Montre: A Tool for Monitoring Timed Regular Expressions

Dogan Ulus[(✉)]

Verimag, Université Grenoble-Alpes,
Grenoble, France
doganulus@gmail.com

Abstract. We present Montre, a monitoring tool to search patterns specified by timed regular expressions over real-time behaviors. We use timed regular expressions as a compact, natural, and highly-expressive pattern specification language for monitoring applications involving quantitative timing constraints. Our tool essentially incorporates online and offline timed pattern matching algorithms so it is capable of finding all occurrences of a given pattern over both logged and streaming behaviors. Furthermore, Montre is designed to work with other tools via standard interfaces to perform more complex and versatile tasks for analyzing and reasoning about cyber-physical systems. As the first of its kind, we believe Montre will enable a new line of inquiries and techniques in these fields.

1   Introduction

Temporal behaviors are sequences of actions and observations in time generated by various systems and the environment around us. A temporal pattern is a set of compositions of different temporal behaviors satisfying some relations among their components such as precedence and coincidence or possessing some properties such as repetition and a certain duration. Searching good [bad, desirable, undesirable] patterