W}$ and signal σ of Overshoot and Ringing. Here, we fixed the number of the entries to be 6,000 and changed the sampling frequency from 0.1 Hz to 4.0 Hz i. e., the simulated time is 6,000 sec. for 1.0Hz and 60,000 sec. for 0.1 Hz.

In Fig. 5.12, we observe that the execution time is cubic, and the memory usage is more or less quadratic with respect to the sampling frequency of the signals. This is because the number of the entries in a certain duration is linear to the sampling frequency, which increases the number of the reachability states of the WSTTSs quadratically.

RQ4: shortest sampling interval In signal monitoring, the shortest feasible sampling interval is an important efficiency criterion. This is because by making the sampling interval shorter, we can make the monitoring result more accurate, while it makes monitoring more computationally demanding because the algorithm has to handle more information in a certain length of time. Fig. 5.13 shows the execution time of QTPM with the sampling interval for

Table 5.5: Execution time and memory usage under high frequency for Overshoot and Ringing for sup-inf semiring

Sampling Freq. [Hz]	Execution Time [s] (Overshoot)	Memory Usage [KiB] (Overshoot)	Execution Time [s] (Ringing)	Memory Usage [KiB] (Ringing)
0.1	$1.64 \cdot 10^0$	7,123.8	$6.42 \cdot 10^0$	7,426.2
0.2	$4.76 \cdot 10^0$	7,438.4	$1.32 \cdot 10^1$	8,012.2
0.3	$9.35 \cdot 10^0$	7,986	$2.53 \cdot 10^1$	8,635.2
0.4	$1.50 \cdot 10^1$	8,730.8	$4.37 \cdot 10^1$	9,657.8
0.5	$2.26 \cdot 10^1$	9,731.4	$8.43 \cdot 10^1$	11,310.6
0.6	$3.24 \cdot 10^1$	11,160.2	$1.33 \cdot 10^2$	13,344.6
0.7	$4.41 \cdot 10^1$	13,153	$1.93 \cdot 10^2$	15,191.8
0.8	$5.74 \cdot 10^1$	15,716.2	$2.49 \cdot 10^2$	17,829.4
0.9	$7.46 \cdot 10^1$	18,890.4	$2.94 \cdot 10^2$	19,620.8
1.0	$9.36 \cdot 10^1$	22,688	$3.31 \cdot 10^2$	20,763
1.1	$1.18 \cdot 10^2$	27,293.6	$4.13 \cdot 10^2$	23,025.2
1.2	$1.45 \cdot 10^2$	32,736.4	$4.50 \cdot 10^2$	27,082.6
1.3	$1.80 \cdot 10^2$	38,954.8	$5.27 \cdot 10^2$	27,305.6
1.4	$2.11 \cdot 10^2$	45,924.2	$5.82 \cdot 10^2$	28,909.2
1.5	$2.64 \cdot 10^2$	54,422.2	$6.89 \cdot 10^2$	35,029.6
1.6	$3.05 \cdot 10^2$	63,616	$7.25 \cdot 10^2$	35,460.6
1.7	$3.70 \cdot 10^2$	74,029	$8.40 \cdot 10^2$	39,885.8
1.8	$4.17 \cdot 10^2$	85,779.6	$9.21 \cdot 10^2$	41,936
1.9	$4.99 \cdot 10^2$	98,694.6	$1.08 \cdot 10^3$	46,025
2.0	$5.64 \cdot 10^2$	112,683.4	$1.16 \cdot 10^3$	49,504.8
2.1	$6.21 \cdot 10^2$	128,929	$1.46 \cdot 10^3$	58,201.2
2.2	$7.12 \cdot 10^2$	145,465	$1.89 \cdot 10^3$	89,350
2.3	$8.30 \cdot 10^2$	163,803.6	$2.29 \cdot 10^3$	74,712.8
2.4	$9.07 \cdot 10^2$	183,922.8	$2.61 \cdot 10^3$	84,850.2
2.5	$9.87 \cdot 10^2$	206,233.8	$2.87 \cdot 10^3$	79,574.8
2.6	$1.07 \cdot 10^3$	230,562	$3.41 \cdot 10^3$	96,280
2.7	$1.21 \cdot 10^3$	255,283	$3.78 \cdot 10^3$	96,124.8
2.8	$1.40 \cdot 10^3$	283,075.6	$3.81 \cdot 10^3$	114,410
2.9	$1.55 \cdot 10^3$	311,450.2	$4.30 \cdot 10^3$	122,042.2
3.0	$1.65 \cdot 10^3$	342,199.8	$4.46 \cdot 10^3$	136,999.8
3.1	$1.78 \cdot 10^3$	378,197.2	$5.00 \cdot 10^3$	135,252
3.2	$1.87 \cdot 10^3$	410,698.8	$5.14 \cdot 10^3$	130,856.6
3.3	$1.99 \cdot 10^3$	448,147.6	$6.02 \cdot 10^3$	155,443.6
3.4	$2.19 \cdot 10^3$	487,956	$6.12 \cdot 10^3$	139,874.8
3.5	$2.42 \cdot 10^3$	529,348.2	$6.82 \cdot 10^3$	169,890.8
3.6	$2.69 \cdot 10^3$	575,203.8	$7.07 \cdot 10^3$	178,998.4
3.7	$2.97 \cdot 10^3$	621,682.6	$7.32 \cdot 10^3$	188,703.4
3.8	$3.20 \cdot 10^3$	671,744.6	$7.54 \cdot 10^3$	225,122.6
3.9	$3.37 \cdot 10^3$	724,041.8	$7.61 \cdot 10^3$	195,706.2
4.0	$3.50 \cdot 10^3$	777,275.4	$8.38 \cdot 10^3$	208,952.2

each TSWA $\mathcal{W}$ and signal σ of Overshoot and Ringing. We used the same experiment result as RQ3.

In Fig. 5.13, we observe that the execution time is shorter than the simulation time when the sampling interval is longer than 400 milliseconds. On the one hand, since the sampling interval of the sensors may be around 10 milliseconds to 100 milliseconds, QTPM is not fast enough to monitor the raw sensor values. An optimization by an algorithmic improvement is a future work. On the other hand, as Figs. 5.7 and 5.8 shows, the time-bound of the benchmarks Overshoot and Ringing are 150 seconds and 80 seconds. This is much longer than the available sampling intervals, and the performance of QTPM may be enough for some applications. Overall, although further optimization is a future work, our algorithm is still online capable for some usage scenarios.

Table 5.6: Execution time and memory usage under high frequency for OVERSHOOT and RINGING for tropical semiring

Sampling Freq. [Hz]	Execution Time [s] (OVERSHOOT)	Memory Usage [KiB] (OVERSHOOT)	Execution Time [s] (RINGING)	Memory Usage [KiB] (RINGING)
0.1	$2.08 \cdot 10^0$	7,097.8	$6.03 \cdot 10^0$	7,336.4
0.2	$6.92 \cdot 10^0$	7,444.4	$1.12 \cdot 10^1$	7,717.8
0.3	$1.45 \cdot 10^1$	8,192.6	$1.70 \cdot 10^1$	8,166.8
0.4	$2.41 \cdot 10^1$	9,212	$2.60 \cdot 10^1$	8,868.4
0.5	$3.72 \cdot 10^1$	10,920.2	$3.93 \cdot 10^1$	9,991.6
0.6	$5.57 \cdot 10^1$	13,281.8	$6.39 \cdot 10^1$	11,461.6
0.7	$7.88 \cdot 10^1$	16,497.8	$8.59 \cdot 10^1$	12,472.2
0.8	$1.06 \cdot 10^2$	20,371.2	$1.08 \cdot 10^2$	13,337
0.9	$1.43 \cdot 10^2$	25,628.8	$1.19 \cdot 10^2$	14,434.8
1.0	$1.85 \cdot 10^2$	31,701.2	$1.20 \cdot 10^2$	14,840
1.1	$2.44 \cdot 10^2$	39,399	$1.18 \cdot 10^2$	14,884.8
1.2	$3.14 \cdot 10^2$	48,346.8	$1.32 \cdot 10^2$	15,669.8
1.3	$3.95 \cdot 10^2$	58,905.2	$1.34 \cdot 10^2$	16,585.6
1.4	$4.69 \cdot 10^2$	70,886	$1.32 \cdot 10^2$	15,462.2
1.5	$5.73 \cdot 10^2$	84,744.2	$1.51 \cdot 10^2$	17,946.4
1.6	$6.77 \cdot 10^2$	100,155	$1.53 \cdot 10^2$	18,365.6
1.7	$8.21 \cdot 10^2$	118,195.8	$1.53 \cdot 10^2$	19,729.6
1.8	$9.19 \cdot 10^2$	138,113.2	$1.76 \cdot 10^2$	28,508.8
1.9	$1.10 \cdot 10^3$	160,375.4	$1.84 \cdot 10^2$	22,240
2.0	$1.24 \cdot 10^3$	183,886	$2.04 \cdot 10^2$	24,377.4
2.1	$1.32 \cdot 10^3$	211,464.6	$2.88 \cdot 10^2$	23,809.8
2.2	$1.52 \cdot 10^3$	241,210.8	$2.83 \cdot 10^2$	28,554
2.3	$1.78 \cdot 10^3$	273,248	$2.89 \cdot 10^2$	29,372.4
2.4	$1.97 \cdot 10^3$	307,758.8	$2.89 \cdot 10^2$	30,195.2
2.5	$2.11 \cdot 10^3$	345,901.6	$3.97 \cdot 10^2$	34,630.8
2.6	$2.32 \cdot 10^3$	386,944.8	$3.84 \cdot 10^2$	41,156
2.7	$2.62 \cdot 10^3$	431,212.6	$3.90 \cdot 10^2$	35,633.6
2.8	$2.98 \cdot 10^3$	478,369.6	$5.10 \cdot 10^2$	44,330.2
2.9	$3.29 \cdot 10^3$	529,608	$4.39 \cdot 10^2$	43,304.2
3.0	$3.49 \cdot 10^3$	583,681	$5.29 \cdot 10^2$	46,483.2
3.1	$3.74 \cdot 10^3$	642,023.6	$7.53 \cdot 10^2$	85,028.6
3.2	$3.93 \cdot 10^3$	701,285.2	$4.55 \cdot 10^2$	36,554.6
3.3	$4.32 \cdot 10^3$	769,337	$6.13 \cdot 10^2$	52,744.2
3.4	$4.82 \cdot 10^3$	838,889.8	$6.45 \cdot 10^2$	49,122.4
3.5	$5.34 \cdot 10^3$	912,721	$6.52 \cdot 10^2$	59,171.6
3.6	$5.92 \cdot 10^3$	990,741.4	$7.61 \cdot 10^2$	73,032.6
3.7	$6.50 \cdot 10^3$	1,073,107.8	$8.83 \cdot 10^2$	65,822.8
3.8	$6.97 \cdot 10^3$	1,159,919.2	$7.61 \cdot 10^2$	51,009.2
3.9	$7.37 \cdot 10^3$	1,251,782.4	$6.68 \cdot 10^2$	64,513
4.0	$7.64 \cdot 10^3$	1,343,401.6	$8.36 \cdot 10^2$	65,017.4

Performance comparison between the benchmarks In Fig. 5.10, we observe that the execution time and memory usage of RINGING are higher than those of OVERSHOOT. This is because the TSA of RINGING of is more complex than that of OVERSHOOT: it has more states and clock variables, and it contains a loop. We also observe that for RINGING, the execution time for the tropical semiring is shorter. This is because staying at the locations with $\top$ minimizes the weight for tropical semiring, and we need less exploration.

5.8 Related work

Monitoring can be formulated in various ways. They are classified according to the following criteria. Table 5.7 shows a comparison of the present study with some related studies.

Table 5.7: Comparison of the problem settings with related studies

	Quantitative?	Online?	Dense time?	Result of which part?
[BFN+18]	No	Yes	Yes	All subsignals (pattern matching)
[BFMU17]	Yes	No	Yes	All subsignals (pattern matching)
[JBGN18]	Yes	Yes	No	The whole signal
[DDG+15]	Yes	Yes	Yes	The whole signal
This chapter	**Yes**	**Yes**	**Yes**	**All subsignals (pattern matching)**

Qualitative vs. quantitative semantics When an alphabet admits subtraction and comparisons, in addition to the qualitative semantics (i. e., true or false), one can define a *quantitative* semantics (e. g., robustness) of a signal with respect to the specification [FP09, DM10, AH15, BFMU17]. *Robust semantics* shows how robustly a signal satisfies (or violates) the given specification. For instance, the specification $v > 70$ is satisfied more robustly by $v = 170$ than by $v = 70.0001$. In the context of CPSs, *robust semantics* for signal temporal logic is used in robustness-guided falsification [Don10, ALFS11]. *Weighted automata* are used for quantitative monitoring in [CHO16, JBG+18a, JBGN18], but the time model was *discrete*.

Offline vs. online Consider monitoring of a signal $\sigma = \sigma_1 \cdot \sigma_2$ over a specification $\mathcal{W}$. In offline monitoring, the monitor returns the result $\mathcal{M}(\sigma, \mathcal{W})$ after obtaining the entire signal σ. In contrast, in online monitoring, the monitor starts returning the result before obtaining the entire signal σ. For example, the monitor may return a partial result $\mathcal{M}(\sigma_1, \mathcal{W})$ for the first part σ_1 before obtaining the second part σ_2.

Discrete vs. dense time In a discrete time setting, timestamps are natural numbers while, in a dense time setting, timestamps are positive (or non-negative) real numbers.

Result of which part? Given a signal σ, we may be interested in the properties of different sets of subsignals of σ. The simplest setting is where we are interested only in the whole signal σ (e. g.,[DDG+15, JBGN18]). Another more comprehensive setting is where we are interested in the property of *each* subsignal of σ; problems in this setting are called *timed pattern matching* [UFAM14, WAH16, BFMU17].

Timed pattern matching Among these related studies, the line of works on timed pattern matching are closely related. Since the formulation of *qualitative* timed pattern matching [UFAM14], many algorithms have been presented [UFAM14, UFAM16, WAH16, WHS17, BFN+18], including the online algorithms [WHS17, BFN+18] using timed automata. In the consequence, two tools have been presented [Ulu17, WHS18]. *Quantitative* timed pattern matching was formulated and solved by an offline algorithm in [BFMU17]. This offline algorithm is based on the syntax trees of signal regular expressions, and it is difficult to extend for online monitoring.

Online quantitative monitoring The online quantitative monitoring for signal temporal logic [DDG+15] is also a highly related work. Since we use the clock variables of TSAs to represent the intervals of timed pattern matching, it seems hard to use the algorithm in [DDG+15] for quantitative timed pattern matching.

5.9 Conclusion and perspectives

5.9.1 Conclusion

Using an automata-based approach, we proposed an online algorithm for quantitative timed pattern matching. The key idea of this approach is the reduction to the shortest distance of a weighted graph using zones. Moreover, we utilized an algebraic structure called semiring in our definition of the semantics and the algorithm. Our algorithm works for various semantics, e. g., the semantics defined by the sup-inf semiring and the tropical semiring in Example 5.8.

5.9.2 Perspectives

Comparison of the expressiveness of TSWAs with other formalisms e. g., *signal temporal logic* [MN04] or *signal regular expressions* [BFMU17] is future work. Another future work is the comparison with the quantitative semantics based on the distance between traces presented in [JBGN18].

CHAPTER 6

Model-Bounded Monitoring of Hybrid Systems

In this chapter, we introduce the *model-bounded monitoring* problem of hybrid system using *linear hybrid automata* (*LHAs*). This chapter is based on the unpublished joint work [WAH20] with Étienne André and Ichiro Hasuo. Useful comments from the anonymous referees are gratefully acknowledged.

Organization of the chapter Section 6.1 summarizes the technical contribution in this chapter. We recall LHAs in Section 6.2. After we introduce monitored languages $\mathcal{L}_{\mathrm{mon}}$ for LHAs in Section 6.3, model-bounded monitoring is formalized in Section 6.4, and we prove its correctness. We show that $\mathcal{L}_{\mathrm{mon}}$ membership is undecidable in Section 6.5. We present two partial algorithms: *i*) the one in Section 6.6 relies on an existing model checker PHAVerLite via suitable translation; and *ii*) the one in Section 6.7 is a dedicated algorithm. We perform extensive experiments in Section 6.8. After reviewing the related work in Section 6.9, we conclude in Section 6.10.

6.1 Summary

Here, we summarize our contribution from the technical viewpoint. See Section 1.6.4 for a summary from more application viewpoint.

Hybrid system monitoring In this chapter, we study monitoring of CPSs, with a particular emphasis on their *hybrid* aspect (i. e., the interplay between continuous and discrete worlds).

We sketch the workflow of hybrid system monitoring in Fig. 6.1. We are given a specific behavior σ of the system under monitoring (SUM), and a specification φ (in this paper we focus on safety specifications). The problem is to decide whether σ is safe or not, in the sense that σ satisfies φ. We assume this problem is solved by a computer. Therefore, an input to a monitor must be a discrete-time signal w, obtained from the continuous-time signal σ via sampling. We shall call such w a *log* of the SUM induced by the behavior σ.

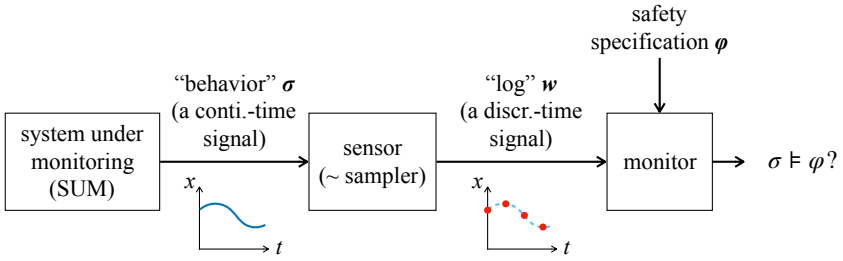
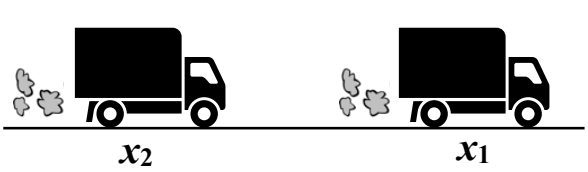
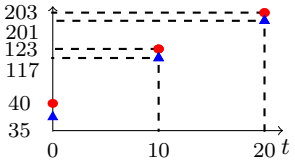


Figure 6.1: Hybrid system monitoring and sampling uncertainties



(a) Automotive platooning



(b) The log w. The red circles are x_1 and the blue triangles are x_2.

Figure 6.3: A leading example: automotive platooning

Sampling uncertainties in hybrid system monitoring There is a methodological difficulty already in the high-level schematics in Fig. 6.1:

> By looking only at a sampled log w, how can a monitor say anything decisive about the real behavior σ?

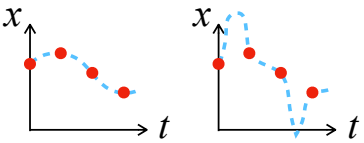
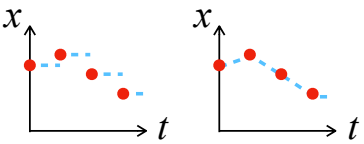


Figure 6.2: w and σ

The same log w can result from different behaviors σ. An example is shown in Fig. 6.2, where we cannot decide if a safety property "x is always nonnegative" is satisfied by σ. In other words, the way we interpolate the log w and recover σ is totally arbitrary. Therefore, we cannot exclude potential violations of any safety specification, unless the specification happens to talk only about values at sampling instants.

This issue of *sampling uncertainties* is often ignored in the hybrid system monitoring literature. They typically employ heuristic interpolation methods, such as piecewise-constant and piecewise-linear interpolation (above). Use of these heuristic interpolation methods is often justified, typically when the sampling rate is large enough. However, in *networked monitoring* scenarios where a sensor and a monitor are separated by, e. g., a wireless network, the sampling rate is small, and the interpolation of a log becomes a real issue. Network monitoring is increasingly common in IoT applications, and smaller sampling rates (i. e., longer sampling intervals) are preferred for energy efficiency.



Example 6.1 (automotive platooning)**.** Consider a situation where two vehicles drive one after the other, with their distance kept small. Such *automotive platooning* attracts interest as a measure for enhanced road capacity as well as for fuel efficiency (by reducing air resistance).

Assume that the monitoring is conducted on a remote server. Each vehicle intermittently sends its position to the server via the Internet. Thus, only a coarse-grained log is available to the remote monitor. Concretely, a log w is given in Fig. 6.3b, by the position x_1, x_2 (meters) of each of the two vehicles, sampled at time $t = 0, 10, 20$ (seconds).

Let us now ask this question: *have the two vehicles touched each other*? Physical contact of the vehicles is not observed in Fig. 6.3b, but we cannot be sure what happened between the

sampling instants. The piecewise-constant and piecewise-linear interpolation can only answer to this question approximately. Moreover, such approximation is not of much help in the current example where sampling intervals are long. ◇

Interpolation assisted by system knowledge The following idea underpins the current work.

> Prior knowledge about a system is a powerful tool to bound sampling uncertainties.

The latter means excluding some candidates when we recover a behavior σ from a word w by interpolation (cf. Fig. 6.2). For the log in Fig. 6.3b, for example, we can say x_1 never reached 10^4, knowing that the vehicle cannot accelerate that quickly.

Putting this idea to actual use requires a careful choice of a knowledge representation formalism.

- For one, it is desired to be *expressive*. The above "acceleration rate" argument can be formulated in terms of Lipschitz constants, but it is nice to also include mode switching—an important feature of hybrid systems.
- For another, a formalism should be *computationally tractable*. Monitoring is a practice-oriented method that often tries to process a large amount of data with limited computing resources (especially in embedded applications). Therefore, inference over knowledge represented in the chosen formalism should better be efficient.

Note that these two concerns—expressivity and computational tractability—are in a trade-off.

Bounding models given by LHAs In this chapter, we express such prior knowledge about a system using a *linear hybrid automaton (LHA)* [HPR94]. This LHA is called a *bounding model*, and serves as an overapproximation of the target system.

LHA is one of the well-known subclasses of hybrid automata (HA); an example is in Fig. 6.6a. LHA's notable simplifying feature is that flow dynamics is restricted to a conjunction of linear (in)equalities over the derivatives $\dot{x}_1, \dot{x}_2, \ldots, \dot{x}_M$. Its expressivity is limited—for example, a flow specification $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{b}$ is not allowed since the variables $\mathbf{x}$ occur there. Differential inclusions are allowed, nevertheless (such as $\dot{x}_1 \in [7.5, 8.5]$ and $\dot{x}_1 - \dot{x}_2 \leq 1$); these are useful in expressing known safety envelopes, as in Fig. 6.6a. Most importantly, analysis of LHAs is tractable, with convex polyhedra providing an efficient means to study the reachability problem.

Model-bounded hybrid system monitoring Our proposal is a scheme that we call *model-bounded monitoring* of hybrid systems. Its workflow is in Fig. 6.4; its features are as follows.

1. We use our prior knowledge about the SUM in order to reduce sampling uncertainties. The knowledge is expressed by an LHA; it is called a *bounding model*.
2. We restrict to a safety specification φ given by a conjunction of linear (in)equalities.[1] We interpret φ globally ("$\sigma(t)$ satisfies φ at any time t"). We combine φ with the bounding model $\mathcal{M}$, obtaining an LHA $\mathcal{M}_{\neg\varphi}$.

[1] This restriction is for the ease of presentation. Extension to LTL specifications should not be hard: an LTL formula can be translated to an automaton; and it can then be combined with a bounding model $\mathcal{M}$. This is future work.

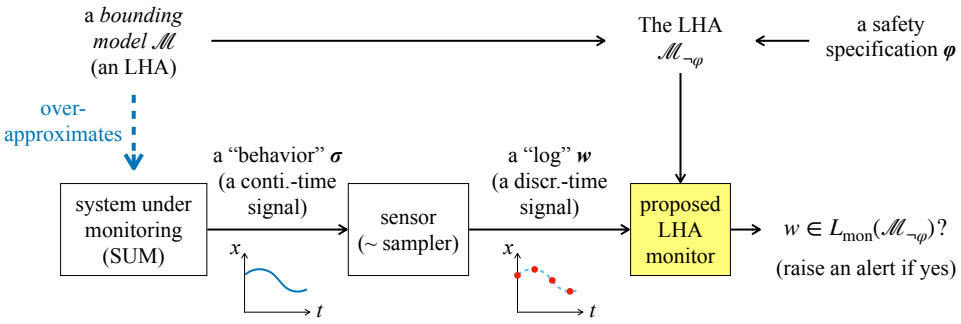


Figure 6.4: Model-bounded monitoring of hybrid systems

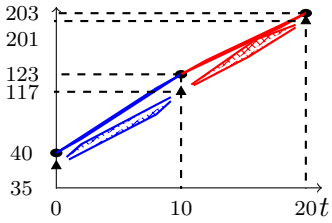
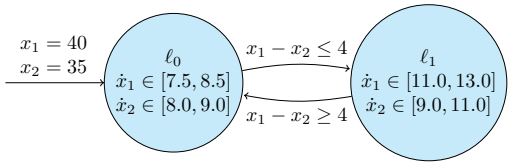
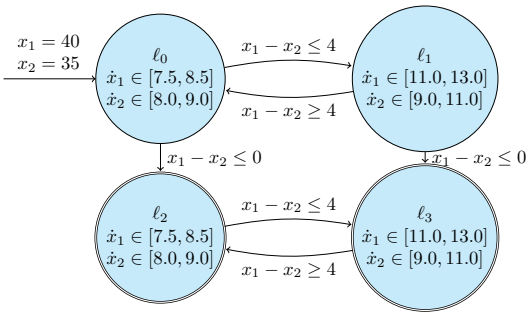


Figure 6.5: Model-bounded monitoring of the log w in Fig. 6.3b. The bounding model $\mathcal{A}$ in Fig. 6.6a confines interpolation to the hatched area. Thus, no collision in $t \in [0, 10]$; potential collision in $t \in [10, 20]$.



(a) A bounding model $\mathcal{A}$ for the platooning example, expressed as an LHA



(b) The LHA $\mathcal{A}_{\neg\varphi}$ for $\varphi = (x_1 - x_2 > 0)$

Figure 6.6: LHAs for the automotive platooning example

3. We introduce the notion of *monitored language* $\mathcal{L}_{\text{mon}}$ of an LHA. Roughly speaking, it is the set of "logs which have a corresponding signal accepted by the LHA." The notion differs from known language notions for LHA (e. g., in [AKV98]), in that mode switches in an LHA need not be visible in a log (modes may change between sampling instants).

4. We show the following meta-level correctness result: $w \in \mathcal{L}_{\text{mon}}(\mathcal{M}_{\neg\varphi})$ if and only if there exists a continuous-time signal σ such that

   a) σ induces w by sampling,

   b) σ conforms with the bounding model $\mathcal{M}$, and

   c) σ violates the safety specification φ.

Our main technical contribution consists of *i*) the introduction of the new language notion $\mathcal{L}_{\text{mon}}$, *ii*) the use of $\mathcal{L}_{\text{mon}}$ in the proposed model-bounded monitoring scheme, and *iii*) (partial) algorithms that solve $\mathcal{L}_{\text{mon}}$ membership. Used in the scheme in Fig. 6.4, these algorithms check if the given log w belongs to $\mathcal{L}_{\text{mon}}(\mathcal{M}_{\neg\varphi})$, whose answer is then used for the safety

analysis of the (unknown) actual behavior σ. The last point is discussed in the next paragraph about usage scenarios.

We present two (partial) algorithms: one reduces the $\mathcal{L}_{\mathrm{mon}}$ membership problem to the reachability problem of LHAs, translating a log w into an LHA. The other is a direct algorithm that relies on polyhedra computation. These algorithms are necessarily partial since $\mathcal{L}_{\mathrm{mon}}$ membership is undecidable (Theorem 6.17). However, their positive and negative answers are guaranteed to be correct. Moreover, we observe that the latter direct algorithm terminates in most benchmarks, especially when a bounding model's dimensionality is not too large.

Example 6.2. We continue Example 6.1. For the log w in Fig. 6.3b, the bounding model $\mathcal{A}$ in Fig. 6.6a confines potential interpolation between the samples to the hatched areas in Fig. 6.5. The two areas are separate in $t \in [0, 10]$, which means the two cars were safe in the period. For $t \in [10, 20]$, the two areas overlap, suggesting potential collision.

The above analysis is automated by our automata-theoretic framework in Fig. 6.4. We shall sketch its workflow. Let φ be the safety specification $x_1 - x_2 > 0$ ("no physical contact"). The formal construction of $\mathcal{A}_{\neg\varphi}$ (Definition 6.12) yields the LHA in Fig. 6.6b. In $\mathcal{A}_{\neg\varphi}$, the original LHA $\mathcal{A}$ (Fig. 6.6a) is duplicated, and once φ is violated, the execution can move from the first copy (the top two states in Fig. 6.6b) to the second (the bottom states). The bottom states are accepting—they detect violation of φ.

Now we use one of our algorithms to solve the membership problem, i. e., if the log w belongs to $\mathcal{L}_{\mathrm{mon}}(\mathcal{A}_{\neg\varphi})$. Solving this membership problem amounts to computing the hatched areas in Fig. 6.5—it is done relying on polyhedra computation—and checking if the safety specification is violated.

◇

Usage scenarios The scheme in Fig. 6.4 is used as follows. As a basic prerequisite, we assume that the bounding model $\mathcal{M}$ *overapproximates* the SUM: for each continuous-time signal σ,

> **(soundness of a bounded model)**
>
> σ is a behavior of the SUM $\Longrightarrow$ σ is a run of $\mathcal{M}$.

We do not require the other implication. Due to the limited expressivity of LHAs (that is the price for computational tractability), $\mathcal{M}$ would not tightly describe the SUM.

Assume first that our monitor did *not* raise an alert (i. e., $w \notin \mathcal{L}_{\mathrm{mon}}(\mathcal{A}_{\neg\varphi})$). Let σ_0 be the (unknown) actual behavior of the SUM that is behind the log w. By the feature 4 of the scheme, we conclude that σ_0 was safe. Indeed, σ_0 satisfies (a) by definition. It comes from the SUM, and thus by the soundness assumption, σ_0 satisfies (b). Hence, (c) must fail.

Let us turn to the case where our monitor *did* raise an alert ($w \in \mathcal{L}_{\mathrm{mon}}(\mathcal{M}_{\neg\varphi})$). This can be a false alarm. For one, the existence of unsafe σ (as in the feature 4) does not imply that the actual behavior σ_0 was unsafe. For another, (b) does not guarantee that σ is indeed a possible behavior of the SUM, since we only assume *soundness* of the bounding model. Nevertheless, a positive answer of our monitor comes with a reachability witness (a trace) in $\mathcal{M}_{\neg\varphi}$, which serves as a useful clue for further examination.

Summarizing, our monitor's alert can be false, while the absence of an alert proves safety. We can thus say our model-bounded monitoring scheme is *sound*.

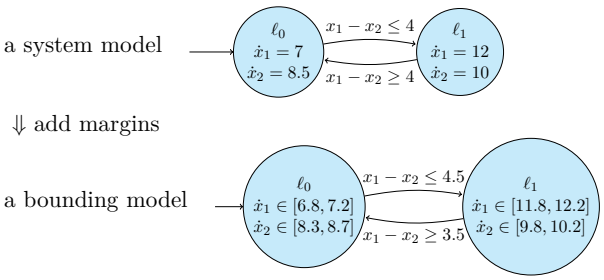


Figure 6.7: *Adding margins* to obtain bounding models. The top model gets loosened by perception uncertainties (margin 0.5) and actuation uncertainties (margin 0.2)

Bounding models We note that the roles of bounding models are different from common roles played by *system models*. A system model aims to describe the system's behaviors in a *sound* and *complete* manner. In contrast, bounding models focus on overapproximation, trading completeness for computational tractability that is needed in monitoring applications.

The *overapproximating* nature of a bounding model is less of a problem in monitoring, compared to other exhaustive applications such as model checking. In the latter, approximation errors accumulate over time, leading to increasingly loose overapproximation. In contrast, in our usage, a bounding model is used to interpolate between samples (Fig. 6.5). Here overapproximation errors get reset to zero by new samples.

Bounding models can arise in different ways, including:

- **(Adding margins to a system model)** If a system model is given as an LHA, we can use it as a bounding model. A more realistic scenario is to add some margins to address potential perception and actuation errors. LHAs' feature that they allow differential *inclusions* is particularly useful here. An example is in Fig. 6.7, where perception and actuation uncertainties are addressed by the additional margins in the transition guards and flow dynamics, respectively.

- **(LHA approximation of a system model)** LHA is one of the subclasses of HA for which exact reachability is attackable (it is hopeless for general HA). Consequently, tools have been proposed for analyzing LHA, including PHAVerLite [BZ19] and its predecessor PHAVer [Fre08]. Moreover, for their application, overapproximation of other dynamics by LHAs has been studied and tool-supported. See e. g., [Fre08, Section 3.2]. These techniques can be used to obtain an LHA bounding model from a more complex model.

- **(From a third-party vendor)** HA is a well-accepted formalism in academia and industry. It is conceivable that a system vendor provides an LHA as the system's "safety specification." It serves as a bounding model.

Contributions We summarize our main contributions.

- We tackle the issue of sampling uncertainties in hybrid system monitoring, proposing the *model-bounded monitoring* scheme (Fig. 6.4) as a countermeasure. The scheme uses LHAs as *bounding models*.

- We introduce the novel technical notion of *monitored language* $\mathcal{L}_{\text{mon}}$ for LHAs. In $\mathcal{L}_{\text{mon}}$, unlike in other language notions, input words and mode switches do not necessarily

synchronize. We show that $\mathcal{L}_{\text{mon}}$ membership is undecidable, yet we introduce two partial algorithms.

- We establish soundness of our model-bounded monitoring scheme: absence of an alert guarantees that every possible behavior σ behind the log w is safe.
- The practical relevance and algorithmic scalability is demonstrated by experiments, using benchmarks that are taken from automotive platooning scenarios.

Our focus is in the algorithmic aspects of the scheme in Fig. 6.4. The rate of false alerts is a major issue that this paper does not address—this issue is about the quality of an input to the algorithm ("how tight a bounding model is") rather than about the algorithm itself. Further discussions are in Remark 6.20.

In this chapter, we assume that each sample in the log w contains the complete valuation i. e., we know the value of each variable x at each sampling. We note that it is easy to relax this assumption and allow partial valuations i. e., we do not know the value of all the variables at each sampling. See Appendix D for the detail of this generalization. By using partial valuation, we can also encode parameters in the specification or the model.

6.2 Preliminaries: Linear hybrid automata

Let $\mathcal{I}(\mathbb{Q})$ be the set of closed intervals on $\mathbb{Q}$, i. e., of the form $[a, b]$, where $a, b \in \mathbb{Q}$ and $a \leq b$. For a partial function $f\colon X \nrightarrow Y$, the domain $\{x \in X \mid f(x) \text{ is defined}\}$ is denoted by $\mathrm{dom}(f)$. We fix a set $\mathbb{X} = \{x_1, \dots, x_M\}$ of real-valued *variables*. A *(variable) valuation* is a function $v : \mathbb{X} \to \mathbb{R}$. When $\mathbb{X}$ is clear from the context, a valuation v is expressed by the tuple $(v(x_1), v(x_2), \dots, v(x_M))$. Given $\mu\colon \mathbb{X} \nrightarrow \mathcal{I}(\mathbb{Q})$, we define the *update* of a valuation v, written $[v]_\mu$, as follows: $[v]_\mu(x) \in \mu(x)$ if $\mu(x)$ is defined, and $[v]_\mu(x) = v(x)$ otherwise.

We assume $\bowtie \in \{\leq, =, \geq\}$. Let $\Phi(\mathbf{x})$ be the set of *linear systems* over $\mathbb{X}$ defined by a finite conjunction of inequalities of the form $a_1x_1 + a_2x_2 + \cdots + a_Mx_M \bowtie d$, with $d, a_1, a_2, \dots, a_M \in \mathbb{Z}$. We let $\top = \bigwedge \emptyset$ and $\bot$ be the contradiction. The set $\Phi(\dot{\mathbf{x}})$ is defined similarly; it consists of constraints over derivatives $\dot{x}_1, \dots, \dot{x}_M$.

6.2.1 Syntax

Definition 6.3 (linear hybrid automata (LHA) [HPR94])**.** An *LHA* is a tuple

$$\mathcal{A} = (L, L_F, \mathbb{X}, \mathrm{Init}, \mathcal{F}, \mathrm{Inv}, E) \ ,$$

where:

1. L is a finite set of locations,
2. $L_F \subseteq L$ is the set of accepting locations,
3. $\mathbb{X}$ is a finite set of variables,
4. $\mathrm{Init}\colon L \to \Phi(\mathbf{x})$ is the initial variable valuation for each location,
5. $\mathcal{F}\colon L \to \Phi(\dot{\mathbf{x}})$ is the *flow*, assigning to every $\ell \in L$ the set of derivatives ("rates") $\{(\dot{x}_1, \dot{x}_2, \dots, \dot{x}_M) \mid (\dot{x}_1, \dot{x}_2, \dots, \dot{x}_M) \models \mathcal{F}(\ell)\}$,

6. $\mathrm{Inv}\colon L \to \Phi(\mathbf{x})$ assigns to $\ell \in L$ an *invariant* $\mathrm{Inv}(\ell) \in \Phi(\mathbf{x})$,
7. E is a finite set of *edges* $e = (\ell, g, \mu, \ell')$ where *i*) $\ell, \ell' \in L$ are the source and target locations, *ii*) $g \in \Phi(\mathbf{x})$ is the guard, *iii*) $\mu\colon \mathbb{X} \nrightarrow \mathcal{I}(\mathbb{Q})$ is the update function.

◇

Note that Definition 6.3 allows for non-deterministic initial locations. A location ℓ that cannot be initial is such that $\mathrm{Init}(\ell) = \bot$.

Example 6.4. Consider the LHA in Fig. 6.6a, where Init is such that $\mathrm{Init}(\ell_0) = \{x_1 = 40 \wedge x_2 = 35\}$ and $\mathrm{Init}(\ell_1) = \bot$. This LHA, giving a bounding model for an automotive platooning system (Example 6.1), contains 2 locations and 2 variables $\mathbb{X} = \{x_1, x_2\}$. This LHA features no invariant (i. e., all invariants are $\top$). Note that this LHA fits into a subclass in which the derivatives for the flows are all in bounded, constant intervals.

In this LHA (Fig. 6.6a), x_1 (resp. x_2) denotes the position of Vehicle 1 (resp. 2), initially 40 and 35 respectively. In ℓ_0, both vehicles run roughly at the same speed, although Vehicle 2 can be slightly faster (e. g., due to smaller air resistance, as it follows Vehicle 1). When the distance between both vehicles becomes less than 4, they enter mode ℓ_1, where Vehicle 1 drives faster than in ℓ_0.

In the LHA $\mathcal{A}_{\neg\varphi}$ in Fig. 6.6b, the vertical edges are enabled once the specification $x_1 - x_2 > 0$ is violated, that is, once the two vehicles touch each other. ◇

A *timed automaton* (TA) [AD94] is an LHA *i*) each variable is a *clock*, i. e., its derivative is 1 in all locations, *ii*) each update μ attached to edges is of the form $\mu\colon \mathbb{X} \nrightarrow \{0\}$, and *iii*) the initial state is deterministic, i. e., a single location is initial and all values are initially equal to a constant integer-value.[2]

Example 6.5. Consider the LHA in Fig. 6.8, where *i*) $\mathbb{X} = \{x_1, x_2, t_{abs}, t_{rel}\}$, *ii*) Init is such that $\mathrm{Init}(\ell_0) = \{x_1 = 40 \wedge x_2 = 35 \wedge t_{abs} = 0 \wedge t_{rel} = 0\}$ and $\mathrm{Init}(w_i) = \bot$ for $1 \leq i \leq 3$, and *iii*) $\dot{x}_1 = \dot{x}_2 = \dot{t}_{abs} = \dot{t}_{rel} = 1$ in all locations (not depicted in Fig. 6.8). Then this LHA is a TA. Note that we use the TA notation for invariants, i. e., boxed under the location. ◇

6.2.2 Semantics

We recall the standard semantics of LHAs called concrete semantics. It is formulated as a timed transition system [HMP91].

Definition 6.6 (concrete semantics of an LHA)**.** Given an LHA $\mathcal{A} = (L, L_F, \mathbb{X}, \mathrm{Init}, \mathcal{F}, \mathrm{Inv}, E)$, the *concrete semantics* of $\mathcal{A}$ is given by the timed transition system (TTS) $(S, S_0, \to)$, with

- $S = \{(\ell, v) \in L \times \mathbb{R}^M \mid v \models \mathrm{Inv}(\ell)\}$,
- $S_0 = \{(\ell, v) \mid \ell \in L, v \in \mathrm{Init}(\ell)\} \cap S$,
- $\to$ consists of the discrete and continuous transition relations:
  1. discrete transitions: $(\ell, v) \overset{e}{\mapsto} (\ell', v')$, if there exists $e = (\ell, g, \mu, \ell') \in E$ such that $v \models g$, $v' \in [v]_\mu$.

[2] Strictly speaking, the original definition [AD94] and the definition in Section 2.2 require clock to be zero, and they do not have invariants, but this is equivalent.

2. continuous transitions: $(\ell, v) \overset{d,f}{\mapsto} (\ell, v')$, with the delay $d \in \mathbb{R}_{\geq 0}$ and the flow $f \colon \mathbb{X} \to \mathbb{R}$ satisfying, $f \models \mathcal{F}(\ell)$, $\forall d' \in [0, d], (\ell, v + d'f) \in S$, and $v' = v + df$, where $v + d'f$ is the valuation satisfying $(v + d'f)(x) = v(x) + d'f(x)$ for any $x \in \mathbb{X}$.

◇

Definition 6.7 ((accepting) run)**.** Given an LHA $\mathcal{A}$ with concrete semantics $(S, S_0, \to)$, we refer to the states of S as the *concrete states* of $\mathcal{A}$. A *run* of $\mathcal{A}$ is an alternating sequence $\rho = s_0, \to_1, s_1, \to_2, \dots, \to_n, s_n$ of concrete states $s_i \in S$ and transitions $\to_i \in \to$ satisfying $s_0 \in S_0$ and $s_0 \to_1 s_1 \to_2 \cdots \to_n s_n$. For a run ρ, the *duration* $\mathrm{Dur}(\rho) \in \mathbb{R}_{\geq 0}$ is the sum of the delays in ρ. We denote the i-th prefix $s_0 \to_1 s_1 \to_2 \cdots \to_i s_i$ of ρ by $\rho[i]$.

A run is *accepting* if its last state (ℓ, v) is such that $\ell \in L_F$. ◇

Example 6.8. Let $\mathcal{A}$ be the LHA in Fig. 6.6a. The sequence $\rho = (\ell_0, v_0) \overset{10,(8.3,8.2)}{\mapsto} (\ell_0, v_1) \overset{\frac{4}{3},(7.5,9)}{\mapsto} (\ell_0, v_2) \overset{e_1}{\mapsto} (\ell_1, v_2) \overset{\frac{2}{3},(12,9)}{\mapsto} (\ell_1, v_3) \overset{e_2}{\mapsto} (\ell_0, v_3) \overset{8,(7.75,8.25)}{\mapsto} (\ell_0, v_4)$ is a run of $\mathcal{A}$, where $v_0 = (40, 35)$, $v_1 = (123, 117)$, $v_2 = (133, 129)$, $v_3 = (141, 135)$, $v_4 = (203, 201)$, and e_1 and e_2 are the edges from ℓ_0 and ℓ_1, respectively. ◇

6.3 Monitored languages of LHAs

We introduce another semantics of an LHA besides concrete semantics (Definition 6.6); it is called the *monitored language*. The two semantics are used in Fig. 6.4 in the following way: *i*) concrete semantics is (roughly) about whether a continuous-time signal σ ("behavior") conforms with the LHA $\mathcal{A}$; *ii*) the monitored language $\mathcal{L}_{\mathrm{mon}}(\mathcal{A})$ is about whether a discrete-time signal w ("log") conforms with $\mathcal{A}$.

Definition 6.9 (timed quantitative words)**.** A *timed quantitative word* w is a sequence

$$(u_1, \tau_1), (u_2, \tau_2), \dots, (u_m, \tau_m)$$

of pairs (u_i, τ_i) of a valuation $u_i \colon \mathbb{X} \to \mathbb{R}$ and a timestamp $\tau_i \in \mathbb{R}_{\geq 0}$ satisfying $\tau_i \leq \tau_{i+1}$ for each $i \in \{1, 2, \dots, m-1\}$.

For a timed quantitative word $w = (u_1, \tau_1), (u_2, \tau_2), \dots, (u_m, \tau_m)$, we let $|w| = m$ and for any $i \in \{1, 2, \dots, n\}$, we let $w[i] = (u_1, \tau_1), (u_2, \tau_2), \dots, (u_i, \tau_i)$. ◇

We sometimes refer to pairs (u_i, τ_i) as *samples*—these are the red dots in Fig. 6.4.

Definition 6.10 (monitored language $\mathcal{L}_{\mathrm{mon}}(\mathcal{A})$)**.** Let $\rho = s_0 \to_1 s_1 \to_2 \cdots \to_n s_n$ be a run of an LHA $\mathcal{A}$ (Definition 6.6), and $w = (u_1, \tau_1), (u_2, \tau_2), \dots, (u_m, \tau_m)$ be a timed quantitative word. We say w is *associated* to ρ if, for each $j \in \{1, 2, \dots, m\}$, we have either of the following two. Here ℓ_i, v_i are so that $s_i = (\ell_i, v_i)$ for each $i \in \{0, 1, \dots, n\}$.

1. There exists $i \in \{0, 1, 2, \dots, n\}$ such that $\mathrm{Dur}(\rho[i]) = \tau_j$ and $u_j = v_i$; or
2. There exists $i \in \{0, 1, 2, \dots, n-1\}$ such that $\mathrm{Dur}(\rho[i]) < \tau_j < \mathrm{Dur}(\rho[i+1])$ and for any $x \in \mathbb{X}$, $u_j(x) = v_i(x) + (\tau_j - \mathrm{Dur}(\rho[i]))f_i(\dot{x})$ holds, where $\to_i = \overset{d_i,f_i}{\mapsto}$.

Finally, the *monitored language* $\mathcal{L}_{\mathrm{mon}}(\mathcal{A})$ of an LHA $\mathcal{A}$ is the set of timed quantitative words associated with some accepting run of $\mathcal{A}$. ◇

In the above definition of association of w to ρ, note that the lengths of ρ and w can differ ($n \neq m$). The condition 1 is when a sample in w happens to be simultaneous with some transition in ρ. This special case is not required to happen at all, for w to be associated to ρ.

For example, in Fig. 6.5, mode switches (i. e., discrete transitions) in the LHA in Fig. 6.6a can occur at times other than $t = 0, 10, 20$. This is in contrast to the language of hybrid automata in [AKV98], where (observable) discrete transitions are always *synchronous* with the word, much like the condition 1.

Example 6.11. Let $\mathcal{A}$ be the LHA in Fig. 6.6a, ρ be the run of $\mathcal{A}$ in Example 6.8. The timed quantitative word w in Fig. 6.3b is associated to ρ. We note that the sampling and the discrete transitions are *asynchronous*: the sampling is at 0, 10, and 20, while the discrete transitions are at $\frac{34}{3}$ and 12. This is in contrast to the *synchronous* language in [AKV98]: the accepted words represent the discrete transitions e. g., at $\frac{34}{3}$ and 12. ◇

6.4 The model-bounded monitoring scheme

Based on the technical definitions in Section 6.3, we formally introduce the scheme that we sketched in Fig. 6.4. (Partial) algorithms for computing if $w \in \mathcal{L}_{\mathrm{mon}}(\mathcal{M}_{\neg\varphi})$ are introduced in later sections.

Recall that we focus on safety specifications that are global and linear.

Definition 6.12 (the LHA $\mathcal{A}_{\neg\varphi}$)**.** Let $\mathcal{A}$ be an LHA, and $\varphi \in \Phi(\mathbf{x})$ (Section 6.2). The LHA $\mathcal{A}_{\neg\varphi}$ is defined by

- making a copy $\mathcal{A}^\circ$ of $\mathcal{A}$,
- making every location ℓ° of $\mathcal{A}^\circ$ non-initial (by letting $\mathit{Immd}(\ell^\circ) = \bot$),
- letting L_F consist of all the states ℓ° of $\mathcal{A}^\circ$, and
- for each location $\ell \in \mathcal{A}$, creating an edge $(\ell, \neg\varphi, \emptyset, \ell^\circ)$ from ℓ to its copy ℓ°, labeling the edge with the safety specification φ as a guard and no update.

◇

An example of $\mathcal{A}_{\neg\varphi}$ is shown in Fig. 6.6. We note that, in $\mathcal{A}_{\neg\varphi}$, having a single accepting sink state for violation of φ is not enough. After detecting violation, we are still obliged to check if the rest of a word w conforms with the bounding model $\mathcal{A}$. Therefore we maintain a copy of $\mathcal{A}$.

Lemma 6.13. *The following are equivalent, for each sequence ρ.*

1. *Both of the following hold: i) ρ is a (non-necessarily accepting) run of $\mathcal{A}$, and ii) ρ violates φ at a certain time instant.*
2. *There exists an accepting run ρ' of $\mathcal{A}_{\neg\varphi}$ such that*
   $\{w \mid w \text{ is associated to } \rho\}$
   $= \{w \mid w \text{ is associated to } \rho'\}$. □

The proof is easy by definition. The runs ρ and ρ' can differ only in the locations they visit—in an LHA, an enabled transition is not always taken. Note, however, that violation of φ and w's association are two properties that are insensitive to locations.

We are ready to state the correctness of our scheme (Fig. 6.4). The proof is straightforward by Lemma 6.13 and Definition 6.10.

Theorem 6.14 (correctness)**.** *In the setting of Definition 6.12, let w be a timed quantitative word. We have $w \in \mathcal{L}_{\mathrm{mon}}(\mathcal{A}_{\neg\varphi})$ if and only if there is a (non-necessarily accepting) run ρ of $\mathcal{A}$ such that i) w is associated to ρ, and ii) ρ violates φ at some time instant.* □

Identifying a run ρ with a behavior σ, and association of w to ρ with sampling, the theorem establishes the feature 4 of our scheme (Section 6.1).

The consequence in the safety analysis of the real SUM (instead of its bounding model $\mathcal{A}$) is discussed in the "usage scenario" paragraph of Section 6.1. In particular, due to potential gaps between the SUM and the bounding model $\mathcal{A}$, an alert of our monitor can be false, while the absence of an alert proves safety. Overall, *our model-bounded monitoring scheme is sound.*

Example 6.15. We show how the illustration in Example 6.2 is formalized by the monitored language $\mathcal{L}_{\mathrm{mon}}(\mathcal{A}_{\neg\varphi})$ of the bounded model $\mathcal{A}_{\neg\varphi}$. Let $\mathcal{A}_{\neg\varphi}$ be the LHA in Fig. 6.6b and w be the timed quantitative word in Fig. 6.3b.

We have $w[2] \notin \mathcal{L}_{\mathrm{mon}}(\mathcal{A}_{\neg\varphi})$ because all the runs to which $w[2]$ is associated are not accepting. That is, the log w is safe until time $t = 10$.

However, for the full log we have $w \in \mathcal{L}_{\mathrm{mon}}(\mathcal{A}_{\neg\varphi})$, because w is associated to the following accepting run ρ: $\rho = \Big[(\ell_0, u_1) \overset{10,(8.3,8.2)}{\mapsto} (\ell_0, u_2) \overset{4,(7.5,9.0)}{\mapsto} (\ell_0, v) \overset{e}{\mapsto} (\ell_2, v) \overset{6,\left(\frac{25}{3},8.0\right)}{\mapsto} (\ell_2, u_2)\Big]$, where $v = (153, 153)$. ◇

6.5 Membership for monitored languages: symbolic interpolation

The rest of the chapter is devoted to solving the membership problem of $\mathcal{L}_{\mathrm{mon}}(\mathcal{A})$, a core computation task in Fig. 6.4. We will present two (partial) algorithms: they are symbolic algorithms that iteratively update polyhedra.

The $\mathcal{L}_{\mathrm{mon}}$ membership problem:
INPUT: An LHA $\mathcal{A}$ and a timed quantitative word w.
PROBLEM: Return the set $\mathsf{C}(w, \mathcal{A})$ of indices i satisfying $u[i] \in \mathcal{L}_{\mathrm{mon}}(\mathcal{A})$. In particular, $w \in \mathcal{L}_{\mathrm{mon}}(\mathcal{A})$ iff $|w| \in \mathsf{C}(w, \mathcal{A})$.

Example 6.16. Let $\mathcal{A}_{\neg\varphi}$ be the LHA in Fig. 6.6b and w be the timed quantitative word in Fig. 6.3b. We have $\mathsf{C}(w, \mathcal{A}_{\neg\varphi}) = \{3\}$, meaning $w[1], w[2] \notin \mathcal{L}_{\mathrm{mon}}(\mathcal{A}_{\neg\varphi})$, and $w[3] = w \in \mathcal{L}_{\mathrm{mon}}(\mathcal{A}_{\neg\varphi})$. This result corresponds to the illustration in Fig. 6.5. ◇

The following "no-go" theorem is sort of expected, given previous results from the hybrid automata literature.

Theorem 6.17 (undecidability)**.** *For an LHA $\mathcal{A}$ and a timed quantitative word w, it is undecidable to determine the emptiness of $\mathsf{C}(w, \mathcal{A})$.*

Proof. We prove the claim by a reduction from the bounded-time reachability of the LHA $\mathcal{A} = (L, L_F, \mathbb{X}, \mathrm{Init}, \mathcal{F}, \mathrm{Inv}, E)$. Let $\tau \in \mathbb{R}_{>0}$ be the time bound of the reachability checking. Let $u\colon \mathbb{X} \to \mathbb{R}$ be the valuation satisfying $u(x) = 0$ for any $x \in \mathbb{X}$, $w = (u, \tau)$, and $\mathcal{A}' = (L \amalg \{\ell_f\}, \{\ell_f\}, \mathbb{X}, \mathrm{Init}, \mathcal{F}', \mathrm{Inv}', E \amalg E')$, where:

- $\mathcal{F}'$ is $\mathcal{F}'(\ell) = \mathcal{F}(\ell)$ for $\ell \in L$ and $\mathcal{F}'(\ell_f)$ is such that $\dot{x} = 0$ for any $x \in \mathbb{X}$;
- Inv' is $\mathrm{Inv}'(\ell) = \mathrm{Inv}(\ell)$ for $\ell \in L$ and $\mathrm{Inv}'(\ell_f) = \top$; and
- $E' = \{(\ell, \top, \mu, \ell_f) \mid \ell \in L_F, \mu(x) = 0 \text{ for any } x \in \mathbb{X}\}$.

We have $(u, \tau) \in \mathcal{L}_{\mathrm{mon}}(\mathcal{A})$ if and only if L_F is reachable in $\mathcal{A}$ within τ. Since we have $|w| = 1$, the bounded-time reachability of the LHA $\mathcal{A}$, which is undecidable [BDG+11], is reduced to the emptiness checking of $\mathsf{C}(w, \mathcal{A}')$. □

On the one hand, given the undecidability result, we can think of restricting the class of the models. For example, the problems become decidable if the number of discrete transitions within a time unit is bounded.

On the other hand, in practice, as we observe in Section 6.8, our partial algorithms below perform effectively for many benchmarks—especially our latter, direct algorithm.

Our two partial algorithms have the following features.

- Instead of solving one-way reachability (forward or backward) as many existing algorithms do, they solve *interpolation* between two points (i. e., samples, see Fig. 6.5).
- They work in a *one-shot manner*, collecting the linear constraints for "reachability from the given origin" and those for "reaching the given end" simultaneously. Instead, a naive method would iterate between forward and backward reachability analysis.

6.6 Algorithm I: via reduction to LHA reachability analysis

In our first solution, we reduce the $\mathcal{L}_{\mathrm{mon}}$ membership problem to reachability analysis of LHAs. In practice, we will use PHAVerLite, one of the most efficient tools for reachability analysis of hybrid systems according to [BZ19].

The idea of reducing monitoring to reachability analysis of extensions of finite-state automata is also employed in Section 3.3. Indeed, the reduction in this section is similar to that in Section 3.3. While both in Section 3.3 and the method we introduce in this section are symbolic, the differences are in the formalism and problem. On the one hand, in Section 3.3, we used parametric timed automata as a parametric specification and performs *parametric* timed pattern matching (which can be seen as parametric monitoring). On the other hand, we use LHAs for the bounding model and perform symbolic monitoring. An extension for a parametric setting is a future work, which is technically not much demanding.

Our workflow is as follows:

1. We transform the input timed quantitative word w into an LHA $\mathcal{A}_w$ (that is in fact only timed, i. e., it only uses clocks), that uses two extra clocks:
   a) t_{abs} measures the absolute time since the beginning of the word; and
   b) t_{rel} measures the (relative) time since the last sampled timed quantitative word.

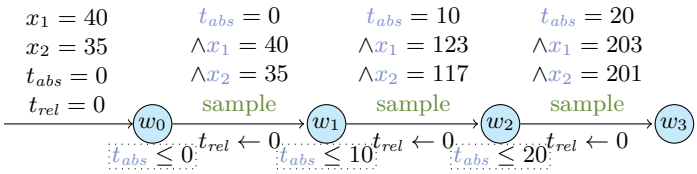


Figure 6.8: TWQ2LHA applied to the timed quantitative word in Fig. 6.3b. Here, *i)* $\mathbb{X} = \{x_1, x_2, t_{abs}, t_{rel}\}$, *ii)* Init is such that $\text{Init}(\ell_0) = \{x_1 = 40 \wedge x_2 = 35 \wedge t_{abs} = 0 \wedge t_{rel} = 0\}$ and $\text{Init}(w_i) = \bot$ for $1 \leq i \leq 3$, and *iii)* $\dot{t}_{abs} = \dot{t}_{rel} = 1$ in all locations. We use the TA notation for invariants, i. e., boxed under the location.

2. We perform the synchronized product $\mathcal{A} \mid\mid \mathcal{A}_w$ of the given LHA $\mathcal{A}$ with the transformed LHA $\mathcal{A}_w$.

3. We run the reachability analysis procedure for the product LHA $\mathcal{A} \mid\mid \mathcal{A}_w$, to derive all possible locations w_i of $\mathcal{A}_w$ such that (ℓ, w_i) is reachable in $\mathcal{A} \mid\mid \mathcal{A}_w$ with $t_{rel} = 0$, where ℓ is an accepting location of the given LHA $\mathcal{A}$.

We explain these steps in the following.

6.6.1 Transforming the timed quantitative word into an LHA

First, we transform the input timed quantitative word w into an LHA, and in fact into the *timed automata* fragment of LHA. The resulting LHA $\mathcal{A}_w$ is a simple sequence of locations with guarded transitions in between, also resetting t_{rel}.

The LHA $\mathcal{A}_w$ features an absolute time clock t_{abs} (initially 0, of rate 1 and never reset), and can test all variables of the system in guards (these are not reset in this LHA though). More in details, we simply convert each sample (u_i, τ_i) of the timed quantitative word w into a guard of the LHA testing for the timestamp using the absolute time clock t_{abs}, and for the value of the variables. The invariant of the location preceding a timestamp τ_i also features the clock constraint $t_{abs} \leq \tau_i$ (this is not crucial for correctness but limits the state space explosion). The transitions are all labeled with a fresh action sample (which could be replaced with an unobservable action, but such actions are not accepted by the PHAVerLite model checker). Each transition resets t_{rel}. Let TWQ2LHA denote this procedure.

For example, consider the timed quantitative word w in Fig. 6.3b. The result $\mathcal{A}_w$ of TWQ2LHA(w) is given in Fig. 6.8.

6.6.2 Reachability analysis using PHAVerLite

We perform the synchronized product $\mathcal{A}_w \mid\mid \mathcal{A}$ ("parallel composition") of the LHA $\mathcal{A}_w$ constructed from w together with the given automaton $\mathcal{A}$. The synchronized product of two LHA is known to be an LHA [HPR94].

Then, we run the reachability analysis, setting as target the states for which both of the following conditions hold:

1. the monitor is in an accepting location; and

2. $t_{rel} = 0$.

The latter condition ensures that only the states such that we just sampled a word are accepting. Thanks to the latter condition, we can take into account of the next sample without any explicit backward reachability analysis.

Algorithm 10: Outline of our incremental procedure for the $\mathcal{L}_{\text{mon}}$ membership problem

Input: A timed quantitative word $w = (u_1, \tau_1), (u_2, \tau_2), \ldots, (u_n, \tau_n)$ and an LHA $\mathcal{A} = (L, L_F, \mathbb{X}, \text{Init}, \mathcal{F}, \text{Inv}, E)$.
Output: The Boolean sequence $\mathit{Result}_1, \ldots, \mathit{Result}_n$, where $\mathit{Result}_i = \top \iff w[i] \in \mathcal{L}_{\text{mon}}(\mathcal{A})$

1 $\mathit{Conf}_0 \leftarrow \{(\ell, v) \mid \ell \in L, v \in \text{Init}(\ell)\}$
2 **for** $i \leftarrow 1$ **to** n **do**
3 $\quad \mathit{Conf}'_i \leftarrow$ reachable states from Conf_{i-1} in duration $\tau_i - \tau_{i-1}$
4 $\quad \mathit{Conf}_i \leftarrow \{(\ell, v) \in \mathit{Conf}'_i \mid v = u_i\}$
5 $\quad \mathit{Result}_i \leftarrow \exists (\ell, v) \in \mathit{Conf}_i.\, \ell \in L_F$

Example 6.18. Let us exemplify the need for the latter condition. Consider the LHA $\mathcal{A}$ in Fig. 6.6b and the timed quantitative word w in Fig. 6.3b transformed into the TA $\mathcal{A}_w$ in Fig. 6.8 (only the time frame in $[0, 10]$ is of interest in this example). Clearly, this log is safe w.r.t. the automaton $\mathcal{A}$ in Fig. 6.6b, that is neither ℓ_2 nor ℓ_3 are reachable, because the distance between both vehicles cannot have been ≤ 0 in the $[0, 10]$ time frame.

Now, if we simply run the reachability procedure looking for ℓ_2 or ℓ_3 as target (without condition on t_{rel}), the procedure will output that at least ℓ_2 is reachable. Indeed, it is possible that vehicle 1 runs at the minimal rate of 7.5 while vehicle 2 runs at the maximal rate of 9. In that case, after 10 time units, vehicle 1 (resp. 2) reaches x-coordinate 115 (resp. 125), and therefore their distance is ≤ 0, making ℓ_2 reachable. While this behavior is indeed possible from the knowledge we have of the first sample, it is actually impossible knowing the full log and in particular the second sample. This phenomenon is illustrated in the part of Fig. 6.5 restricted to the $[0, 10]$ time frame: the blue part depicts all possible valuations knowing the first and second sample. ◇

Hence, adding the condition $t_{rel} = 0$ forces the model checker to take into consideration the next sample before making a decision concerning the reachability of a possible target location.

6.7 Algorithm II: Direct method by polyhedra computation

In our second solution, we directly solve the $\mathcal{L}_{\text{mon}}$ membership problem. We iteratively compute the runs of the LHA $\mathcal{A}$ associated with the prefixes of the timed quantitative word w utilizing bounded reachability analysis. This is our main contribution.

Algorithm 10 shows an outline of our incremental procedure for the $\mathcal{L}_{\text{mon}}$ membership problem. Algorithm 10 incrementally constructs the intermediate states Conf_i and outputs the partial result Result_i showing if $w[i] \in \mathcal{L}_{\text{mon}}(\mathcal{A})$. In line 1, we construct the initial states Conf_0. We note that although $\mathit{Immd}(\ell)$ is in general an infinite set, it is given as a convex polyhedron, and we can represent Conf_0 as a finite list of pairs of a location and a convex polyhedron.

From line 2 to line 5 is the main part of Algorithm 10: we incrementally compute Conf_i and Result_i. In line 3, we compute the reachable states Conf_i from Conf_{i-1} after the executions of duration $\tau_i - \tau_{i-1}$. This part is essentially the same as the bounded-time reachability analysis, and thus, it is undecidable for LHAs in general [BDG+11]. Nevertheless, in practice, the reachable states Conf'_i are usually effectively computable as a finite union of convex polyhedra.

In line 4, we require Conf_i to be the subset of Conf'_i compatible with the current observation u_i. Thanks to this requirement, we can take into account of the next sample just by the forward

Table 6.1: Summary of the benchmarks

Name	Dimension ($= d$)	# of locs.	max. len. of the logs
ACCC	5,10,15	$d+1$	1,000
ACCI	2	4	100,000
ACCD	$2, 3, \ldots, 8$	2^d	1,000

reachability analysis. Finally, in line 5, we determine the partial result $Result_i$ by checking the reachability to the accepting locations.

An example is in the appendix (Example C.1).

The intermediate states $Conf_i$ is the set of last states of the runs of $\mathcal{A}$ associated with $w[i]$ and of duration τ_i. Therefore, we have the following.

Theorem 6.19 (correctness of Algorithm 10)**.** *Given a timed quantitative word w and an LHA $\mathcal{A}$, Algorithm 10 returns the sequence $Result_1, \ldots, Result_n$ satisfying $Result_i = \top \iff w[i] \in \mathcal{L}_{\mathrm{mon}}(\mathcal{A})$.* □

6.8 Experimental evaluation

We experimentally evaluated our model-bounded monitoring scheme, where we used the two procedures for $\mathcal{L}_{\mathrm{mon}}$ membership. For the first procedure (reduction to reachability analysis, in Section 6.6), we used PHAVerLite [BZ19] for conducting reachability analysis. For the second direct procedure (in Section 6.7), we implemented a prototypical tool HAMoni.

We pose the following research questions.

RQ1 Is our dedicated implementation HAMoni worthwhile, performance-wise?

RQ2 Is HAMoni scalable with respect to the length of the input log?

RQ3 How is the scalability of PHAVerLite and HAMoni with respect to the dimension of the bounding model?

Remark 6.20. The following questions are future work.

Q4 How precise is the proposed monitoring scheme? What is the rate of false alerts?

Q5 How easy is it to analyze an alert and to see if it is true or false?

We leave out Q4 and Q5 since, in this chapter, we focus on the algorithms for the model-bounded monitoring scheme (Fig. 6.4). In contrast, Q4 and Q5 are questions on the quality of the *input* to the algorithms. To answer them, we need to address how a bounding model $\mathcal{A}$ is obtained. The latter is a big engineering problem—see the different ways to obtain $\mathcal{A}$ in the introduction—that deserves a separate methodology paper.

We note that the algorithmic performance evaluation in this chapter will remain valid in addressing Q4 and Q5, since it is not affected by the tightness of $\mathcal{A}$.

6.8.1 Benchmarks

In the experiments, we used the following three benchmarks on adaptive cruise controller: Piecewise-Constant ACC (ACCC); Interval ACC (ACCI); and Diagonal ACC (ACCD). The bounding models for the benchmarks are mostly taken from the literature (see below); they

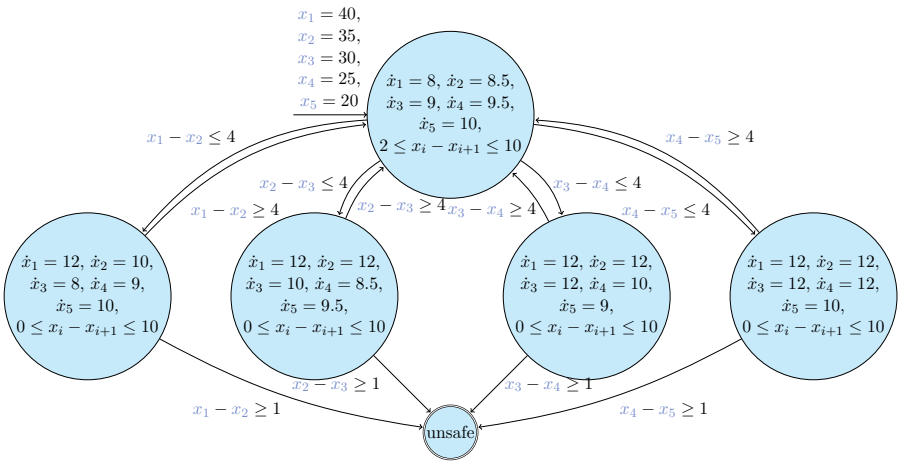


Figure 6.9: The LHA of dimension 5 in ACCC, where $i \in \{1, 2, 3, 4\}$

express keeping the inter-vehicular distance by switching between the normal cruise mode and the recovery mode. This is much like Fig. 6.6a.

The input logs w were randomly generated by following the flows and the transitions of the bounding model $\mathcal{A}$. This means that our SUM is the bounding model itself (Fig. 6.4). We note that this coincidence is not mandatory. More realistic evaluation is future work where a bounding model is obtained in the ways described in the introduction (the "Bounding Models" paragraph). Table 6.1 summarizes the benchmarks.

Piecewise-constant ACC (ACCC) The bounding models for ACCC are taken from [BRS19]. The accepting locations of ACCC happen to be unreachable, therefore there will be no alerts. This is no problem because we focus on algorithmic performance (Remark 6.20). In ACCC, the velocities of the cars at each location are constant. ACCC contains three LHAs of dimensions 5, 10, and 15. Fig. 6.9 is the LHA of dimension 5.

Interval ACC (ACCI) ACCI is a variant of the ACCC benchmark. In the bounding model for ACCI, the velocities of the cars at each location are nondeterministically chosen from the given interval. It is shown in Fig. 6.6b.

Diagonal ACC (ACCD) The bounding models for ACCD are taken from [FAA+19]. In ACCD, the velocities of the cars at each location are constrained by the following diagonal constraints (i. e., constraints of the form $x_i - x_j \bowtie n$, $n \in \mathbb{Z}$): when recovering the distance between x_i and x_{i+1}, we have $|\dot{x}_i - \dot{x}_{i+1} - \varepsilon| < 1$, where ε is the slow-down parameter; otherwise, we have $|\dot{x}_i - \dot{x}_{i+1}| < 1$. We used $\varepsilon = 0.9$ and $\varepsilon = 2.0$. The safety specification in ACCD is $x_i > x_{i-1}$ for each i. ACCD contains seven LHAs of dimensions from 2 to 8. The LHAs of dimension 2 are shown in Fig. 6.10.

We use Definition 6.12 to construct $\mathcal{A}_{\neg\varphi}$. In some benchmarks, we apply manual optimization and collapse some equivalent accepting states.

6.8.2 Experiments

For the reachability analysis in the procedure presented in Section 6.6, we used PHAVerLite 0.2.1. PHAVerLite relies on PPLite [BZ19] to compute symbolic states. We implemented an

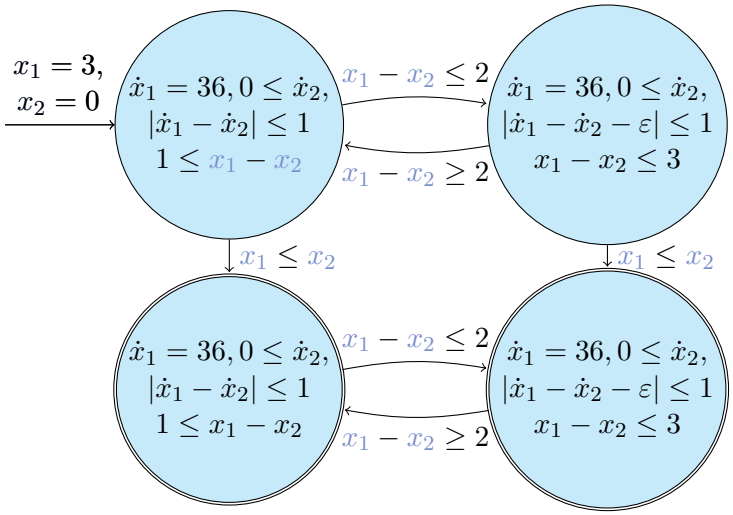


Figure 6.10: The LHA of dimension 2 in ACCD

Table 6.2: The experiment result on ACCC [sec.]

dim.	len.	PHAVerLite	HAMoni
5	10	500.02	0.46
5	25	540.02	0.86
5	50	480.06	0.78
5	75	580.01	0.79
5	100	520.07	1.21
5	200	560.06	1.00
5	300	540.13	1.37
5	400	T.O.	1.77

dim.	len.	PHAVerLite	HAMoni
5	500	480.41	2.27
5	600	500.40	2.12
5	700	520.40	2.37
5	800	540.33	2.58
5	900	580.12	2.76
5	1000	560.26	3.26
10	10	267.60	204.79
15	10	T.O.	T.O.

Table 6.3: The experiment result on ACCI [sec.]

len.	PHAVerLite	HAMoni
10000	13.42	2.18
20000	40.29	4.29
30000	83.39	6.43
40000	141.55	8.56
50000	225.23	10.76

len.	PHAVerLite	HAMoni
60000	116.67	12.86
70000	26.60	15.00
80000	227.29	17.14
90000	259.17	19.28
100000	227.12	21.45

OCaml program and a Python script to construct a PHAVerLite model from an LHA $\mathcal{A}$ and a timed quantitative word w. For the procedure presented in Section 6.7, we implemented HAMoni in C++ using Parma Polyhedra Library (PPL) [BHZ08] and compiled using GCC 7.4.0. In both PHAVerLite and HAMoni, closed convex polyhedra are used to analyze the reachability [BZ19].

Since the difficulty of the $\mathcal{L}_{\text{mon}}$ membership problem depends on the given timed quantitative word w, we randomly generated 30 logs for each experiment setting and measured the average of the execution time. The sampling interval of the w is from 1 to 5 seconds, uniformly distributed. The timeout is 10 minutes. We conducted the 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). Tables 6.2 to 6.5 summarize the experiment results.

Table 6.4: The experiment result on ACCD (dimension 2–6) [sec.]

		$\varepsilon = 0.9$		$\varepsilon = 2.0$	
dim.	len.	PHAVerLite	HAMoni	PHAVerLite	HAMoni
2	10	0.02	0.00	0.02	0.00
2	25	0.03	0.00	0.03	0.00
2	50	0.04	0.00	0.04	0.00
2	75	0.05	0.01	0.05	0.00
2	100	0.06	0.01	0.06	0.01
2	200	0.12	0.02	0.11	0.01
2	300	0.16	0.02	0.16	0.02
2	400	0.21	0.03	0.20	0.02
2	500	0.27	0.04	0.26	0.03
2	600	0.32	0.04	0.30	0.03
2	700	0.35	0.05	0.36	0.04
2	800	0.40	0.05	0.40	0.05
2	900	0.46	0.06	0.45	0.05
2	1000	0.51	0.07	0.50	0.06
3	10	0.05	0.02	0.04	0.01
3	25	0.08	0.03	0.08	0.02
3	50	0.14	0.04	0.12	0.03
3	75	0.19	0.05	0.18	0.03
3	100	0.23	0.05	0.22	0.04
3	200	0.44	0.07	0.42	0.05
3	300	0.65	0.09	0.62	0.06
3	400	0.84	0.12	0.82	0.09
3	500	1.07	0.15	1.00	0.10
3	600	1.25	0.16	1.24	0.11
3	700	1.46	0.16	1.42	0.13
3	800	1.73	0.20	1.61	0.13
3	900	1.84	0.19	1.83	0.16
3	1000	2.04	0.20	2.00	0.17
4	10	0.28	0.35	0.22	0.42
4	25	0.46	0.46	0.34	0.22
4	50	0.66	0.42	0.59	0.38
4	75	0.92	0.47	0.82	0.36
4	100	1.21	0.64	1.05	0.35
4	200	2.13	0.82	2.02	0.87
4	300	2.98	0.74	2.81	0.40
4	400	3.98	0.92	3.76	0.62
4	500	4.79	0.85	4.69	0.65
4	600	5.71	0.86	5.63	0.68
4	700	6.60	0.81	6.53	0.93
4	800	7.67	1.15	7.28	0.97
4	900	8.39	1.04	0.02	0.87
4	1000	9.26	1.18	9.15	0.89
5	10	2.65	7.51	22.38	29.33
5	25	3.84	10.13	3.05	9.22
5	50	5.25	9.68	12.93	31.06
5	75	7.13	14.41	18.32	29.93
5	100	8.14	12.87	7.58	21.13
5	200	11.39	8.94	29.52	36.17
5	300	17.33	15.03	15.38	14.56
5	400	20.44	9.81	19.47	15.34
5	500	24.87	11.72	36.83	34.12
5	600	28.78	8.79	28.47	16.37
5	700	35.14	17.05	32.86	16.10
5	800	37.76	12.41	41.97	39.07
5	900	42.22	9.83	48.26	46.68
5	1000	47.11	11.55	57.49	54.44
6	10	47.42	221.55	80.26	428.66
6	25	41.18	165.63	76.40	510.55
6	50	65.36	243.97	120.77	518.01
6	75	58.21	173.46	125.61	480.19
6	100	87.47	209.53	131.88	421.59

Table 6.5: The experiment result on ACCD (dimension 7 and 8) [sec.]

dim.	len.	$\varepsilon = 0.9$		$\varepsilon = 2.0$	
		PHAVerLite	HAMoni	PHAVerLite	HAMoni
7	10	525.07	560.69	T.O.	594.05
7	25	489.35	559.56	T.O.	526.14
7	50	514.10	562.68	T.O.	566.01
7	75	T.O.	588.23	T.O.	583.49
7	100	T.O.	577.29	T.O.	578.39
8	10	T.O.	598.23	T.O.	T.O.
8	25	T.O.	T.O.	T.O.	T.O.
8	50	T.O.	T.O.	T.O.	593.32
8	75	T.O.	593.84	T.O.	T.O.
8	100	T.O.	T.O.	T.O.	T.O.

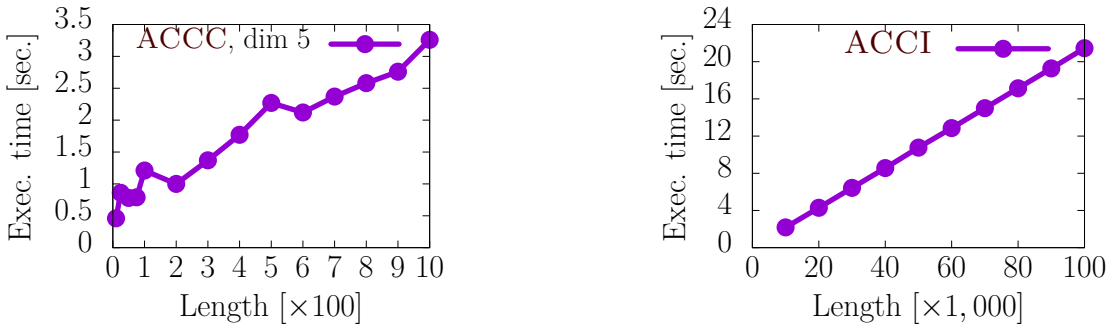


Figure 6.11: The execution time of HAMoni for ACCC dimension 5 (left) and ACCI (right)

RQ1: worthwhileness of a dedicated implementation In Tables 6.2 to 6.5, we observe that HAMoni tends to outperform PHAVerLite. Especially, in Table 6.2, we observe that for ACCC, HAMoni performed drastically faster than PHAVerLite for dimension 5. This is because PHAVerLite is not a specific tool for the $\mathcal{L}_{\text{mon}}$ membership problem but a tool for reachability analysis in general. Thus, despite the engineering cost for the implementation that is not small, a dedicated solve i. e., HAMoni is worthwhile.

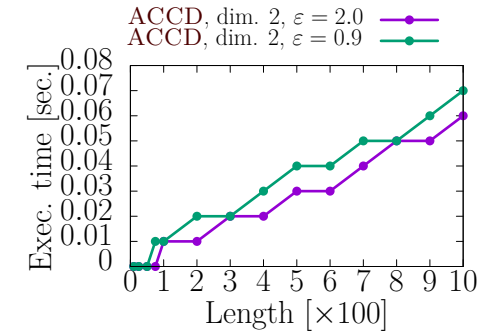
However, in Tables 6.4 and 6.5, we also observe that in ACCD, when the dimension of the LHA is relatively large and the timed quantitative word is not too large, PHAVerLite often outperformed HAMoni. This is because of the more optimized reachability analysis algorithm in PHAVerLite.

Optimization of the reachability analysis algorithm in HAMoni e. g., utilizing the techniques in [BZ19] is a future work.

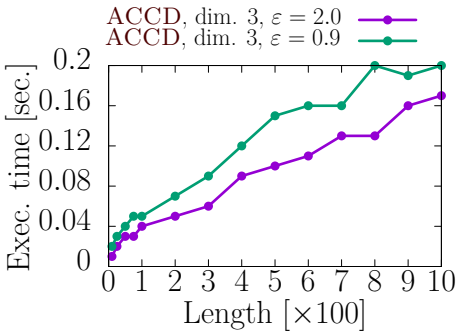
RQ2: scalability with respect to the word length Figs. 6.11 and 6.12 show the execution time of HAMoni with respect to the length of the timed quantitative word for selected experiment settings.

In Figs. 6.11, 6.12a and 6.12b, we observe that when the dimension of the LHA is not large, the execution time was more or less linear to the word length. This is because, when the number of the intermediate states ($Conf_i$ in Algorithm 10) is constant and the execution time of the bounded-time reachability analysis (line 3 of Algorithm 10) is constant for each iteration, the execution time of Algorithm 10 is linear to the word length. Thanks to the merging of the convex polyhedra, such saturation often happens when the word length is long enough for the complexity of the LHA.

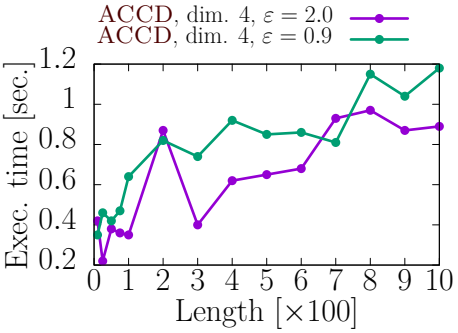
In Figs. 6.12c and 6.12d, we observe that for ACCI of dimension 5, the execution time was more or less constant with respect to the word length. Such behavior also happens for



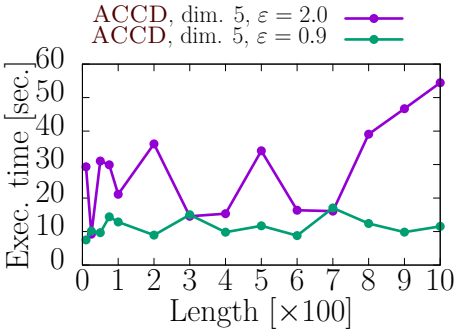
(a) Result for ACCD dimension 2



(b) Result for ACCD dimension 3



(c) Result for ACCD dimension 4



(d) Result for ACCD dimension 5

Figure 6.12: The execution time of HAMoni for ACCD

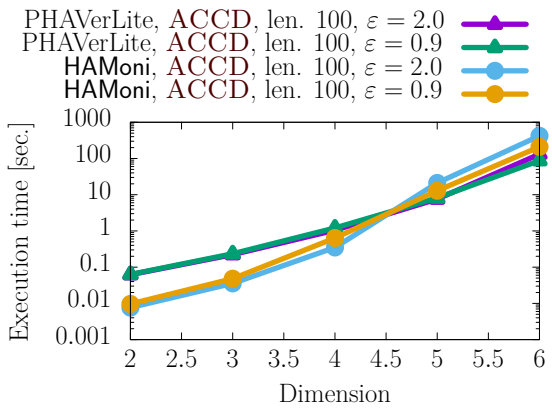


Figure 6.13: The execution time of PHAVerLite and HAMoni for ACCD fixing the word length to be 100

other benchmarks when the word length is short e. g., when the word length is less than 200 for ACCC of dimension 5.

Overall, in our experiments, the execution time was less than linear, and we conclude that at least for many benchmarks, HAMoni is scalable to the word length.

RQ3: scalability with respect to the dimensionality of the LHA Fig. 6.13 shows the execution time of PHAVerLite and HAMoni to the dimension of the LHA, where the length of the log is fixed to be 100. As we mentioned in Section 6.8.2, the sampling interval is from 1 to 5 seconds, uniformly distributed; therefore, each w spans 300 seconds in average. Note that

the y-axes of Fig. 6.13 follow a logarithmic scale.

In Fig. 6.13, we observe that the execution time is more or less exponential to the dimension of the LHA. This is due to the exponential complexity of the convex polyhedra operations. However, in Tables 6.2 to 6.5, we observe that for any benchmark (excluding ACCI, which is not suitable for this discussion)[3], HAMoni can effectively process a huge log up to around 5 dimensions. This is important for monitoring where the log tends to be huge while the dimension of the bounding model may not be much large.

On the other hand, in Fig. 6.13, we observe that PHAVerLite is more scalable to the model dimension than HAMoni. This is thanks to the optimized convex polyhedra algorithms and implementations in PHAVerLite. Again, our future work will consist in optimizing HAMoni with the most promising features of PHAVerLite, e. g., by using the techniques from [BZ19].

Overall, our experiment results suggest that, for signals of up-to 5 dimensions, our monitoring works more or less in real-time because both of the implementation handle the logs spans at least 100 seconds (a few paragraphs ago) in less than 30 seconds on average (Fig. 6.13).

6.9 Related work

In the IoT applications [GBMP13], energy efficiency is of paramount importance. Energy efficiency demands longer sampling and communication intervals; the current work presents an automatic and sound method to mitigate the uncertainties that result from those longer intervals.

In the context of quality assurance of CPSs, monitoring of *digital* (i. e., discrete-valued) or *analog* (i. e., continuous-valued) signals takes an important role. There have been many works on signal monitoring using various logic e. g., *signal temporal logic (STL)* [MN04, FP09], *timed regular expressions (TREs)* [UFAM14], *timed automata (TAs)* [BFN+18], or *timed symbolic weighted automata (TSWAs)* [Wag19] and in Chapter 5. However, in most of the existing works, interpolation of the sampled signals is limited to only piecewise-constant or piecewise-linear. Recently, in [APM19], the authors presented more general signal interpolation methods using signal processing techniques; they showed that the choice of interpolation methods have a significant effect on the monitoring result.

There are a few works on monitoring utilizing system models. In [ZLD12], a set of predictive words are generated through a static analysis of the monitored program and monitored against linear temporal logic. In [PJT+17], the system model and the property are given as TAs to construct a monitor predicting the satisfaction (or violation) of the monitored property. In [BGF18], the stochastic system model is trained as a hidden Markov model, and the predicted system behavior is monitored against a specification in a DFA.

Overall, prediction (i. e., *extrapolation*) of the future behaviors is the main purpose of the existing model-based monitoring works [ZLD12, PJT+17, BGF18] to the best of our knowledge. Our approach utilizes system knowledge for *interpolation* of the infrequently sampled signals.

There are existing language notions for LHAs [AKV98]. These are different from the notion $\mathcal{L}_{\mathrm{mon}}$ that we introduce; hence the results in [AKV98] do not subsume ours. The key difference is whether we require an input word and mode switches synchronize; see Example 6.11 and the preceding discussions.

[3] ACCI is not suitable to observe the scalability with respect to the model dimension because ACCI consists of only one LHA of dimension 2.

6.10 Conclusion and perspectives

6.10.1 Conclusion

Based on a novel language notion $\mathcal{L}_{\text{mon}}$ for LHAs, we formulated what we call the *model-bounded monitoring* scheme for hybrid systems. It features the use of a bounding model of the system to bridge the gap between (continuous-time) system behaviors and (discrete-time) logs that a monitor can access. While the $\mathcal{L}_{\text{mon}}$ membership problem is undecidable, our two partial algorithms (especially our dedicated HAMoni) work well for automotive platooning benchmarks. Overall, our results show the power of symbolic manipulation of polyhedra in the domain of cyber-physical systems.

6.10.2 Perspectives

So far, HAMoni is a quickly developed prototype. Optimization of the reachability analysis using the technique in [CÁF11] is a first future work. In addition, importing the latest optimizations from PHAVerLite [BZ19] into HAMoni is on our agenda.

One potential extension of the $\mathcal{L}_{\text{mon}}$ membership problem is to make it more quantitative: returning a distance between the monitored language $\mathcal{L}_{\text{mon}}(\mathcal{A})$ and the observation w rather than checking the membership of the observation w to the monitored language $\mathcal{L}_{\text{mon}}(\mathcal{A})$. This makes the result of the analysis even more useful. Another extension is to use intervals rather than points for the measured values to cope with the uncertainty in the measured values.

In the current setting, the safety verdict by the monitor is *guaranteed* thanks to the convex polyhedra analysis. A testing-based approach e. g., in [DN09] is a future work for more efficiency at the expense of the safety guarantee.

Finally, the precision of the proposed scheme relies on the quality of a bounding model $\mathcal{A}$. Its evaluation involves big engineering problems, including how we obtain $\mathcal{A}$ and how an alert (as an outcome of Fig. 6.4) is manually matched against the SUM (instead of against $\mathcal{A}$). See Remark 6.20. This is future work that is best studied with a real-world industry example.

CHAPTER 7

Discussion

7.1 Conclusions

High-level contribution: polyhedra-based abstraction is useful to improve runtime verification The high-level contribution of this thesis is to show that polyhedra-based abstraction is useful for improving runtime verification by conducting concrete improvements of runtime verification. We show the usefulness by extensively using polyhedra-based analysis in various advanced runtime verification algorithms. As shown in Fig. 7.1, polyhedra-based abstraction allows a symbolic analysis of the continuously many points in each area rather than an analysis of each point.

Polyhedra-based abstraction is typically used for exhaustive verification of automata with continuous state space, e. g., timed automata [AD94], parametric timed automata [HRSV02], and linear hybrid automata [AHH96]. For these class of automata, we cannot check if a state is reachable by BFS or DFS on the concrete state space because we have *continuous* option of time elapse. For example, a combination of the time elapse at the first and the second locations can be represented by a point in the left of Fig. 7.1. Since there are continuously many such combination, we cannot try all of them. Nevertheless, the reachability checking is tractable thanks to the *discrete* abstraction of the continuous state space by using polyhedra. In the

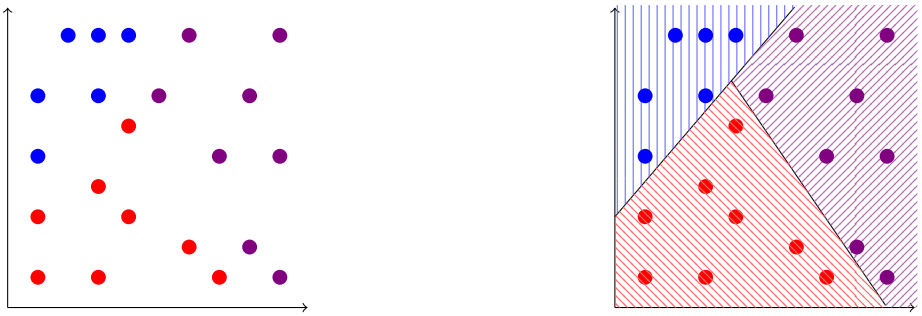


Figure 7.1: Concrete points in a continuous 2-dimensional space (left) and their polyhedra-based abstraction (right). By using polyhedra-based abstraction (right), we can symbolically analyze continuously many points in each area while by evaluating the concrete points (left), we can evaluate only finite points.

example in the right of Fig. 7.1, the continuously many possible time elapses are abstracted into the three areas. See Fig. 1.27 for a concrete example of such an analysis.

In naive runtime verification, e. g., monitoring of LTL properties [BLS11], such an abstraction is unnecessary because we have only finite branching in the LTL monitor and the log is finite length. However, in advanced runtime verification techniques with continuous options, we have a similar difficulty to exhaustive verification. Again, we used polyhedra-based abstraction to solve this issue. For example, in parametric timed pattern matching in Chapter 3, we have to consider continuously many combinations of the parameter valuations, much like the points in the left of Fig. 7.1. We used polyhedra-based abstraction to abstract the parameter space (much like in the right of Fig. 7.1) and solved the parametric timed pattern matching problem using an algorithm similar to an algorithm for non-parametric timed pattern matching [WAH16, WHS17].

Concrete improvements and usage of polyhedra In Chapters 3 to 6, we improved runtime verification using polyhedra-based abstraction to show the aforementioned high-level contribution. The following summarizes the usages of polyhedra and the enhanced features in each chapter. Here, we categorize the improvements into *genericity*, *flexibility*, *informativeness*, and *efficiency*.

Chapter 3: Parametric timed pattern matching

Usage of polyhedra We used polyhedra to discretize the real-valued parameter space as well as the matching intervals. Such a symbolic analysis is essential for runtime verification with parametric specification.

Flexibility By using timing parameters, we allow incomplete timing constraints (i. e., constraints with unknown threshold) in the monitored specification.

Informativeness Parametric timed pattern matching returns the set of feasible parameter valuations as well as the corresponding intervals. By using the parameters, we can synthesize the safety degree. For example, we can encode the robustness by replacing a guard $x < 3$ with $x < 3 + p_x$, where p_x is a parameter showing the satisfaction degree.

Usage of polyhedra We used polyhedra in the skip value computation. Concretely, the skip value functions Δ_{KMP} and Δ_{QS} defined in Definitions 3.11 and 3.14 require the emptiness checking of the parametric timed language, which is solved by the reachability checking with polyhedra-based abstraction

Efficiency Since the computationally expensive part of the skip value functions only depends on the PTA given as a specification, we can compute it before starting monitoring. Since the skip value function reduces the number of the matching trials, it makes the whole algorithm efficient as shown in Section 3.4.2. We note that this optimization does not change the result thanks to the soundness Theorems 3.12 and 3.15.

Chapter 4: Symbolic monitoring against specifications parametric in time and data

Usage of polyhedra In addition to the usage of polyhedra to abstract the timing parameter space in Chapter 3, we use polyhedra to abstract the parameter space of the numeric data and parameter valuations.

Flexibility In addition to the timing parameters in Chapter 3, we allow parameters in infinite domain data.

Informativeness Similarly to Chapter 3, we can synthesize the safety degree using the parameters. Moreover, we can extract timing and data information from the log, as shown in Section 4.5.3.

Chapter 5: Online quantitative timed pattern matching with semiring-valued weighted automata

Usage of polyhedra In signal monitoring, there are continuously many possible timings of the switching in a temporal specification due to the continuity of the signals. We use polyhedra to obtain discrete abstraction of such possibilities.

Genericity The quantitative semantics is parameterized with semiring $\mathbb{S}$ and the cost function κ. Our algorithm works for any complete idempotent semiring.

Informativeness Quantitative timed pattern matching computes the quantitative semantics for each subsignal. This quantitative semantics represents e. g., unsafe degree, which is more informative than mere Boolean semantics.

Chapter 6: Model-bounded monitoring of hybrid systems

Usage of polyhedra In model-bounded monitoring, we symbolically interpolate the sampled points using convex polyhedra.

Flexibility We introduce signal monitoring with flexible interpolation of the sampled log, considering the system's prior knowledge.

Informativeness By using the extension detailed in Appendix D, we can synthesize parameter valuations and unobserved signal values, much like Chapters 3 and 4.

7.2 Perspectives

Some future directions are as follows. We believe the use of polyhedra-based abstraction will be also useful in many of these future directions.

7.2.1 Short-term: Improvement of the current methods

Improvements of the current methods are the short-term future works. The following are the examples.

Improvement of the tool interface We implemented several prototypical tools to evaluate the practical relevance of our approach. Although we can conduct monitoring with these tools, we need more improvement, especially on the interface, to make it useful for non-specialists. For example, an improvement in the syntax, better presentation of the monitored result, and giving a GUI will be useful for many users.

Integration with other tools Our current tools monitor the log from a file or the standard input. Thanks to the flexibility of the pipeline in Unix-like operating systems, any system can be monitored if the log is written to a file or the standard output. Nevertheless, it will also be useful to implement an integration with widely used simulators or middleware such as Simulink or ROS [QCG+09].

Investigating an efficient alternative for embedded usage Since the computation resource is limited in embedded devices, it will be useful to investigate an efficient alternative with a scalability guarantee, e. g., the number of the loops for each input is bounded, or the memory allocation can be static. For such efficiency, we can, for example, restrict the specification or make the output less informative.

7.2.2 Easing formal specification writing

Even if we allow using parameters in defining a specification, it is still challenging to write a specification in a formal language, e. g., automata or temporal logic. The following are some future works for such difficulty.

Validation of the formal specification "Does our formal specification specify what we want?" This is a natural question after writing a formal specification. One of the methods to answer this question is to test the formal specification. Namely, we feed logs to the specification and check if the monitoring result is as we expected. For example, STLInspector [RHM17] is a tool in this direction. Although such a test itself is nothing but a naive runtime verification, advanced usage of the test results, e. g., repairing the specification from the failed test inputs, would require advanced runtime verification techniques and the theory of automata or logics.

Another method to answer the question is to generate the logs illustrating the specification, e. g., typical logs satisfying or violating the specification and the borderline logs. Such an enumeration would require a symbolic analysis of the specification.

Combination with anomaly detection Another direction is not to write a formal specification but to learn it from labeled or unlabeled training data, or through the questions to the oracle. Although the high-level picture of this direction is the same as anomaly detection, we can potentially improve the explainability in a specification-free approach because the learned formal specification is white-box.

Grammatical inference [dlH16, LG16] is a research area on such learning, where a formal language, typically represented by an automaton, is learned. The typical setting in grammatical inference is the exact learning of a language over a finite alphabet, and it is still challenging to learn from numeric data with noise.

We note that the parametric runtime verification in Chapters 3 and 4 can also be used to learn a concrete formal specification from a parametric specification given as a template, e. g., in [FdSC+17]. However, we have to give an appropriate parametric specification; this is challenging in some usage scenarios.

7.2.3 Improvement in networked monitoring

It is getting common to conduct monitoring in an IoT setting, where the monitored log is sampled on an embedded device that is connected to other computers via the internet. The

following are some challenges for IoT monitoring.

Efficiency in the embedded devices In IoT monitoring, on the one hand, the computational resource of the embedded devices with sensors tends to be limited, and it makes sense to delegate the monitoring task to another computer. On the other hand, since communication requires much energy, it also makes sense to reduce the communication frequency to make the embedded device energy efficient.

Symbolic analysis is useful for such a networked monitoring with small frequency. One of the usage of model-bounded monitoring in Chapter 6 is to reduce the communication frequency with the soundness guarantee at the cost of false alarms. In [WH18], by a symbolic analysis of the specification, a Moore-machine filter is constructed to reduce the amount of the log by removing the subsequence whose safety is efficiently determinable.

In these works, a scenario with one embedded system and one high-performance server is considered. Optimization for other setting, e. g., task distribution among multiple embedded systems, is future work.

Monitoring of distributed systems with no synchronous clock In IoT monitoring, it is also often the case that the monitoring is conducted by combining the information from different devices with various sensors. In such a situation, precisely synchronous clock among the embedded devices is often expensive. To conduct monitoring without precisely synchronous clocks, monitoring considering clock perturbation or monitoring with logical clocks [KS08] would be future directions.

7.2.4 Future vision: Monitoring to accelerate the innovation

Runtime verification automates the analysis of system behavior and makes it efficient. Since this reduces the cost of the evaluation after prototyping, runtime verification accelerates the cycle of prototyping and evaluation. Moreover, because the alarm raised by runtime verification can trigger the emergency brake, runtime verification can make hardware test safer. Overall, we believe that, in the long term, runtime verification will make system development more efficient and accelerate the innovation of technologies.

In terms of the application of automata and formal language theory, grep is a very successful utility used in various systems. Given a regular expression and a text, grep finds the matching of the regular expression in the text. Despite its simplicity, grep is a powerful utility to analyze various text files, e. g., source code, configuration file, and log, and it is one of the standard commands in Unix-like operating systems. Moreover, a similar text search technique is implemented in various text editors and IDEs to make the development efficient.

We believe that in the future, the advanced runtime verification techniques in this thesis will be used in various systems as a handy utility for system log analysis, much like grep for text analysis.

APPENDIX A

Construction of $V_{\ell,n}$ in Chapter 3

We present a construction of $V_{\ell,n}$ in Definition 3.11. The construction is by using EFsynth in Section 3.1.2. We fix a PTA $\mathcal{A} = (\Sigma, L, \ell_0, L_F, \mathbb{X}, \mathbb{P}, E)$, a location $\ell \in L$, and $n \in \mathbb{Z}_{>0}$. Since $V_{\ell,n}$ is the set of parameter valuations $v \in (\mathbb{Q}_+)^{\mathbb{P}}$ such that there is a $v' \in (\mathbb{Q}_+)^{\mathbb{P}}$ satisfying $\mathcal{L}(v(\mathcal{A}_\ell)) \cdot \mathcal{T}(\Sigma) \cap \mathcal{T}^n(\Sigma) \cdot \{w'' + t \mid w'' \in \mathcal{L}_{-\$}(v'(\mathcal{A})), t > 0\} \cdot \mathcal{T}(\Sigma) \neq \emptyset$, $V_{\ell,n}$ is constructed by EFsynth of $\mathcal{A}'_\ell \parallel \mathcal{A}'_{+n}$, where $\mathcal{A}'_\ell$ and $\mathcal{A}'_{+n}$ are the PTAs satisfying $\mathcal{L}(v(\mathcal{A}'_\ell)) = \mathcal{L}(v(\mathcal{A}_\ell)) \cdot \mathcal{T}(\Sigma)$ and $\mathcal{L}(v'(\mathcal{A}'_{+n})) = \mathcal{T}^n(\Sigma) \cdot \{w + t \mid w \in \mathcal{L}_{-\$}(v'(\mathcal{A})), t > 0\} \cdot \mathcal{T}(\Sigma)$.

We define $\mathcal{A}'_\ell$ as $\mathcal{A}_\ell = (\Sigma, L \amalg \{\ell_{\text{fin}}\}, \ell_0, \{\ell, \ell_{\text{fin}}\}, \mathbb{X}, \mathbb{P}, E'_\ell)$, where $E'_\ell = E \cup \{(\ell, \top, a, \emptyset, \ell_{\text{fin}}) \mid a \in \Sigma\} \cup \{(\ell_{\text{fin}}, \top, a, \emptyset, \ell_{\text{fin}}) \mid a \in \Sigma\}$. For any $v \in (\mathbb{Q}_+)^{\mathbb{P}}$, $w' \in \mathcal{L}(v(\mathcal{A}_\ell))$, and $w'' \in \mathcal{T}(\Sigma)$, we have $\ell_0 \xrightarrow{w'} \ell \xrightarrow{w''} \ell_{\text{fin}}$ in $v(\mathcal{A}'_\ell)$ and $w' \cdot w'' \in \mathcal{L}(v(\mathcal{A}'_\ell))$. For any $v \in (\mathbb{Q}_+)^{\mathbb{P}}$ and $w \in \mathcal{L}(v(\mathcal{A}'_\ell))$, there exist timed words $w', w'' \in \mathcal{T}(\Sigma)$ satisfying $w = w' \cdot w''$ and $\ell_0 \xrightarrow{w'} \ell$ in $v(\mathcal{A}'_\ell)$, which implies $w' \in \mathcal{L}(v(\mathcal{A}_\ell))$. Therefore, we have $\mathcal{L}(v(\mathcal{A}'_\ell)) = \mathcal{L}(v(\mathcal{A}_\ell)) \cdot \mathcal{T}(\Sigma)$.

We define $\mathcal{A}'_{+n}$ as $\mathcal{A}_{+n} = (\Sigma \amalg \{\varepsilon\}, L', \ell_{n+1}, L'_F, \mathbb{X}, \mathbb{P}', E')$, where

- ε is the unobservable character;
- $L' = L \amalg \{l_{\text{i}} \mid i \in \{1, 2, \ldots, n+1\}\} \amalg \{l_{\text{fin}}\}$;
- $L'_F = \{\ell \mid \exists \ell' \in L_F. (\ell, g, a, R, \ell') \in E\} \amalg \{\ell_{\text{fin}}\}$;
- $\mathbb{P}'$ is a disjoint copy of $\mathbb{P}$; and
- $E' = E \amalg \{(l_{i+1}, \top, a, \emptyset, l_i) \mid a \in \Sigma, i \in \{1, 2, \ldots, n\}\} \amalg \{(l_1, \top, \varepsilon, \mathbb{X}, \ell_0)\} \amalg \{(l, \top, a, \emptyset, l_{\text{fin}}) \mid a \in \Sigma, l \in L'_F\} \amalg \{(l_{\text{fin}}, \top, a, \emptyset, l_{\text{fin}}) \mid a \in \Sigma\}$, where any parameter $p \in \mathbb{P}$ is replaced with $p' \in \mathbb{P}'$.

For any parameter valuation $v' \in (\mathbb{Q}_+)^{\mathbb{P}'}$, timed words $w' \in \mathcal{T}^n(\Sigma)$, $w'' \in \mathcal{L}(v'(\mathcal{A}))$, $w''' \in \mathcal{T}(\Sigma)$, and $t \in \mathbb{R}_{>0}$, we have $\ell_{n+1} \xrightarrow{w'} \ell_1 \xrightarrow{(\varepsilon,t)} \ell_0 \xrightarrow{w''} \ell_f \xrightarrow{w'''} \ell_{\text{fin}}$ in $v'(\mathcal{A})$, where $\ell_f \in L'_F$. For any parameter valuation $v' \in (\mathbb{Q}_+)^{\mathbb{P}'}$ and for any timed word $w \in \mathcal{L}(v'(\mathcal{A}'_{+n}))$, there exist $w' \in \mathcal{T}(\Sigma)$, $w'' \in \mathcal{T}(\Sigma)$, $w''' \in \mathcal{T}(\Sigma)$, and $t \in \mathbb{R}_{>0}$ satisfying $w = w' \cdot (\varepsilon, t) \cdot w'' \cdot w'''$ and

$\ell_0 \xrightarrow{w''} \ell_f$ in $v(\mathcal{A}'_{+n})$, where $\ell_f \in L'_F$, and therefore, we have $w'' \in \mathcal{L}_{-\$}(v'(\mathcal{A}))$. Overall, for any $v \in (\mathbb{Q}_+)^{\mathbb{P}'}$, we have $\mathcal{L}(v'(\mathcal{A}'_{+n})) = \mathcal{T}^n(\Sigma) \cdot \{w + t \mid w \in \mathcal{L}_{-\$}(v'(\mathcal{A})), t > 0\} \cdot \mathcal{T}(\Sigma)$.

To take the intersection of $\mathcal{L}(v(\mathcal{A}'_\ell))$ and $\mathcal{L}(v'(\mathcal{A}'_{+n}))$, we use the synchronous product. For any $v \in (\mathbb{Q}_+)^{\mathbb{P}}$ and $v' \in (\mathbb{Q}_+)^{\mathbb{P}'}$, we have $\mathcal{L}((v \amalg v')(\mathcal{A}'_\ell \parallel \mathcal{A}'_{+n})) = \mathcal{L}(v(\mathcal{A}'_\ell)) \cap \mathcal{L}(v'(\mathcal{A}'_{+n}))$, where $v \amalg v'$ is the parameter valuation $v \amalg v' : \mathbb{P} \amalg \mathbb{P}' \to \mathbb{Q}_+$ such that for any $p \in \mathbb{P}$, $(v \amalg v')(p) = v(p)$ and for any $p' \in \mathbb{P}'$, $(v \amalg v')(p') = v'(p')$. Therefore, we have $V_{\ell,n} = \{v \mid \exists v' \in (\mathbb{Q}_+)^{\mathbb{P}'}. \mathcal{L}((v \amalg v')(\mathcal{A}'_\ell \parallel \mathcal{A}'_{+n})) \neq \emptyset\}$, which can be computed by EFsynth.

APPENDIX B

Omitted proofs of Chapter 5

Definition B.1 (path value)**.** For a WTTS $\mathcal{S} = (Q, Q_0, Q_F, \to, W)$, a sequence $q_0, q_1, \ldots, q_n$ of Q is a *path* of $\mathcal{S}$ if we have $q_0 \to q_1 \to \ldots \to q_n$. For a WTTS $\mathcal{S} = (Q, Q_0, Q_F, \to, W)$ and a path $\pi = q_0, q_1, \ldots, q_n$ of $\mathcal{S}$, the *path value* is $\mu(\pi) = \bigotimes_{i=1}^{n} W(q_{i-1}, q_i)$. ◇

For any WTTS $\mathcal{S} = (Q, Q_0, Q_F, \to, W)$, we have $\alpha(\mathcal{S}) = \bigoplus_{\pi \in ARuns(\mathcal{S})} \mu(\pi)$, where $ARuns(\mathcal{S})$ is the set of paths of $q_0, q_1, \ldots, q_n$ of $\mathcal{S}$ satisfying $q_0 \in Q_0$ and $q_n \in Q_F$.

B.1 Finiteness of the reachable part of WSTTSs

For a WSTTS $\mathcal{S}^{\mathrm{sym}} = (Q^{\mathrm{sym}}, Q_0^{\mathrm{sym}}, Q_F^{\mathrm{sym}}, \to^{\mathrm{sym}}, W^{\mathrm{sym}})$, we denote the reachable set by $\mathrm{Reach}(\mathcal{S}^{\mathrm{sym}}) = Q_0^{\mathrm{sym}} \cup \{q^{\mathrm{sym}} \in Q^{\mathrm{sym}} \mid \exists q_0^{\mathrm{sym}} \in Q_0^{\mathrm{sym}}, q_1^{\mathrm{sym}}, q_2^{\mathrm{sym}}, \ldots, q_m^{\mathrm{sym}} \in Q^{\mathrm{sym}}.\, q_0^{\mathrm{sym}}, q_1^{\mathrm{sym}}, \ldots, q_m^{\mathrm{sym}}, q^{\mathrm{sym}}$ is a path of $\mathcal{S}^{\mathrm{sym}}\}$.

Lemma B.2. *Let $\mathcal{S}^{\mathrm{sym}} = (Q^{\mathrm{sym}}, Q_0^{\mathrm{sym}}, Q_F^{\mathrm{sym}}, \to^{\mathrm{sym}}, W^{\mathrm{sym}})$ be a WSTTS of a signal $\sigma = a_1^{\tau_1} a_2^{\tau_2} \ldots a_n^{\tau_n}$ and a TSWA $\mathcal{W}$. For any $(\ell, Z, \overline{a}) \in \mathrm{Reach}(\mathcal{S}^{\mathrm{sym}})$, $\nu \in Z$, and $x \in \mathbb{X}$, we have $0 \le \nu(x) \le |\sigma|$.*

Proof. Since for any $(\ell, Z, \overline{a}) \in Q^{\mathrm{sym}}$ and $\nu \in Z$, we have $\nu(T) \le |\sigma|$, it suffices to prove $\nu(x) \le \nu(T)$ for any $x \in \mathbb{X}$. Let $(\ell, Z, \overline{a}) \in \mathrm{Reach}(\mathcal{S}^{\mathrm{sym}})$. If $(\ell, Z, \overline{a}) \in Q_0^{\mathrm{sym}}$, we have $Z = \mathbf{0}_{\mathbb{X} \amalg \{T\}}$ and for any $\nu \in Z$ and for any $x \in \mathbb{X}$, we have $\nu(x) = \nu(T) = 0$.

Assume $(\ell, Z, \overline{a}) \notin Q_0^{\mathrm{sym}}$ and let $(\ell', Z', \overline{a}') \in \mathrm{Reach}(\mathcal{S}^{\mathrm{sym}})$ satisfying $((\ell', Z', \overline{a}'), (\ell, Z, \overline{a})) \in \to^{\mathrm{sym}}$. If $\overline{a} = \varepsilon$, there exists $(\ell, g, R, \ell') \in \Delta$ satisfying $Z = \{[\nu']_R \mid \nu' \in Z', \nu' \models g\}$. Since $T \notin R$, $\forall \nu' \in Z', x \in \mathbb{X}.\, \nu'(x) \le \nu'(T)$ implies $\forall \nu \in Z, x \in \mathbb{X}.\, \nu(x) \le \nu(T)$.

If $\overline{a} \neq \varepsilon$, for any $\nu \in Z$, there are $\nu' \in Z'$ and $\tau \in \mathbb{R}_{>0}$ satisfying $\nu = \nu' + \tau$. Therefore, $\forall \nu' \in Z', x \in \mathbb{X}.\, \nu'(x) \le \nu'(T)$ implies $\forall \nu \in Z, x \in \mathbb{X}.\, \nu(x) \le \nu(T)$. □ □

For a nonempty zone $Z \in \mathcal{Z}(\mathbb{X} \amalg \{T\})$ and $x, x' \in \mathbb{X} \amalg \{T, 0\}$, we define $\prec_{Z,x,x'} \in \{<, \le\}$ and $d_{Z,x,x'} \in \mathbb{R} \amalg \{\infty\}$ be the smallest elements satisfying the following, where we define $<$ is smaller than $\le$ and we denote $\nu(0) = 0$.

$$Z = \left\{ \nu \;\middle|\; \bigwedge_{x,x' \in \mathbb{X} \amalg \{T,0\}} (\nu(x) - \nu(x')) \prec_{Z,x,x'} d_{Z,x,x'} \right\}$$

Though the $d_{Z,x,x'}$ may not be integer, its domain is discrete.

Lemma B.3. *Let* $\mathcal{S}^{\mathrm{sym}} = (Q^{\mathrm{sym}}, Q_0^{\mathrm{sym}}, Q_F^{\mathrm{sym}}, \to^{\mathrm{sym}}, W^{\mathrm{sym}})$ *be a WSTTS of a signal* $\sigma = a_1^{\tau_1} a_2^{\tau_2} \dots a_n^{\tau_n}$ *and a TSWA* $\mathcal{W}$*. For any* $(\ell, Z, \overline{a}) \in \mathrm{Reach}(\mathcal{S}^{\mathrm{sym}})$ *and* $x, x' \in \mathbb{X} \amalg \{T, 0\}$*, we have* $d_{Z,x,x'} = \infty$ *or there is* $k_i \in \mathbb{Z}$ *for each* $i \in \{0, 1, \dots, n\}$ *satisfying* $d_{Z,x,x'} = k_0 + \sum_{i=1}^{n} (k_i (\sum_{j=1}^{i} \tau_j))$*.*

Proof. If $(\ell, Z, \overline{a}) \in Q_0^{\mathrm{sym}}$, we have $Z = \mathbf{0}_{\mathbb{X} \amalg \{T\}}$ and we have $d_{Z,x,x'} = 0$ for each $x, x' \in \mathbb{X} \amalg \{T, 0\}$.

Assume $(\ell, Z, \overline{a}) \notin Q_0^{\mathrm{sym}}$ and let $(\ell', Z', \overline{a}') \in \mathrm{Reach}(\mathcal{S}^{\mathrm{sym}})$ satisfying $((\ell', Z', \overline{a}'), (\ell, Z, \overline{a})) \in \to^{\mathrm{sym}}$. If $\overline{a} = \varepsilon$, there exists $(\ell, g, R, \ell') \in \Delta$ satisfying $Z = \{[\nu']_R \mid \nu' \in Z', \nu' \models g\}$. For each $x \in R$, we have $d_{Z,x,0} = d_{Z,0,x} = 0$. For each $x, x' \in \mathbb{X} \amalg \{T, 0\}$, $d_{Z,x,x'}$ is the shortest distance in the graph interpretation of Z', where for each $x \in R$, $d_{Z',x,0}$ and $d_{Z,0,x}$ are replaced with 0 and the edges corresponding to the constraints in g are added. (See e. g., [BY03b] for the graph interpretation of a zone.) We note that the additional edges are with integer weights because the constants in g are integer. Therefore, for each $x, x' \in \mathbb{X} \amalg \{T, 0\}$, there are $k_{x'',x'''} \in \{0, 1\}$ and $k \in \mathbb{Z}$ satisfying $d_{Z,x,x'} = k + \sum_{x'',x''' \in \mathbb{X} \amalg \{T,0\}} k_{x'',x'''} d_{Z',x'',x'''}$. By induction hypothesis, For any $x, x' \in \mathbb{X} \amalg \{T, 0\}$, we have $d_{Z,x,x'} = \infty$ or there is $k_i \in \mathbb{Z}$ for each $i \in \{0, 1, \dots, n\}$ satisfying $d_{Z,x,x'} = k_0 + \sum_{i=1}^{n} (k_i (\sum_{j=1}^{i} \tau_j))$.

If $\overline{a} \neq \varepsilon$, $d_{Z,x,x'}$ are computed by the following procedure.

1. For each $x \in \mathbb{X}$, we replace $(d_{Z',x,0}, \prec_{Z',x,0})$ with $(\infty, <)$.
2. We replace $(d_{Z',T,0}, \prec_{Z',T,0})$ and $(d_{Z',0,T}, \prec_{Z',0,T})$ with $(\sum_{j=0}^{i} \tau_j, <)$ and $(-\sum_{j=0}^{i-1} \tau_j, <)$, or $(\sum_{j=0}^{i} \tau_j, \leq)$ and $(-\sum_{j=0}^{i} \tau_j, \leq)$, respectively.
3. We take the shortest distance in the graph interpretation of Z', where some weights are replaced by the above.

Therefore, for each $x, x' \in \mathbb{X} \amalg \{T, 0\}$ and for each $i \in \{0, 1, \dots, n\}$, there are $k_{x'',x'''} \in \{0, 1\}$ and $k_i \in \mathbb{Z}$ satisfying the following.

$$d_{Z,x,x'} = \sum_{x'',x''' \in \mathbb{X} \amalg \{T,0\}} k_{x'',x'''} d_{Z',x'',x'''} + \sum_{i=1}^{n} \left(k_i \sum_{j=1}^{i} \tau_j \right)$$

By induction hypothesis, For any $x, x' \in \mathbb{X} \amalg \{T, 0\}$, we have $d_{Z,x,x'} = \infty$ or there is $k_i \in \mathbb{Z}$ for each $i \in \{0, 1, \dots, n\}$ satisfying $d_{Z,x,x'} = k_0 + \sum_{i=1}^{n} (k_i (\sum_{j=1}^{i} \tau_j))$. □

Lemma B.4. *For any WSTTS* $\mathcal{S}^{\mathrm{sym}} = (Q^{\mathrm{sym}}, Q_0^{\mathrm{sym}}, Q_F^{\mathrm{sym}}, \to^{\mathrm{sym}}, W^{\mathrm{sym}})$ *and for any* $(\ell, Z, \overline{a}) \in \mathrm{Reach}(\mathcal{S}^{\mathrm{sym}})$*, either* $\overline{a} = \varepsilon$ *holds or there is* $t \in \mathbb{R}_{\geq 0}$ *such that for any* $\nu \in Z$*,* $\overline{a} = \mathit{Values}(\sigma([t, \nu(T)]))$ *holds.*

Proof. Let $(\ell, Z, \overline{a}) \in \mathrm{Reach}(\mathcal{S}^{\mathrm{sym}})$. If $(\ell, Z, \overline{a}) \in Q_0^{\mathrm{sym}}$, we have $\overline{a} = \varepsilon$.

If $(\ell, Z, \overline{a}) \notin Q_0^{\mathrm{sym}}$, there is $(\ell', Z', \overline{a}') \in \mathrm{Reach}(\mathcal{S}^{\mathrm{sym}})$ such that $((\ell', Z', \overline{a}'), (\ell, Z, \overline{a})) \in \to^{\mathrm{sym}}$. If $\overline{a} \neq \varepsilon$, we have $\overline{a} = \overline{a}' \circ \mathit{Values}(\sigma([\nu(T), \nu'(T)]))$. By induction hypothesis, there is $\nu' \in Z'$ such that for any $\nu \in Z$, we have $\overline{a} = \mathit{Values}(\sigma([\nu'(T), \nu(T)]))$ or there is $t \in \mathbb{R}_{\geq 0}$ such that for any $\nu \in Z$, $\overline{a} = \mathit{Values}(\sigma([t, \nu(T)]))$ holds. □ □

Theorem B.5 (finiteness)**.** *For any WSTTS* $\mathcal{S}^{\mathrm{sym}}$*, there are only finitely many states reachable from* Q_0^{sym}*.*

Proof. The locations L is a finite set. By Lemma B.2 and Lemma B.3, the number of zones appearing in $\mathrm{Reach}(\mathcal{S}^{\mathrm{sym}})$ is finitely many. By Lemma B.4, the subsequences $\overline{a}$ appearing in $\mathrm{Reach}(\mathcal{S}^{\mathrm{sym}})$ is finitely many. Therefore, $\mathrm{Reach}(\mathcal{S}^{\mathrm{sym}})$ is a finite set. □ □

B.2 Proof of Theorem 5.13

First, we define symbolic path value.

Definition B.6 (symbolic path value)**.** For a WSTTS $\mathcal{S}^{\mathrm{sym}} = (Q^{\mathrm{sym}}, Q_0^{\mathrm{sym}}, Q_F^{\mathrm{sym}}, \to^{\mathrm{sym}}, W^{\mathrm{sym}})$, a sequence $q_0^{\mathrm{sym}}, q_1^{\mathrm{sym}}, \ldots, q_n^{\mathrm{sym}}$ of Q^{sym} is a *path* of $\mathcal{S}^{\mathrm{sym}}$ if for any $i \in \{1, \ldots, n\}$, we have $(q_{i-1}^{\mathrm{sym}}, q_i^{\mathrm{sym}}) \in \to^{\mathrm{sym}}$. For a WSTTS $\mathcal{S}^{\mathrm{sym}} = (Q^{\mathrm{sym}}, Q_0^{\mathrm{sym}}, Q_F^{\mathrm{sym}}, \to^{\mathrm{sym}}, W^{\mathrm{sym}})$ and a path $\pi^{\mathrm{sym}} = q_0^{\mathrm{sym}}, q_1^{\mathrm{sym}}, \ldots q_n^{\mathrm{sym}}$ of $\mathcal{S}^{\mathrm{sym}}$, the *symbolic path value* is $\mu^{\mathrm{sym}}(\pi^{\mathrm{sym}}) = \bigotimes_{i=1}^{n} W^{\mathrm{sym}}(q_{i-1}^{\mathrm{sym}}, q_i^{\mathrm{sym}})$. ◇

Similarly to the trace value $\alpha(\mathcal{S})$, for any WSTTS $\mathcal{S}^{\mathrm{sym}} = (Q^{\mathrm{sym}}, Q_0^{\mathrm{sym}}, Q_F^{\mathrm{sym}}, \to^{\mathrm{sym}}, W^{\mathrm{sym}})$, we have $\alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}}) = \bigoplus_{\pi^{\mathrm{sym}} \in ARuns(\mathcal{S}^{\mathrm{sym}})} \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}})$, where $ARuns(\mathcal{S}^{\mathrm{sym}})$ is the set of paths of $q_0^{\mathrm{sym}}, q_1^{\mathrm{sym}}, \ldots, q_n^{\mathrm{sym}}$ of $\mathcal{S}^{\mathrm{sym}}$ satisfying $q_0^{\mathrm{sym}} \in Q_0^{\mathrm{sym}}$ and $q_n^{\mathrm{sym}} \in Q_F^{\mathrm{sym}}$.

For a semiring $\mathbb{S} = (S, \oplus, \otimes, e_\oplus, e_\otimes)$, we denote the canonical order by $\preceq \subseteq S \times S$, where $s \preceq s' \iff s \oplus s' = s'$. When $\mathbb{S}$ is idempotent, $s = s'$ if and only of $s \preceq s'$ and $s' \preceq s$ because: if $s = s'$, we have $s \oplus s' = s' \oplus s' = s'$ and $s' \oplus s = s \oplus s = s$; and if $s \preceq s'$ and $s' \preceq s$, we have $s = s \oplus s' = s' \oplus s = s'$.

For simplicity, we assume that for any signal $\sigma = a_1^{\tau_1} a_2^{\tau_2} \ldots a_n^{\tau_n} \in \mathcal{T}(\mathbb{D}^{\mathbb{V}})$ and for any $i \in \{1, 2, \ldots, n-1\}$, we have $a_i \neq a_{i+1}$.

Lemma B.7. *Let $\sigma = a_1^{\tau_1} a_2^{\tau_2} \ldots a_n^{\tau_n} \in \mathcal{T}(\mathbb{D}^{\mathbb{V}})$ be a signal, let $\mathcal{W}$ be a TSWA, and let $\mathcal{S}^{\mathrm{sym}} = (Q^{\mathrm{sym}}, Q_0^{\mathrm{sym}}, Q_F^{\mathrm{sym}}, \to^{\mathrm{sym}}, W^{\mathrm{sym}})$ be the WSTTS of σ and $\mathcal{W}$. For any $(\ell, Z_1, \overline{a}), (\ell', Z_1', \overline{a'}) \in Q^{\mathrm{sym}}$ satisfying $((\ell, Z_1, \overline{a}), (\ell', Z_1', \overline{a'})) \in \to^{\mathrm{sym}}$ and for any $(\ell, Z_2, \overline{a}) \in Q^{\mathrm{sym}}$ satisfying $Z_1 \subseteq Z_2$, there exists $(\ell', Z_2', \overline{a'}) \in Q^{\mathrm{sym}}$ satisfying $Z_1' \subseteq Z_2'$ and $((\ell, Z_2, \overline{a}), (\ell', Z_2', \overline{a'})) \in \to^{\mathrm{sym}}$.*

Proof. If $\overline{a'} = \varepsilon$, there exists $(\ell, g, R, \ell') \in \Delta$ satisfying $Z_1' = \{[\nu]_R \mid \nu \in Z_1, \nu \models g\}$. Since $Z_1 \subseteq Z_2$, $Z_2' = \{[\nu]_R \mid \nu \in Z_2, \nu \models g\}$ is nonempty, we have $((\ell, Z_2, \overline{a}), (\ell', Z_2', \overline{a'})) \in \to^{\mathrm{sym}}$. We also have $Z_1' \subseteq Z_2'$.

If $\overline{a'} \neq \varepsilon$, let M be either $M_{i,=} = \{\nu \mid \nu(T) = \sum_{j=0}^{i} \tau_j\}$ or $M_i = \{\nu \mid \sum_{j=0}^{i-1} \tau_j < \nu(T) < \sum_{j=0}^{i} \tau_j\}$ satisfying $Z_1' = \{\nu + \tau \mid \nu + Z_1, \tau \in \mathbb{R}_{>0}\} \cap M$, where $i \in \{1, 2, \ldots, n\}$. Let $Z_2' = \{\nu + \tau \mid \nu \in Z_2, \tau \in \mathbb{R}_{>0}\} \cap M$. Then, we have $((\ell, Z_2, \overline{a}), (\ell', Z_2', \overline{a'})) \in \to^{\mathrm{sym}}$ and since $Z_1 \subseteq Z_2$, we have $Z_1' \subseteq Z_2'$. □

Lemma B.8. *Let $\sigma = a_1^{\tau_1} a_2^{\tau_2} \ldots a_n^{\tau_n} \in \mathcal{T}(\mathbb{D}^{\mathbb{V}})$ be a signal and let $\mathcal{W} = (\mathcal{A}, \kappa)$ be a TSWA, where $\mathcal{A} = (\mathbb{V}, L, L_0, L_F, \mathbb{X}, \Delta, \Lambda)$. Let $\mathcal{S}$ and $\mathcal{S}^{\mathrm{sym}}$ be the WTTS and WSTTS of σ and $\mathcal{W}$, respectively. For any $(\ell, \nu, t, \overline{a}) \to (\ell', \nu', t', \overline{a'})$, there is a zone $Z' \in \mathcal{Z}(\mathbb{X} \amalg \{T\})$ satisfying the following.*

- *$((\ell, \{\nu_Z\}, \overline{a}), (\ell', Z', \overline{a'})) \in \to^{\mathrm{sym}}$, where $\nu_Z \in (\mathbb{R}_{\geq 0})^{\mathbb{X} \amalg \{T\}}$ is for any $x \in \mathbb{X}$, $\nu_Z(x) = \nu(x)$ and $\nu_Z(T) = t$.*
- *There exists $\nu_Z' \in Z'$ such that for any $x \in \mathbb{X}$ $\nu_Z'(x) = \nu'(x)$ and $\nu_Z'(T) = t'$.*
- *$W(((\ell, \nu, t, \overline{a}), (\ell', \nu', t', \overline{a'}))) = W^{\mathrm{sym}}(((\ell, \{\nu_Z\}, \overline{a}), (\ell', Z', \overline{a'})))$*

Proof. If $\overline{a'} = \varepsilon$, there is $(\ell, g, R, \ell') \in \Delta$ satisfying $\nu \models g$, $\nu' = [\nu]_R$, $t' = t$, $\overline{a} \neq \varepsilon$, and $\overline{a'} = \varepsilon$. Let Z' be $Z' = \{[\nu_Z]_R\}$. Since $\nu_Z \models g$, $\overline{a} \neq \varepsilon$, and $\overline{a'} = \varepsilon$, we have $((\ell, \{\nu_Z\}, \overline{a}), (\ell', Z', \overline{a'})) \in \to^{\mathrm{sym}}$. Since $\nu' = [\nu]_R$, for any $x \in \mathbb{X}$, we have $([\nu_Z]_R)(x) = \nu'(x)$. Since $t = t'$ and $T \notin R$, we have $([\nu_Z]_R)(T) = \nu_Z(T) = t = t'$. We also have $W(((\ell, \nu, t, \overline{a}), (\ell', \nu', t', \overline{a'}))) = \kappa(\Lambda(\ell), \overline{a}) = W^{\mathrm{sym}}(((\ell, \{\nu_Z\}, \overline{a}), (\ell', Z', \overline{a'})))$.

If $\overline{a'} \neq \varepsilon$, $\ell = \ell'$ and there is $\tau \in \mathbb{R}_{>0}$ satisfying $\nu' = \nu + \tau$, $t' = t + \tau$, and $\overline{a'} = \overline{a} \circ \sigma([t, t+\tau))$. Let M be either $M_{i,=} = \{\nu \mid \nu(T) = \sum_{j=0}^{i} \tau_j\}$ or $M_i = \{\nu \mid \sum_{j=0}^{i-1} \tau_j < \nu(T) < \sum_{j=0}^{i} \tau_j\}$ satisfying $t' \in M$, where $i \in \{1, 2, \ldots, n\}$. Let $Z' = \{\nu_Z + \tau \mid \tau \in \mathbb{R}_{>0}\} \cap M$. We have $((\ell, \{\nu_Z\}, \overline{a}), (\ell', Z', \overline{a'})) \in \rightarrow^{\mathrm{sym}}$. Since $\nu' = \nu + \tau$ and $t' \in M$, there exists $\nu'_Z \in Z'$ such that for any $x \in \mathbb{X}$ $\nu'_Z(x) = \nu'(x)$ and $\nu'_Z(T) = t'$. We also have $W(((\ell, \nu, t, \overline{a}), (\ell', \nu', t', \overline{a'}))) = e_{\otimes} = W^{\mathrm{sym}}(((\ell, \{\nu_Z\}, \overline{a}), (\ell', Z', \overline{a'})))$. □

Lemma B.9. *Let $\sigma \in \mathcal{T}(\mathbb{D}^{\mathbb{V}})$ be a signal and let $\mathcal{W} = (\mathcal{A}, \kappa)$ be a TSWA, where $\mathcal{A} = (\mathbb{V}, L, L_0, L_F, \mathbb{X}, \Delta, \Lambda)$. Let $\mathcal{S}$ and $\mathcal{S}^{\mathrm{sym}}$ be the WTTS and WSTTS of σ and $\mathcal{W}$, respectively. For any path $\pi = (\ell_0, \nu_0, t_0, \overline{a_0}), (\ell_1, \nu_1, t_1, \overline{a_1}), \ldots, (\ell_n, \nu_n, t_n, \overline{a_n})$ of $\mathcal{S}$, there is a path $\pi^{\mathrm{sym}} = (\ell_0, Z_0, \overline{a_0}), (\ell_1, Z_1, \overline{a_1}), \ldots, (\ell_n, Z_n, \overline{a_n})$ of $\mathcal{S}$, such that $Z_0 = \{\nu_{Z,0} \mid \forall x \in \mathbb{X}. \nu_{Z,0}(x) = \nu_0(x), \nu_{Z,0}(T) = t_0\}$ and for any $i \in \{1, 2, \ldots, n\}$, there exists $\nu_{Z,i} \in Z_i$ satisfying $\nu_{Z,i}(x) = \nu_i(x)$ for any $x \in \mathbb{X}$ and $\nu_{Z,i}(T) = t_i$.*

Proof. We prove the lemma by induction on n.

When $n = 1$, by Lemma B.8, for $Z_0 = \{\nu_{Z,0} \in (\mathbb{R}_{\geq 0})^{\mathbb{X} \amalg \{T\}} \mid \forall x \in \mathbb{X}. \nu_{Z,0}(x) = \nu_0(x), \nu_{Z,0}(T) = t_0\}$ there is a zone $Z_1 \in \mathcal{Z}(\mathbb{X} \amalg \{T\})$ satisfying:

- $((\ell_0, Z_0, \overline{a_0}), (\ell_1, Z_1, \overline{a_1})) \in \rightarrow^{\mathrm{sym}}$; and
- there exists $\nu_{Z,1} \in Z_1$ satisfying $\nu_{Z,1}(x) = \nu_1(x)$ for any $x \in \mathbb{X}$ and $\nu_{Z,1}(T) = t_1$.

When $n > 1$, by Lemma B.8, for $Z'_{n-1} = \{\nu_{Z,n-1} \in (\mathbb{R}_{\geq 0})^{\mathbb{X} \amalg \{T\}} \mid \forall x \in \mathbb{X}. \nu_{Z,n-1}(x) = \nu_{n-1}(x), \nu_{Z,n-1}(T) = t_{n-1}\}$ there is a zone $Z'_n \in \mathcal{Z}(\mathbb{X} \amalg \{T\})$ satisfying:

- $((\ell_{n-1}, Z'_{n-1}, \overline{a_{n-1}}), (\ell_n, Z'_n, \overline{a_n})) \in \rightarrow^{\mathrm{sym}}$; and
- there exists $\nu_{Z,n} \in Z'_n$ satisfying $\nu_{Z,n}(x) = \nu_n(x)$ for any $x \in \mathbb{X}$ and $\nu_{Z,n}(T) = t_n$.

By induction hypothesis, there is a path $(\ell_0, Z_0, \overline{a_0}), (\ell_1, Z_1, \overline{a_1}), \ldots, (\ell_{n-1}, Z_{n-1}, \overline{a_{n-1}})$ of $\mathcal{S}^{\mathrm{sym}}$, such that $Z_0 = \{\nu_{Z,0} \mid \forall x \in \mathbb{X}. \nu_{Z,0}(x) = \nu_0(x), \nu_{Z,0}(T) = t_0\}$ and for any $i \in \{1, 2, \ldots, n-1\}$, there exists $\nu_{Z,i} \in Z_i$ satisfying $\nu_{Z,i}(x) = \nu_i(x)$ for any $x \in \mathbb{X}$ and $\nu_{Z,i}(T) = t_i$. Since $Z'_{n-1} \subseteq Z_{n-1}$ and Lemma B.7, there exists $Z_n \in \mathcal{Z}(\mathbb{X} \amalg \{T\})$ satisfying $Z'_n \subseteq Z_n$ and $((\ell_{n-1}, Z_{n-1}, \overline{a_{n-1}}), (\ell_n, Z_n, \overline{a_n})) \in \rightarrow^{\mathrm{sym}}$. Therefore, $(\ell_0, Z_0, \overline{a_0}), (\ell_1, Z_1, \overline{a_1}), \ldots, (\ell_n, Z_n, \overline{a_n})$ is a path of $\mathcal{S}^{\mathrm{sym}}$, such that $Z_0 = \{\nu_{Z,0} \mid \forall x \in \mathbb{X}. \nu_{Z,0}(x) = \nu_0(x), \nu_{Z,0}(T) = t_0\}$ and for any $i \in \{1, 2, \ldots, n\}$, there exists $\nu_{Z,i} \in Z_i$ satisfying $\nu_{Z,i}(x) = \nu_i(x)$ for any $x \in \mathbb{X}$ and $\nu_{Z,i}(T) = t_i$. □

Lemma B.10. *Let $\sigma \in \mathcal{T}(\mathbb{D}^{\mathbb{V}})$ be a signal and let $\mathcal{W} = (\mathcal{A}, \kappa)$ be a TSWA, where $\mathcal{A} = (\mathbb{V}, L, L_0, L_F, \mathbb{X}, \Delta, \Lambda)$. Let $\mathcal{S}$ and $\mathcal{S}^{\mathrm{sym}}$ be the WTTS and WSTTS of σ and $\mathcal{W}$, respectively. For any path π of $\mathcal{S}$, there is a path π^{sym} of $\mathcal{S}^{\mathrm{sym}}$ satisfying $\mu(\pi) = \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}})$. Moreover, for any $\pi \in ARuns(\mathcal{S})$, there is $\pi^{\mathrm{sym}} \in ARuns(\mathcal{S}^{\mathrm{sym}})$ satisfying $\mu(\pi) = \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}})$.*

Proof. By Lemma B.9, for any path $\pi = (\ell_0, \nu_0, t_0, \overline{a_0}), (\ell_1, \nu_1, t_1, \overline{a_1}), \ldots, (\ell_n, \nu_n, t_n, \overline{a_n})$ of $\mathcal{S}$, there is a path $\pi^{\mathrm{sym}} = (\ell_0, Z_0, \overline{a_0}), (\ell_1, Z_1, \overline{a_1}), \ldots, (\ell_n, Z_n, \overline{a_n})$ of $\mathcal{S}^{\mathrm{sym}}$. For any $i \in$

$\{1, 2, \ldots, n\}$, we have

$$\begin{aligned}
&W((\ell_{i-1}, \nu_{i-1}, t_{i-1}, \overline{a_{i-1}}), (\ell_i, \nu_i, t_i, \overline{a_i})) \\
=&\begin{cases} \kappa(\Lambda(\ell_{i-1}, \overline{a_{i-1}})) & \text{if } \overline{a_i} = \varepsilon \\ e_\otimes & \text{if } \overline{a_i} \neq \varepsilon \end{cases} \\
=&W^{\mathrm{sym}}((\ell_{i-1}, Z_{i-1}, \overline{a_{i-1}}), (\ell_i, Z_i, \overline{a_i}))
\end{aligned}$$

Therefore, we have $\mu(\pi) = \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}})$.

When $\pi \in ARuns(\mathcal{S})$, we have $\ell_0 \in L_0$, $\nu_0 = \mathbf{0}_C$, $t_0 = 0$, $\overline{a_0} = \varepsilon$, $\ell_n \in L_F$, $t_n = |\sigma|$, and $\overline{a_n} = \varepsilon$. By Lemma B.9, we have $Z_0 = \{\mathbf{0}_{\mathbb{X} \amalg \{T\}}\}$ and there is $\nu_{Z,n} \in Z_n$ satisfying $\nu_{Z,n}(T) = t_n$. Therefore, $\pi^{\mathrm{sym}} \in ARuns(\mathcal{S}^{\mathrm{sym}})$ also holds. □

Theorem B.11. *Let $\sigma \in \mathcal{T}(\mathbb{D}^{\mathbb{V}})$ be a signal and let $\mathcal{W} = (\mathcal{A}, \kappa)$ be a TSWA, where $\mathcal{A} = (\mathbb{V}, L, L_0, L_F, \mathbb{X}, \Delta, \Lambda)$. Let $\mathcal{S}$ and $\mathcal{S}^{\mathrm{sym}}$ be the WTTS and WSTTS of σ and $\mathcal{W}$, respectively. If the semiring $\mathbb{S}$ is idempotent, we have $\alpha(\mathcal{S}) \preceq \alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}})$.*

Proof. By Lemma B.10, there is a mapping $f\colon ARuns(\mathcal{S}) \to ARuns(\mathcal{S}^{\mathrm{sym}})$ satisfying $W^{\mathrm{sym}}(f(\pi)) = W(\pi)$. We have

$$\begin{aligned}
&\alpha(\mathcal{S}) \oplus \alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}}) \\
=&\Big(\bigoplus_{\pi \in ARuns(\mathcal{S})} \mu(\pi)\Big) \oplus \Big(\bigoplus_{\pi^{\mathrm{sym}} \in ARuns(\mathcal{S}^{\mathrm{sym}})} \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}})\Big) \\
=&\Big(\bigoplus_{\pi^{\mathrm{sym}} \in f(ARuns(\mathcal{S}))} \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}})\Big) \oplus \Big(\bigoplus_{\pi^{\mathrm{sym}} \in ARuns(\mathcal{S}^{\mathrm{sym}})} \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}})\Big) \\
=&\Big(\bigoplus_{\pi^{\mathrm{sym}} \in ARuns(\mathcal{S}^{\mathrm{sym}})} \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}})\Big) = \alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}})
\end{aligned}$$

Therefore, we have $\alpha(\mathcal{S}) \preceq \alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}})$. □

Lemma B.12. *Let $\sigma \in \mathcal{T}(\mathbb{D}^{\mathbb{V}})$ be a signal and let $\mathcal{W} = (\mathcal{A}, \kappa)$ be a TSWA, where $\mathcal{A} = (\mathbb{V}, L, L_0, L_F, \mathbb{X}, \Delta, \Lambda)$. Let $\mathcal{S}$ and $\mathcal{S}^{\mathrm{sym}}$ be the WTTS and WSTTS of σ and $\mathcal{W}$, respectively. For any $((\ell, Z, \overline{a}), (\ell', Z', \overline{a'})) \in \to^{\mathrm{sym}}$, and $\nu'_Z \in Z'$, there is a clock valuation $\nu_Z \in Z$ satisfying the following.*

- $(\ell, \nu_Z{\downarrow_{\mathbb{X}}}, \nu_Z(T), \overline{a}) \to (\ell', \nu'_Z{\downarrow_{\mathbb{X}}}, \nu'_Z(T), \overline{a'})$, *where $\nu_Z{\downarrow_{\mathbb{X}}}, \nu'_Z{\downarrow_{\mathbb{X}}} \in (\mathbb{R}_{\geq 0})^{\mathbb{X}}$ are for any $x \in \mathbb{X}$, $\nu_Z{\downarrow_{\mathbb{X}}}(x) = \nu_Z(x)$ and $\nu'_Z{\downarrow_{\mathbb{X}}}(x) = \nu'_Z(x)$.*
- $W(((\ell, \nu_Z{\downarrow_{\mathbb{X}}}, \nu_Z(T), \overline{a}), (\ell', \nu'_Z{\downarrow_{\mathbb{X}}}, \nu'_Z(T), \overline{a'}))) = W^{\mathrm{sym}}(((\ell, Z, \overline{a}), (\ell', Z', \overline{a'})))$

Proof. If $\overline{a'} = \varepsilon$, we have $\overline{a} \neq \varepsilon$ and there is $(\ell, g, R, \ell') \in \Delta$ satisfying $Z' = \{[\nu]_R \mid \nu \in Z, \nu \models g\}$, which is nonempty. By definition of Z', there exists $\nu_Z \in Z$ such that $\nu'_Z = [\nu_Z]_R$. Since such ν_Z satisfies $\nu_Z{\downarrow_{\mathbb{X}}} \models g$ and $\nu_Z(T) = \nu'_Z(T)$, we have $(\ell, \nu_Z{\downarrow_{\mathbb{X}}}, \nu_Z(T), \overline{a}) \to (\ell', \nu'_Z{\downarrow_{\mathbb{X}}}, \nu'_Z(T), \overline{a'})$. We also have the following.

$$W(((\ell, \nu_Z{\downarrow_{\mathbb{X}}}, \nu_Z(T), \overline{a}), (\ell', \nu'_Z{\downarrow_{\mathbb{X}}}, \nu'_Z(T), \overline{a'}))) = \kappa(\Lambda(\ell), \overline{a}) = W^{\mathrm{sym}}(((\ell, Z, \overline{a}), (\ell', Z', \overline{a'})))$$

If $\overline{a'} \neq \varepsilon$, we have $\ell = \ell'$, $\overline{a'} = \overline{a} \circ \sigma([\nu_Z(T), \nu'_Z(T)))$, and for any $\nu' \in Z'$, there exists $\nu \in Z$ and $\tau \in \mathbb{R}_{>0}$ satisfying $\nu' = \nu + \tau$, where $\nu_Z \in Z$. Let $\nu_Z \in Z$ and $\tau \in \mathbb{R}_{>0}$ be such that $\nu'_Z = \nu_Z + \tau$. Because of

- $\ell = \ell'$,
- $\nu'_Z{\downarrow_{\mathbb{X}}} = \nu_Z{\downarrow_{\mathbb{X}}} + \tau$,
- $\nu'_Z(T) = \nu_Z(T) + \tau$, and
- $\overline{a'} = \overline{a} \circ \sigma([\nu_Z(T), \nu'_Z(T)))$,

we have $(\ell, \nu_Z{\downarrow_{\mathbb{X}}}, \nu_Z(T), \overline{a}) \to (\ell', \nu'_Z{\downarrow_{\mathbb{X}}}, \nu'_Z(T), \overline{a'})$. We also have

$$W\big(((\ell, \nu_Z{\downarrow_{\mathbb{X}}}, \nu_Z(T), \overline{a}), (\ell', \nu'_Z{\downarrow_{\mathbb{X}}}, \nu'_Z(T), \overline{a'}))\big) = e_\otimes = W^{\mathrm{sym}}(((\ell, Z, \overline{a}), (\ell', Z', \overline{a'}))) \ .$$

□

Lemma B.13. *Let $\sigma \in \mathcal{T}(\mathbb{D}^{\mathbb{V}})$ be a signal and let $\mathcal{W} = (\mathcal{A}, \kappa)$ be a TSWA, where $\mathcal{A} = (\mathbb{V}, L, L_0, L_F, \mathbb{X}, \Delta, \Lambda)$. Let $\mathcal{S}$ and $\mathcal{S}^{\mathrm{sym}}$ be the WTTS and WSTTS of σ and $\mathcal{W}$, respectively. For any path $\pi^{\mathrm{sym}} = (\ell_0, Z_0, \overline{a_0}), (\ell_1, Z_1, \overline{a_1}), \ldots, (\ell_n, Z_n, \overline{a_n})$ of $\mathcal{S}$ and for any $\nu_{Z,n} \in Z_n$, there is a path $\pi = (\ell_0, \nu_0, t_0, \overline{a_0}), (\ell_1, \nu_1, t_1, \overline{a_1}), \ldots, (\ell_n, \nu_n, t_n, \overline{a_n})$ of $\mathcal{S}$ such that we have $(\nu_{Z,n}){\downarrow_{\mathbb{X}}} = \nu_n$ and $\nu_{Z,n}(T) = t_n$, and for any $i \in \{0, 1, \ldots, n-1\}$, there exists $\nu_{Z,i} \in Z_i$ satisfying $(\nu_{Z,i}){\downarrow_{\mathbb{X}}} = \nu_i$ and $\nu_{Z,i}(T) = t_i$.*

Proof. We prove the lemma by induction on n. When $n = 1$, by Lemma B.12, for any $\nu_{Z,1} \in Z_1$, there is $\nu_{Z,0} \in Z_0$ satisfying $(\ell, (\nu_{Z,0}){\downarrow_{\mathbb{X}}}, \nu_{Z,0}(T), \overline{a}) \to (\ell', (\nu_{Z,1}){\downarrow_{\mathbb{X}}}, \nu_{Z,1}(T), \overline{a'})$.

When $n > 1$, by induction hypothesis, for any $\nu_{Z,n} \in Z_n$, there is a path

$$(\ell_1, \nu_1, t_1, \overline{a_1}), (\ell_2, \nu_2, t_2, \overline{a_2}), \ldots, (\ell_n, \nu_n, t_n, \overline{a_n})$$

of $\mathcal{S}$, such that we have $(\nu_{Z,n}){\downarrow_{\mathbb{X}}} = \nu_n$ and $\nu_{Z,n}(T) = t_n$, and for any $i \in \{1, 2, \ldots, n-1\}$, there exists $\nu_{Z,i} \in Z_i$ satisfying $(\nu_{Z,i}){\downarrow_{\mathbb{X}}} = \nu_i$ and $\nu_{Z,i}(T) = t_i$. By Lemma B.12, there is a clock valuation $\nu_{Z,0} \in Z_0$ satisfying $(\ell, (\nu_{Z,0}){\downarrow_{\mathbb{X}}}, \nu_{Z,0}(T), \overline{a}) \to (\ell', \nu_1, t_1, \overline{a'})$. Therefore, $(\ell_0, \nu_{Z,0}, \nu_{Z,0}(T), \overline{a_0}), (\ell_1, \nu_1, t_1, \overline{a_1}), \ldots, (\ell_n, \nu_n, t_n, \overline{a_n})$ is a path of $\mathcal{S}$, such that we have $(\nu_{Z,n}){\downarrow_{\mathbb{X}}} = \nu_n$ and $\nu_{Z,n}(T) = t_n$, and for any $i \in \{1, 2, \ldots, n-1\}$, there exists $\nu_{Z,i} \in Z_i$ satisfying $(\nu_{Z,i}){\downarrow_{\mathbb{X}}} = \nu_i$ and $\nu_{Z,i}(T) = t_i$. □

Lemma B.14. *Let $\sigma \in \mathcal{T}(\mathbb{D}^{\mathbb{V}})$ be a signal and let $\mathcal{W} = (\mathcal{A}, \kappa)$ be a TSWA, where $\mathcal{A} = (\mathbb{V}, L, L_0, L_F, \mathbb{X}, \Delta, \Lambda)$. Let $\mathcal{S}$ and $\mathcal{S}^{\mathrm{sym}}$ be the WTTS and WSTTS of σ and $\mathcal{W}$, respectively. For any path π^{sym} of $\mathcal{S}^{\mathrm{sym}}$, there is a path π of $\mathcal{S}$ satisfying $\mu(\pi) = \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}})$. Moreover, for any $\pi^{\mathrm{sym}} \in ARuns(\mathcal{S}^{\mathrm{sym}})$, there is $\pi \in ARuns(\mathcal{S})$ satisfying $\mu(\pi) = \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}})$.*

Proof. By Lemma B.13, for any path

$$\pi^{\mathrm{sym}} = (\ell_0, Z_0, \overline{a_0}), (\ell_1, Z_1, \overline{a_1}), \ldots, (\ell_n, Z_n, \overline{a_n})$$

of $\mathcal{S}^{\mathrm{sym}}$, there is a path

$$\pi = (\ell_0, \nu_0, t_0, \overline{a_0}), (\ell_1, \nu_1, t_1, \overline{a_1}), \ldots, (\ell_n, \nu_n, t_n, \overline{a_n})$$

of $\mathcal{S}$. For any $i \in \{1, 2, \ldots, n\}$, we have

$$\begin{aligned} & W\big((\ell_{i-1}, \nu_{i-1}, t_{i-1}, \overline{a_{i-1}}), (\ell_i, \nu_i, t_i, \overline{a_i})\big) \\ =& \begin{cases} \kappa(\Lambda(\ell_{i-1}, \overline{a_{i-1}})) & \text{if } \overline{a_i} = \varepsilon \\ e_\otimes & \text{if } \overline{a_i} \neq \varepsilon \end{cases} \\ =& W^{\mathrm{sym}}\big((\ell_{i-1}, Z_{i-1}, \overline{a_{i-1}}), (\ell_i, Z_i, \overline{a_i})\big) \end{aligned}$$

Therefore, we have $\mu(\pi) = \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}})$.

When $\pi^{\mathrm{sym}} \in ARuns(\mathcal{S}^{\mathrm{sym}})$, we have $\ell_0 \in L_0$, $Z_0 = \mathbf{0}_{\mathbb{X} \amalg \{T\}}$, $\overline{a_0} = \varepsilon$, $\ell_n \in L_F$, $\exists \nu_{Z,n} \in Z_n.\, \nu_{Z,n} = |\sigma|$, and $\overline{a_n} = \varepsilon$. By Lemma B.13, for $\nu_{Z,n} \in Z_n$ satisfying $\nu_{Z,n} = |\sigma|$, there is a path $\pi = (\ell_0, \nu_0, t_0, \overline{a_0}), (\ell_1, \nu_1, t_1, \overline{a_1}), \ldots, (\ell_n, \nu_n, t_n, \overline{a_n})$ of $\mathcal{S}$ such that we have $(\nu_{Z,n})\downarrow_{\mathbb{X}} = \nu_n$, $\nu_{Z,n}(T) = t_n$ $\nu_0 = \mathbf{0}_C$, $t_0 = 0$, and for any $i \in \{1, 2, \ldots, n-1\}$, there exists $\nu_{Z,i} \in Z_i$ satisfying $(\nu_{Z,i})\downarrow_{\mathbb{X}} = \nu_i$ and $\nu_{Z,i}(T) = t_i$. Therefore, $\pi \in ARuns(\mathcal{S})$ also holds. □

Theorem B.15. *Let $\sigma \in \mathcal{T}(\mathbb{D}^{\mathbb{V}})$ be a signal and let $\mathcal{W} = (\mathcal{A}, \kappa)$ be a TSWA, where $\mathcal{A} = (\mathbb{V}, L, L_0, L_F, \mathbb{X}, \Delta, \Lambda)$. Let $\mathcal{S}$ and $\mathcal{S}^{\mathrm{sym}}$ be the WTTS and WSTTS of σ and $\mathcal{W}$, respectively. If the semiring $\mathbb{S}$ is idempotent, we have $\alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}}) \preceq \alpha(\mathcal{S})$.*

Proof. By Lemma B.14, there is a mapping $f\colon ARuns(\mathcal{S}^{\mathrm{sym}}) \to ARuns(\mathcal{S})$ satisfying $W(f(\pi^{\mathrm{sym}})) = W^{\mathrm{sym}}(\pi^{\mathrm{sym}})$. We have

$$\begin{aligned}
&\alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}}) \oplus \alpha(\mathcal{S}) \\
=&\Big(\bigoplus_{\pi^{\mathrm{sym}} \in ARuns(\mathcal{S}^{\mathrm{sym}})} \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}})\Big) \oplus \Big(\bigoplus_{\pi \in ARuns(\mathcal{S})} \mu(\pi)\Big) \\
=&\Big(\bigoplus_{\pi \in f(ARuns(\mathcal{S}^{\mathrm{sym}}))} \mu(\pi)\Big) \oplus \Big(\bigoplus_{\pi \in ARuns(\mathcal{S})} \mu(\pi)\Big) \\
=&\Big(\bigoplus_{\pi \in ARuns(\mathcal{S})} \mu(\pi)\Big) = \alpha(\mathcal{S})
\end{aligned}$$

Therefore, we have $\alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}}) \preceq \alpha(\mathcal{S})$. □

Theorem 5.13. By Theorem B.11 and Theorem B.15, we have $\alpha(\mathcal{S}) \preceq \alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}})$. and $\alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}}) \preceq \alpha(\mathcal{S})$. Therefore, we have the following.

$$\alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}}) = \alpha(\mathcal{S}) \oplus \alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}}) = \alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}}) \oplus \alpha(\mathcal{S}) = \alpha(\mathcal{S})$$

□

B.3 Proof of Theorem 5.15

For locations ℓ, ℓ' of $\mathcal{A} = (\mathbb{V}, L, L_0, L_F, \mathbb{X}, \Delta, \Lambda)$, $Z, Z' \in \mathcal{Z}(\mathbb{X} \amalg \{T\})$, $\overline{a}, \overline{a'} \in (\mathbb{D}^{\mathbb{V}})^{\circledast}$, and the WSTTS $\mathcal{S}^{\mathrm{sym}}$ of a signal σ and $\mathcal{W} = (\mathcal{A}, \kappa)$, we denote the set of paths from $(\ell, Z, \overline{a})$ to $(\ell', Z', \overline{a'})$ of $\mathcal{S}^{\mathrm{sym}}$ as follows.

$$\begin{aligned}
&Paths(\mathcal{S}^{\mathrm{sym}}, \ell, Z, \overline{a}, \ell', Z', \overline{a'}) \\
=&\{\pi^{\mathrm{sym}} \mid \pi^{\mathrm{sym}} = (\ell, Z, \overline{a}), q_1^{\mathrm{sym}}, q_2^{\mathrm{sym}}, \ldots, q_n^{\mathrm{sym}}, (\ell', Z', \overline{a'}) \text{ is a path of } \mathcal{S}^{\mathrm{sym}} \}
\end{aligned}$$

By symbolic path value $\mu^{\mathrm{sym}}(\pi^{\mathrm{sym}})$, we can rewrite the increment function $incr(a, t)$ as follows, where $\mathcal{S}_{a,t}^{\mathrm{sym}}$ is the WSTTS of a^t and $\mathcal{W}$.

$$\begin{aligned}
incr(a,t)(w) = \Big\{(\ell', Z', \overline{a'}, s') \mid &\ell' \in L, Z' \in \mathcal{Z}(\mathbb{X} \amalg \{T\}), \forall \nu' \in Z'.\, \nu'(T) = t, \overline{a'} \in (\mathbb{D}^{\mathbb{V}})^{\circledast}, \\
&s' = \bigoplus_{\substack{(\ell, Z, \overline{a}, s) \in w \\ \pi^{\mathrm{sym}} \in Paths(\mathcal{S}_{a,t}^{\mathrm{sym}}, \ell, Z, \overline{a}, \ell', Z', \overline{a'})}} s \otimes \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}})\Big\}
\end{aligned}$$

Lemma B.16. *For a TSWA $\mathcal{W}$, a signal $\sigma = a_1^{\tau_1} a_2^{\tau_2} \dots a_n^{\tau_n}$, and $i \in \{1, 2, \dots, n\}$, we have the following.*

$$weight_i = \Big\{ (\ell_i, Z_i, \overline{a_i}, s_i) \mid \ell_i \in L, Z_i \in \mathcal{Z}(\mathbb{X} \amalg \{T\}), \overline{a_i} \in (\mathbb{D}^{\mathbb{V}})^{\circledast}, \forall \nu' \in Z'.\, \nu'(T) = \sum_{j=1}^{i} \tau_j$$
$$s_i = \bigoplus_{\substack{\ell_0 \in L_0 \\ \pi^{\mathrm{sym}} \in Paths(\mathcal{S}^{\mathrm{sym}}, \ell_0, \mathbf{0}_C \mathbb{X} \amalg \{T\}, \varepsilon, \ell_i, Z_i, \overline{a_i})}} \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}}) \Big\}$$

Proof. We prove by induction on i. If $i = 1$, we have the following.

$$\begin{aligned} weight_1 =& incr(a_1, \tau_1)(\{(\ell_0, \mathbf{0}_{\mathbb{X} \amalg \{T\}}, \varepsilon, e_{\otimes}) \mid \ell_0 \in L_0\}) \\ =& \Big\{ (\ell', Z', \overline{a'}, s') \mid \ell' \in L, Z' \in \mathcal{Z}(\mathbb{X} \amalg \{T\}), \overline{a'} \in (\mathbb{D}^{\mathbb{V}})^{\circledast}, \forall \nu' \in Z'.\, \nu'(T) = \tau_1 \\ & s' = \bigoplus_{\substack{\ell_0 \in L_0 \\ \pi^{\mathrm{sym}} \in Paths(\mathcal{S}^{\mathrm{sym}}_{a_1, \tau_1}, \ell_0, \mathbf{0}_{\mathbb{X} \amalg \{T\}}, \varepsilon, \ell', Z', \overline{a'})}} e_{\otimes} \otimes \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}}) \Big\} \\ =& \Big\{ (\ell', Z', \overline{a'}, s') \mid \ell' \in L, Z' \in \mathcal{Z}(\mathbb{X} \amalg \{T\}), \overline{a'} \in (\mathbb{D}^{\mathbb{V}})^{\circledast}, \forall \nu' \in Z'.\, \nu'(T) = \tau_1 \\ & s' = \bigoplus_{\substack{\ell_0 \in L_0 \\ \pi^{\mathrm{sym}} \in Paths(\mathcal{S}^{\mathrm{sym}}, \ell_0, \mathbf{0}_{\mathbb{X} \amalg \{T\}}, \varepsilon, \ell', Z', \overline{a'})}} e_{\otimes} \otimes \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}}) \Big\} \end{aligned}$$

If $i > 1$, by induction hypothesis, we have the following.

$$\begin{aligned} weight_i =& incr(a_i, \tau_i)(weight_{i-1}) \\ =& \Big\{ (\ell', Z', \overline{a'}, s') \mid \ell' \in L, Z' \in \mathcal{Z}(\mathbb{X} \amalg \{T\}), \overline{a'} \in (\mathbb{D}^{\mathbb{V}})^{\circledast}, \\ & s' = \bigoplus_{\substack{(\ell, Z, \overline{a}, s) \in weight_{i-1} \forall \nu' \in Z'.\, \nu'(T) = \sum_{j=1}^{i} \tau_j \\ \pi^{\mathrm{sym}} \in Paths(\mathcal{S}^{\mathrm{sym}}_{a_i, \tau_i}, \ell, Z, \overline{a}, \ell', Z', \overline{a'})}} s \otimes \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}}) \Big\} \\ =& \Big\{ (\ell', Z', \overline{a'}, s') \mid \ell' \in L, Z' \in \mathcal{Z}(\mathbb{X} \amalg \{T\}), \overline{a'} \in (\mathbb{D}^{\mathbb{V}})^{\circledast}, \forall \nu' \in Z'.\, \nu'(T) = \sum_{j=1}^{i} \tau_j, \\ & s' = \bigoplus_{\substack{\ell_0 \in L_0, \ell \in L, Z \in \mathcal{Z}(\mathbb{X} \amalg \{T\}), \overline{a} \in (\mathbb{D}^{\mathbb{V}})^{\circledast}, \\ \pi^{\mathrm{sym}} \in Paths(\mathcal{S}^{\mathrm{sym}}_{a_i, \tau_i}, \ell, Z, \overline{a}, \ell', Z', \overline{a'}) \\ \pi^{\mathrm{sym}\prime} \in Paths(\mathcal{S}^{\mathrm{sym}}, \ell_0, \mathbf{0}_{\mathbb{X} \amalg \{T\}}, \varepsilon, \ell, Z, \overline{a}), \\ \forall \nu \in Z.\, \nu(T) = \sum_{j=1}^{i-1} \tau_j}} \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}\prime}) \otimes \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}}) \Big\} \\ =& \Big\{ (\ell', Z', \overline{a'}, s') \mid \ell' \in L, Z' \in \mathcal{Z}(\mathbb{X} \amalg \{T\}), \overline{a'} \in (\mathbb{D}^{\mathbb{V}})^{\circledast}, \forall \nu' \in Z'.\, \nu'(T) = \sum_{j=1}^{i} \tau_j, \\ & s' = \bigoplus_{\substack{\ell_0 \in L_0 \\ \pi^{\mathrm{sym}} \in Paths(\mathcal{S}^{\mathrm{sym}}, \ell_0, \mathbf{0}_{\mathbb{X} \amalg \{T\}}, \varepsilon, \ell', Z', \overline{a'})}} \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}}) \Big\} \end{aligned}$$

□

Theorem 5.15. By Lemma B.16, we have the following.

$$\bigoplus_{\substack{(\ell,Z,\overline{a})\in Q_F^{\mathrm{sym}}\\(\ell,Z,\overline{a},s)\in weight_n}} s = \bigoplus_{\substack{(\ell,Z,\overline{a})\in Q_0^{\mathrm{sym}}\\(\ell',Z',\overline{a}')\in Q_F^{\mathrm{sym}}\\ \pi^{\mathrm{sym}}\in Paths(\mathcal{S}^{\mathrm{sym}},\ell,Z,\overline{a},\ell',Z',\overline{a}')}} \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}}) = \bigoplus_{\pi^{\mathrm{sym}}\in ARuns(\mathcal{S}^{\mathrm{sym}})} \mu^{\mathrm{sym}}(\pi^{\mathrm{sym}}) = \alpha^{\mathrm{sym}}(\mathcal{S}^{\mathrm{sym}})$$

□

APPENDIX C

Detailed example of Chapter 6

Example C.1. Let w and $\mathcal{A}$ be the ones in Example 6.16. In line 1 of Algorithm 10, we let $Conf_0 = \{(\ell_0, (40, 35))\}$. In line 3, we compute the time-bounded reachability analysis and the result is as follows.

$$
\begin{aligned}
Conf'_1 = \{&(\ell_0, v_1) \mid v_1(x_1) \in [115, 125], v_1(x_2) \in [115, 125]\} \\
&\cup \{(\ell_0, v_1) \mid -3v_1(x_1) + 11v_1(x_2) \geq 876, -2v_1(x_1) + 9v_1(x_2) \geq 789, \\
&\qquad v_1(x_2) \leq 431/3, v_1(x_1) \leq 499/3, v_1(x_2) \geq 115, v_1(x_1) \geq 115, \\
&\qquad 4v_1(x_1) - 7v_1(x_2) \geq -415\} \\
&\cup \{(\ell_1, v_1) \mid -3v_1(x_1) + 11v_1(x_2) \geq 876, -v_1(x_1) + 5v_1(x_2) \geq 456, \\
&\qquad -3v_1(x_2) \geq -431, -3v_1(x_1) \geq -499, v_1(x_2) \geq 115, v_1(x_1) \geq 115, \\
&\qquad 4v_1(x_1) - 7v_1(x_2) \geq -415\} \\
&\cup \{(\ell_2, v_1) \mid -18v_1(x_1) + 15v_1(x_2) \geq -415, -v_1(x_1) + 2v_1(x_2) \geq 115, \\
&\qquad 4v_1(x_1) - 7v_1(x_2) \geq -415, v_1(x_1) \geq 115\} \\
&\cup \{(\ell_3, v_1) \mid v_1(x_1) \in [115, 455/3], -3v_1(x_1) + 11v_1(x_2) \geq 920, \\
&\qquad v_1(x_2) \leq 415/3, 4v_1(x_1) - 7v_1(x_2) \geq -415\}
\end{aligned}
$$

In line 4, we require $x_1 = 123$ and $x_2 = 117$, and we have $Conf_1 = \{(\ell_0, (123, 117)), (\ell_1, (123, 117))\}$. Since $\ell_0 \notin L_F$ and $\ell_1 \notin L_F$, we have $Result_1 = \bot$.

After incrementing i in line 2, in line 3, we again compute the time-bounded reachability

analysis and the result is as follows.

$$\begin{aligned}
Conf'_2 = \{(\ell_0, v_2) \mid\ & v_2(x_1) \in [198, 253], v_2(x_2) \in [197, 227], \\
& -2v_2(x_1) + 9v_2(x_2) \geq 1357, 4v_2(x_1) - 7v_2(x_2) \geq -657\} \\
\cup \{(\ell_1, v_2) \mid\ & v_2(x_1) \in [233, 253], v_2(x_2) \in [207, 227]\} \\
\cup \{(\ell_1, v_2) \mid\ & -3v_2(x_1) + 11v_2(x_2) \geq 1540, -v_2(x_1) + 5v_2(x_2) \geq 784, \\
& -3v_2(x_2) \geq -673, -3v_2(x_1) \geq -737, v_2(x_2) \geq 197, \\
& v_2(x_1) \geq 198, 4v_2(x_1) - 7v_2(x_2) \geq -657\} \\
\cup \{(\ell_2, v_2) \mid\ & -6v_2(x_1) + 5v_2(x_2) \geq -219, -v_2(x_1) + 2v_2(x_2) \geq 198, \\
& 4v_2(x_1) - 7v_2(x_2) \geq -657, v_2(x_1) \geq 198\} \\
\cup \{(\ell_3, v_2) \mid\ & v_2(x_1) \in [198, 231], v_2(x_2) \leq 219, -3v_2(x_1) + 11v_2(x_2) \geq 1584, \\
& 4v_2(x_1) - 7v_2(x_2) \geq -657\}
\end{aligned}$$

In line 4, we require $x_1 = 203$ and $x_2 = 201$. This time, we have $Conf_1 = \{(\ell, (203, 201)) \mid \ell \in L\}$. Since $L \cap L_F \neq \emptyset$, we have $Result_2 = \top$. ◇

APPENDIX D

Model-Bounded Monitoring with Partial Observations

Here, we briefly show that it is straightforward to generalize model-bounded monitoring in Chapter 6 to monitor the signals with partial observations. We note that by using partial observations, we can also use parameters both in the model and the specification because a parameter is equivalent to a variable x with flow $\dot{x} = 0$ and never observed in the log w. Moreover, we can conduct timed pattern matching by constructing the *matching automaton* in [BFN+18] and Section 5.6 with parameters t and t'.

D.1 Partial timed quantitative word

We introduce partial timed quantitative word to model the sampling with *partial* observation. A partial timed quantitative word is a timed quantitative word where the valuation function may be partial. For partial timed quantitative words, we use the same notation as timed quantitative words.

Definition D.1 (partial timed quantitative words)**.** A *partial timed quantitative word* w is a sequence $(u_1, \tau_1), (u_2, \tau_2), \dots, (u_m, \tau_m)$ of pairs (u_i, τ_i) of a partial valuation $u_i \colon \mathbb{X} \nrightarrow \mathbb{R}$ and a timestamp $\tau_i \in \mathbb{R}_{\geq 0}$ satisfying $\tau_i \leq \tau_{i+1}$ for each $i \in \{1, 2, \dots, m-1\}$. For partial valuations $u, u' \colon \mathbb{X} \nrightarrow \mathbb{R}$, we denote $u \sqsubseteq u'$ if for any $x \in \mathrm{dom}(u)$, $u(x) = u'(x)$ holds. ◇

D.2 Partial monitored language

We extend the monitored language in Definition 6.10 to the set of partial timed quantitative words. For partial monitored languages, we use the same notation as monitored languages.

Definition D.2 (partial monitored language $\mathcal{L}_{\mathrm{mon}}(\mathcal{A})$)**.** Let $\rho = s_0 \to_1 s_1 \to_2 \cdots \to_n s_n$ be a run of an LHA $\mathcal{A}$ (Definition 6.6), and $w = (u_1, \tau_1), (u_2, \tau_2), \dots, (u_m, \tau_m)$ be a partial timed quantitative word. We say w is *associated* to ρ if, for each $j \in \{1, 2, \dots, m\}$, we have either of the following two. Here ℓ_i, v_i are so that $s_i = (\ell_i, v_i)$ for each $i \in \{0, 1, \dots, n\}$.

1. There exists $i \in \{0, 1, 2, \ldots, n\}$ such that $\mathrm{Dur}(\rho[i]) = \tau_j$ and $u_j \sqsubseteq v_i$; or
2. There exists $i \in \{0, 1, 2, \ldots, n-1\}$ such that $\mathrm{Dur}(\rho[i]) < \tau_j < \mathrm{Dur}(\rho[i+1])$ and for any $x \in \mathbb{X}$, $u_j(x) = v_i(x) + (\tau_j - \mathrm{Dur}(\rho[i]))f_i(\dot{x})$ holds if $u_j(x)$ is defined, where $\to_i = \overset{d_i, f_i}{\mapsto}$.

Finally, the *partial monitored language* $\mathcal{L}_{\mathrm{mon}}(\mathcal{A})$ of an LHA $\mathcal{A}$ is the set of partial timed quantitative words associated with some accepting run of $\mathcal{A}$. ◇

D.3 Membership constraint problem for partial monitored languages

Then, we formulate the monitoring problem of a partial timed quantitative word w against a specification in an LHA $\mathcal{A}$. In the monitoring problem, the total valuations $v \colon \mathbb{X} \to \mathbb{R}$ compatible with the observation are synthesized as well as the indices i satisfying $w[i] \in \mathcal{L}_{\mathrm{mon}}(\mathcal{A})$ are returned.

The $\mathcal{L}_{\mathrm{mon}}$ membership constraint problem:
INPUT: An LHA $\mathcal{A}$ and a partial timed quantitative word $w = (u_1, \tau_1), \ldots, (u_m, \tau_m)$.
PROBLEM: Returns (constraints that express) the set $\mathsf{C}(w, \mathcal{A})$, consisting of all the pairs (i, v) of an index $i \in \{1, 2, \ldots, m\}$ and a total valuation $v \colon \mathbb{X} \to \mathbb{R}$ satisfying $u_i \sqsubseteq v$ and $(u_1, \tau_1), (u_2, \tau_2), \ldots, (u_{i-1}, \tau_{i-1}), (v, \tau_i) \in \mathcal{L}_{\mathrm{mon}}(\mathcal{A})$.

D.4 Algorithms for the membership constraint problem

Here we show how to modify the algorithms in Sections 6.6 and 6.7 to solve the membership constraint problem.

D.4.1 Algorithm I in Section 6.6

We can directly reuse the algorithm in Section 6.6 because of the following reasons.

- Although the valuation in the partial timed quantitative word w can be partial, the construction of the LHA $\mathcal{A}_w$ is the same; we simply convert each partial valuation to a guard.
- The construction of the synchronized product $\mathcal{A}_w \mid\mid \mathcal{A}$ is the same.
- We need to synthesize the constraint for the reachability rather than deciding the reachability. We can use PHAVerLite to synthesize such a constraint.

D.4.2 Algorithm II in Section 6.7

Since Algorithm 10 does not conduct the constraint synthesis, we need the following slight modification.

- The constraint $v = u_i$ in line 4 of Algorithm 10 is replaced with $u_i \sqsubseteq v$.
- We return the valuation rather than the Boolean acceptance. Namely, line 5 of Algorithm 10 becomes $\mathit{Result}_i \leftarrow \{v \mid \exists (\ell, v) \in \mathit{Conf}_i. \ell \in L_F\}$.

Bibliography

[ABD18] Eugene Asarin, Nicolas Basset, and Aldric Degorre. Distance on timed words and applications. In David N. Jansen and Pavithra Prabhakar, editors, *Proceedings of the 16th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2018)*, volume 11022 of *Lecture Notes in Computer Science*, pages 199–214. Springer, 2018.

[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, October 2009.

[ACM97] Eugene Asarin, Paul Caspi, and Oded Maler. A Kleene theorem for timed automata. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 - July 2, 1997*, pages 160–171. IEEE Computer Society, 1997.

[ACM02] Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *J. ACM*, 49(2):172–206, 2002.

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

[ADMN11] Eugene Asarin, Alexandre Donzé, Oded Maler, and Dejan Nickovic. Parametric identification of temporal properties. In *Proceedings of the Second International Conference on Runtime Verification (RV 2011)*, volume 7186 of *Lecture Notes in Computer Science*, pages 147–160. Springer, 2011.

[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, *Proceedings of the 18th International Symposium on Formal Methods (FM 2012)*, volume 7436 of *Lecture Notes in Computer Science*, pages 33–36. Springer, 8 2012.

[AFMS19] Étienne André, Laurent Fribourg, Jean-Marc Mota, and Romain Soulat. Verification of an industrial asynchronous leader election algorithm using abstractions and parametric model checking. In Constantin Enea and Ruzica Piskac, editors, *Verification, Model Checking, and Abstract Interpretation - 20th International Conference, VMCAI 2019, Cascais, Portugal, January 13-15, 2019, Proceedings*, volume 11388 of *Lecture Notes in Computer Science*, pages 409–424. Springer, 2019.

[AH15] Takumi Akazaki and Ichiro Hasuo. Time robustness in MTL and expressivity in hybrid system falsification. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II*, volume 9207 of *Lecture Notes in Computer Science*, pages 356–374. Springer, 2015.

[AHH96] Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. Software Eng.*, 22(3):181–201, 1996.

[AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the 25th annual ACM symposium on Theory of computing (STOC 1993)*, pages 592–601, New York, NY, USA, 1993. ACM.

[AHW18] Étienne André, Ichiro Hasuo, and Masaki Waga. Offline timed pattern matching under uncertainty. In Anthony Widjaja Lin and Jun Sun, editors, *Proceedings of the 23rd International Conference on Engineering of Complex Computer Systems (ICECCS 2018)*, pages 10–20. IEEE CPS, 2018.

[AKV98] Rajeev Alur, Robert P. Kurshan, and Mahesh Viswanathan. Membership questions for timed and hybrid automata. In *RTSS*, pages 254–263. IEEE Computer Society, 1998.

[AL17] Étienne André and Didier Lime. Liveness in L/U-parametric timed automata. In Alex Legay and Klaus Schneider, editors, *Proceedings of the 17th International Conference on Application of Concurrency to System Design (ACSD 2017)*, pages 9–18. IEEE, 2017.

[ALFS11] Yashwanth Annpureddy, Che Liu, Georgios E. Fainekos, and Sriram Sankaranarayanan. S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In Parosh Aziz Abdulla and K. Rustan M. Leino, editors, *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2011), Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS 2011)*, volume 6605 of *Lecture Notes in Computer Science*, pages 254–257. Springer, 2011.

[And19] Étienne André. What's decidable about parametric timed automata? *International Journal on Software Tools for Technology Transfer*, 21(2):203–219, 4 2019.

[APM19] Houssam Abbas, Yash Vardhan Pant, and Rahul Mangharam. Temporal logic robustness for general signal classes. In Necmiye Ozay and Pavithra Prabhakar, editors, *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019*, pages 45–56. ACM, 2019.

[BBFL03] Gerd Behrmann, Patricia Bouyer, Emmanuel Fleury, and Kim Guldstrand Larsen. Static guard analysis in timed automata verification. In Hubert Garavel and John Hatcliff, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 9th International Conference, TACAS 2003, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw,*

Poland, April 7-11, 2003, Proceedings, volume 2619 of *Lecture Notes in Computer Science*, pages 254–277. Springer, 2003.

[BBLP06] Gerd Behrmann, Patricia Bouyer, Kim Guldstrand Larsen, and Radek Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. *STTT*, 8(3):204–215, 2006.

[BDD+18] Ezio Bartocci, Jyotirmoy V. Deshmukh, Alexandre Donzé, Georgios E. Fainekos, Oded Maler, Dejan Nickovic, and Sriram Sankaranarayanan. Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications. In Ezio Bartocci and Yliès Falcone, editors, *Lectures on Runtime Verification – Introductory and Advanced Topics*, volume 10457 of *Lecture Notes in Computer Science*, pages 135–175. Springer, 2018.

[BDG+11] Thomas Brihaye, Laurent Doyen, Gilles Geeraerts, Joël Ouaknine, Jean-François Raskin, and James Worrell. On reachability for hybrid automata over bounded time. In *ICALP (2)*, volume 6756 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 2011.

[BDSV14] Lubos Brim, Petr Dluhos, David Safránek, and Tomas Vejpustek. STL*: Extending signal temporal logic with signal-value freezing operator. *Information and Computation*, 236:52–67, 2014.

[BER94] Ahmed Bouajjani, Rachid Echahed, and Riadh Robbana. On the automatic verification of systems with continuous variables and unbounded discrete data structures. In Panos J. Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems II*, volume 999 of *Lecture Notes in Computer Science*, pages 64–85. Springer, 1994.

[BF18] Ezio Bartocci and Yliès Falcone, editors. *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457 of *Lecture Notes in Computer Science*. Springer, 2018.

[BFFR18] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. Introduction to runtime verification. In Bartocci and Falcone [BF18], pages 1–33.

[BFH+12] Howard Barringer, Yliès Falcone, Klaus Havelund, Giles Reger, and David E. Rydeheard. Quantified event automata: Towards expressive and efficient runtime monitors. In Dimitra Giannakopoulou and Dominique Méry, editors, *FM*, volume 7436 of *Lecture Notes in Computer Science*, pages 68–84. Springer, 2012.

[BFM18] Alexey Bakhirkin, Thomas Ferrère, and Oded Maler. Efficient parametric identification for STL. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week) (HSCC 2018)*, pages 177–186. ACM, 2018.

[BFMU17] Alexey Bakhirkin, Thomas Ferrère, Oded Maler, and Dogan Ulus. On the quantitative semantics of regular expressions over real-valued signals. In Alessandro Abate and Gilles Geeraerts, editors, *Proceedings of the 15th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2017)*,

volume 10419 of *Lecture Notes in Computer Science*, pages 189–206. Springer, 2017.

[BFN+18] Alexey Bakhirkin, Thomas Ferrère, Dejan Nickovic, Oded Maler, and Eugene Asarin. Online timed pattern matching using automata. In David N. Jansen and Prabhakar Pavithra, editors, *Proceedings of the 16th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2018)*, volume 11022 of *Lecture Notes in Computer Science*, pages 215–232. Springer, 2018.

[BGF18] Reza Babaee, Arie Gurfinkel, and Sebastian Fischmeister. *P*revent : A predictive run-time verification framework using statistical learning. In *SEFM*, volume 10886 of *Lecture Notes in Computer Science*, pages 205–220. Springer, 2018.

[BHZ08] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[BKMZ15a] David A. Basin, Felix Klaedtke, Srdjan Marinovic, and Eugen Zalinescu. Monitoring of temporal first-order properties with aggregations. *Formal Methods in System Design*, 46(3):262–285, 2015.

[BKMZ15b] David A. Basin, Felix Klaedtke, Samuel Müller, and Eugen Zalinescu. Monitoring metric first-order temporal properties. *Journal of the ACM*, 62(2):15:1–15:45, 2015.

[BKT17] David A. Basin, Srdjan Krstic, and Dmitriy Traytel. AERIAL: almost event-rate independent algorithms for monitoring metric regular properties. In Giles Reger and Klaus Havelund, editors, *RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools, September 15, 2017, Seattle, WA, USA*, volume 3 of *Kalpa Publications in Computing*, pages 29–36. EasyChair, 2017.

[BKZ17] David A. Basin, Felix Klaedtke, and Eugen Zalinescu. The MonPoly monitoring tool. In Giles Reger and Klaus Havelund, editors, *RV-CuBES*, volume 3 of *Kalpa Publications in Computing*, pages 19–28. EasyChair, 2017.

[BL09] Laura Bozzelli and Salvatore La Torre. Decision problems for lower/upper bound parametric timed automata. *Formal Methods in System Design*, 35(2):121–151, 2009.

[BLS11] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, 2011.

[BRS19] Lei Bu, Rajarshi Ray, and Stefan Schupp. ARCH-COMP19 category report: Bounded model checking of hybrid systems with piecewise constant dynamics. In *ARCH@CPSIoTWeek*, volume 61 of *EPiC Series in Computing*, pages 120–128. EasyChair, 2019.

[BY03a] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets, Advances in Petri Nets [This tutorial volume originates from the 4th Advanced Course on Petri Nets, ACPN 2003]*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003.

[BY03b] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets, Advances in Petri Nets [This tutorial volume originates from the 4th Advanced Course on Petri Nets, ACPN 2003, held in Eichstätt, Germany in September 2003. In addition to lectures given at ACPN 2003, additional chapters have been commissioned]*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003.

[BZ19] Anna Becchi and Enea Zaffanella. Revisiting polyhedral analysis for hybrid systems. In *SAS*, volume 11822 of *Lecture Notes in Computer Science*, pages 183–202. Springer, 2019.

[CÁF11] Xin Chen, Erika Ábrahám, and Goran Frehse. Efficient bounded reachability computation for rectangular automata. In *RP*, volume 6945 of *Lecture Notes in Computer Science*, pages 139–152. Springer, 2011.

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, 2009.

[CHO16] Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Quantitative monitor automata. In Xavier Rival, editor, *Static Analysis - 23rd International Symposium, SAS 2016, Edinburgh, UK, September 8-10, 2016, Proceedings*, volume 9837 of *Lecture Notes in Computer Science*, pages 23–38. Springer, 2016.

[cru] tprasadtp/cruise-control-simulink: Simulink model for Cruise control system of a car with dynamic road conditions. https://github.com/tprasadtp/cruise-control-simulink.

[Dan03] Zhe Dang. Pushdown timed automata: a binary reachability characterization and safety verification. *Theoretical Computer Science*, 302(1-3):93–121, 2003.

[DDG+15] Jyotirmoy V. Deshmukh, Alexandre Donzé, Shromona Ghosh, Xiaoqing Jin, Garvit Juniwal, and Sanjit A. Seshia. Robust online monitoring of signal temporal logic. In Ezio Bartocci and Rupak Majumdar, editors, *Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings*, volume 9333 of *Lecture Notes in Computer Science*, pages 55–70. Springer, 2015.

[DFM13] Alexandre Donzé, Thomas Ferrère, and Oded Maler. Efficient robust monitoring for STL. In Natasha Sharygina and Helmut Veith, editors, *Proceedings of the 25th International Conference on Computer Aided Verification (CAV 2013)*, volume 8044 of *Lecture Notes in Computer Science*, pages 264–279. Springer, 2013.

[Dil89] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems 1989*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1989.

[dlH16] Colin de la Higuera. Learning grammars and automata with queries. *Topics in Grammatical Inference*, page 47–71, 2016.

[DM10] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In Krishnendu Chatterjee and Thomas A. Henzinger, editors, *Proceedings of the 8th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2010)*, volume 6246 of *Lecture Notes in Computer Science*, pages 92–106. Springer, 2010.

[DMB+12] Alexandre Donzé, Oded Maler, Ezio Bartocci, Dejan Nickovic, Radu Grosu, and Scott A. Smolka. On temporal logic and signal processing. In Supratik Chakraborty and Madhavan Mukund, editors, *Automated Technology for Verification and Analysis - 10th International Symposium, ATVA 2012, Thiruvananthapuram, India, October 3-6, 2012. Proceedings*, volume 7561 of *Lecture Notes in Computer Science*, pages 92–106. Springer, 2012.

[DMP17] Jyotirmoy V. Deshmukh, Rupak Majumdar, and Vinayak S. Prabhu. Quantifying conformance using the Skorokhod metric. *Formal Methods in System Design*, 50(2-3):168–206, 2017.

[DN09] Thao Dang and Tarik Nahhal. Coverage-guided test generation for continuous and hybrid systems. *Formal Methods in System Design*, 34(2):183–213, 2009.

[Don10] Alexandre Donzé. Breach, A toolbox for verification and parameter synthesis of hybrid systems. In Tayssir Touili, Byron Cook, and Paul B. Jackson, editors, *Proceedings of the 22nd International Conference on Computer Aided Verification (CAV 2010)*, volume 6174 of *Lecture Notes in Computer Science*, pages 167–170. Springer, 2010.

[FAA+19] Goran Frehse, Alessandro Abate, Dieky Adzkiya, Anna Becchi, Lei Bu, Alessandro Cimatti, Mirco Giacobbe, Alberto Griggio, Sergio Mover, Muhammad Syifa'ul Mufid, Idriss Riouak, Stefano Tonetta, and Enea Zaffanella. ARCH-COMP19 category report: Hybrid systems with piecewise constant dynamics. In *ARCH@CPSIoTWeek*, volume 61 of *EPiC Series in Computing*, pages 1–13. EasyChair, 2019.

[FdSC+17] Davide Fauri, Daniel Ricardo dos Santos, Elisa Costante, Jerry den Hartog, Sandro Etalle, and Stefano Tonetta. From system specification to anomaly detection (and back). In Bhavani M. Thuraisingham, Rakesh B. Bobba, and Awais Rashid, editors, *Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy, Dallas, TX, USA, November 3, 2017*, pages 13–24. ACM, 2017.

[FJS07] Frantisek Franek, Christopher G. Jennings, and William F. Smyth. A simple fast hybrid pattern-matching algorithm. *Journal of Discrete Algorithms*, 5(4):682–695, 2007.

[FP09] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.

[FR08] François Fages and Aurélien Rizk. On temporal logic constraint solving for analyzing numerical data time series. *Theoretical Computer Science*, 408(1):55–65, 2008.

[Fre08] Goran Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. *International Journal on Software Tools for Technology Transfer*, 10(3):263–279, 2008.

[GBMP13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (IoT): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.*, 29(7):1645–1660, 2013.

[HAF14] Bardh Hoxha, Houssam Abbas, and Georgios E. Fainekos. Benchmarks for temporal logic requirements for automotive systems. In Goran Frehse and Matthias Althoff, editors, *Proceedings of the 1st and 2nd International Workshops on Applied veRification for Continuous and Hybrid Systems (ARCH@CPSWeek 2014 / ARCH@CPSWeek 2015)*, volume 34 of *EPiC Series in Computing*, pages 25–30. EasyChair, 2014.

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 278–292. IEEE Computer Society, 1996.

[HMP91] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. Timed transition systems. In *REX*, volume 600 of *Lecture Notes in Computer Science*, pages 226–251. Springer, 1991.

[HOW14] Hsi-Ming Ho, Joël Ouaknine, and James Worrell. Online monitoring of metric temporal logic. In Borzoo Bonakdarpour and Scott A. Smolka, editors, *Proceedings of the 5th International Conference on Runtime Verification (RV 2014)*, volume 8734 of *Lecture Notes in Computer Science*, pages 178–192. Springer, 2014.

[HP18] Klaus Havelund and Doron Peled. Efficient runtime verification of first-order temporal properties. In María-del-Mar Gallardo and Pedro Merino, editors, *Model Checking Software - 25th International Symposium, SPIN 2018, Malaga, Spain, June 20-22, 2018, Proceedings*, volume 10869 of *Lecture Notes in Computer Science*, pages 26–47. Springer, 2018.

[HPR94] Nicolas Halbwachs, Yann-Éric Proy, and Pascal Raymond. Verification of linear hybrid systems by means of convex approximations. In *SAS*, volume 864 of *Lecture Notes in Computer Science*, pages 223–237. Springer, 1994.

[HPU17] Klaus Havelund, Doron Peled, and Dogan Ulus. First order temporal logic monitoring with BDDs. In Daryl Stewart and Georg Weissenbacher, editors, *Proceedings of the 2017 Formal Methods in Computer Aided Design (FMCAD 2017)*, pages 116–123. IEEE, 2017.

[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.

[HSW10] Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Efficient emptiness check for timed büchi automata. In Tayssir Touili, Byron Cook, and Paul B. Jackson, editors, *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, volume 6174 of *Lecture Notes in Computer Science*, pages 148–161. Springer, 2010.

[JBG+18a] Stefan Jaksic, Ezio Bartocci, Radu Grosu, Thang Nguyen, and Dejan Nickovic. Quantitative monitoring of STL with edit distance. *Formal Methods in System Design*, 53(1):83–112, 2018.

[JBG+18b] Stefan Jakšić, Ezio Bartocci, Radu Grosu, Thang Nguyen, and Dejan Ničković. Quantitative monitoring of STL with edit distance. *Formal Methods in System Design*, 53(1):83–112, 2018.

[JBGN18] Stefan Jaksic, Ezio Bartocci, Radu Grosu, and Dejan Nickovic. An algebraic framework for runtime verification. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 37(11):2233–2243, 2018.

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

[JTS+17] Susmit Jha, Ashish Tiwari, Sanjit A. Seshia, Tuhin Sahai, and Natarajan Shankar. Telex: Passive STL learning using only positive examples. In Shuvendu K. Lahiri and Giles Reger, editors, *Proceedings of the 17th International Conference on Runtime Verification (RV 2017)*, volume 10548 of *Lecture Notes in Computer Science*, pages 208–224. Springer, 2017.

[KCDK15] Aaron Kane, Omar Chowdhury, Anupam Datta, and Philip Koopman. A case study on runtime monitoring of an autonomous research vehicle (ARV) system. In Ezio Bartocci and Rupak Majumdar, editors, *Proceedings of the 6th International Conference on Runtime Verification (RV 2015)*, volume 9333 of *Lecture Notes in Computer Science*, pages 102–117. Springer, 2015.

[KDM+17] Maryam Kamali, Louise A. Dennis, Owen McAree, Michael Fisher, and Sandor M. Veres. Formal verification of autonomous vehicle platooning. *Sci. Comput. Program.*, 148:88–106, 2017.

[KJD+16] James Kapinski, Xiaoqing Jin, Jyotirmoy Deshmukh, Alexandre Donze, Tomoya Yamaguchi, Hisahiro Ito, Tomoyuki Kaga, Shunsuke Kobuna, and Sanjit Seshia. St-lib: a library for specifying and classifying model behaviors. Technical report, SAE Technical Paper, 2016.

[KJP77] Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[Koy90] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[KS08] Ajay D. Kshemkalyani and Mukesh Singhal. Distributed computing. 2008.

[LG16] Damián López and Pedro García. On the inference of finite state automata from positive and negative data. *Topics in Grammatical Inference*, page 73–112, 2016.

[LLN18] Kim G. Larsen, Florian Lorber, and Brian Nielsen. 20 years of UPPAAL enabled industrial model-based validation and beyond. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice - 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part IV*, volume 11247 of *Lecture Notes in Computer Science*, pages 212–229. Springer, 2018.

[LMNS05] Kim Guldstrand Larsen, Marius Mikucionis, Brian Nielsen, and Arne Skou. Testing real-time embedded software using UPPAAL-TRON: an industrial case study. In Wayne H. Wolf, editor, *EMSOFT 2005, September 18-22, 2005, Jersey City, NJ, USA, 5th ACM International Conference On Embedded Software, Proceedings*, pages 299–306. ACM, 2005.

[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, *Proceedings of the Third International Workshop on Hybrid Systems: Computation and Control (HSCC 2000)*, volume 1790 of *Lecture Notes in Computer Science*, pages 296–309. Springer, 2000.

[MLR+10] Marius Mikucionis, Kim Guldstrand Larsen, Jacob Illum Rasmussen, Brian Nielsen, Arne Skou, Steen Ulrik Palm, Jan Storbank Pedersen, and Poul Hougaard. Schedulability analysis using uppaal: Herschel-planck case study. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification, and Validation - 4th International Symposium on Leveraging Applications, ISoLA 2010, Heraklion, Crete, Greece, October 18-21, 2010, Proceedings, Part II*, volume 6416 of *Lecture Notes in Computer Science*, pages 175–190. Springer, 2010.

[MN04] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In Yassine Lakhnech and Sergio Yovine, editors, *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS 2004 and Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2004, Grenoble, France, September 22-24, 2004, Proceedings*, volume 3253 of *Lecture Notes in Computer Science*, pages 152–166. Springer, 2004.

[Moh09] Mehryar Mohri. *Weighted Automata Algorithms*, pages 213–254. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[NBC+18] Laura Nenzi, Luca Bortolussi, Vincenzo Ciancia, Michele Loreti, and Mieke Massink. Qualitative and quantitative monitoring of spatio-temporal properties with SSTL. *Log. Methods Comput. Sci.*, 14(4), 2018.

[NLM+18] Dejan Nickovic, Olivier Lebeltel, Oded Maler, Thomas Ferrère, and Dogan Ulus. AMT 2.0: Qualitative and quantitative trace analysis with extended signal temporal logic. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part II*, volume 10806 of *Lecture Notes in Computer Science*, pages 303–319. Springer, 2018.

[PJT+17] Srinivas Pinisetty, Thierry Jéron, Stavros Tripakis, Yliès Falcone, Hervé Marchand, and Viorel Preoteasa. Predictive runtime verification of timed properties. *Journal of Systems and Software*, 132:353–365, 2017.

[QCG+09] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[Qua15] Karin Quaas. Verification for timed automata extended with discrete data structure. *Logical Methods in Computer Science*, 11(3), 2015.

[RCR15] Giles Reger, Helena Cuenca Cruz, and David E. Rydeheard. MarQ: Monitoring at runtime with QEA. In Christel Baier and Cesare Tinelli, editors, *TACAS*, volume 9035 of *Lecture Notes in Computer Science*, pages 596–610. Springer, 2015.

[RFB14] Thomas Reinbacher, Matthias Függer, and Jörg Brauer. Runtime verification of embedded real-time systems. *Formal Methods in System Design*, 44(3):203–239, 2014.

[RHM17] Hendrik Roehm, Thomas Heinz, and Eva Charlotte Mayer. Stlinspector: STL validation with guarantees. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10426 of *Lecture Notes in Computer Science*, pages 225–232. Springer, 2017.

[San15] Ocan Sankur. Symbolic quantitative robustness analysis of timed automata. In Christel Baier and Cesare Tinelli, editors, *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2015)*, volume 9035 of *Lecture Notes in Computer Science*, pages 484–498. Springer, 2015.

[Sch86] Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, Inc., New York, NY, USA, 1986.

[Sun90] Daniel Sunday. A very fast substring search algorithm. *Commununications of the ACM*, 33(8):132–142, 1990.

[TM17] Tamás Tóth and István Majzik. Lazy reachability checking for timed automata using interpolants. In Alessandro Abate and Gilles Geeraerts, editors, *Formal*

Modeling and Analysis of Timed Systems - 15th International Conference, FORMATS 2017, Berlin, Germany, September 5-7, 2017, Proceedings, volume 10419 of *Lecture Notes in Computer Science*, pages 264–280. Springer, 2017.

[TR05] Prasanna Thati and Grigore Rosu. Monitoring algorithms for metric temporal logic specifications. *Electron. Notes Theor. Comput. Sci.*, 113:145–162, 2005.

[UFAM14] Dogan Ulus, Thomas Ferrère, Eugene Asarin, and Oded Maler. Timed pattern matching. In Axel Legay and Marius Bozga, editors, *Proceedings of the 12th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2014)*, volume 8711 of *Lecture Notes in Computer Science*, pages 222–236. Springer, 2014.

[UFAM16] Dogan Ulus, Thomas Ferrère, Eugene Asarin, and Oded Maler. Online timed pattern matching using derivatives. In Marsha Chechik and Jean-François Raskin, editors, *Proceedings of the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2016)*, volume 9636 of *Lecture Notes in Computer Science*, pages 736–751. Springer, 2016.

[Ulu17] Dogan Ulus. Montre: A tool for monitoring timed regular expressions. In Rupak Majumdar and Viktor Kuncak, editors, *Proceedings of the 29th International Conference on Computer Aided Verification (CAV 2017), Part I*, volume 10426 of *Lecture Notes in Computer Science*, pages 329–335. Springer, 2017.

[UM18] Dogan Ulus and Oded Maler. Specifying timed patterns using temporal logic. In *HSCC*, pages 167–176. ACM, 2018.

[Var95] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In Faron Moller and Graham M. Birtwistle, editors, *Logics for Concurrency - Structure versus Automata (8th Banff Higher Order Workshop, Banff, Canada, August 27 - September 3, 1995, Proceedings)*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer, 1995.

[WA19] Masaki Waga and Étienne André. Online parametric timed pattern matching with automata-based skipping. In *NFM*, volume 11460 of *Lecture Notes in Computer Science*, pages 371–389. Springer, 2019.

[Wag19] Masaki Waga. Online quantitative timed pattern matching with semiring-valued weighted automata. In *Proceedings of the 17th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2019)*, volume 11750 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2019.

[WAH16] Masaki Waga, Takumi Akazaki, and Ichiro Hasuo. A Boyer-Moore type algorithm for timed pattern matching. In Martin Fränzle and Nicolas Markey, editors, *Proceedings of the 14th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2016)*, volume 9884 of *Lecture Notes in Computer Science*, pages 121–139. Springer, 2016.

[WAH19] Masaki Waga, Étienne André, and Ichiro Hasuo. Symbolic monitoring against specifications parametric in time and data. In *Proceedings of the 31st International*

Conference on Computer Aided Verification (CAV 2019), Part I, volume 11561 of *Lecture Notes in Computer Science*, pages 520–539. Springer, 2019.

[WAH20] Masaki Waga, Étienne André, and Ichiro Hasuo. Model-bounded monitoring of hybrid systems. Unpublished manuscript, 2020.

[WH18] Masaki Waga and Ichiro Hasuo. Moore-machine filtering for timed and untimed pattern matching. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 37(11):2649–2660, 2018.

[WHS17] Masaki Waga, Ichiro Hasuo, and Kohei Suenaga. Efficient online timed pattern matching by automata-based skipping. In Alessandro Abate and Gilles Geeraerts, editors, *Proceedings of the 15th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2017)*, volume 10419 of *Lecture Notes in Computer Science*, pages 224–243. Springer, 2017.

[WHS18] Masaki Waga, Ichiro Hasuo, and Kohei Suenaga. MONAA: A tool for timed pattern matching with automata-based acceleration. In *Proceedings of the 3rd Workshop on Monitoring and Testing of Cyber-Physical Systems (MT@CPSWeek 2018)*, pages 14–15. IEEE, 2018.

[ZLD12] Xian Zhang, Martin Leucker, and Wei Dong. Runtime verification with predictive semantics. In *NFM*, volume 7226 of *Lecture Notes in Computer Science*, pages 418–432. Springer, 2012.

Index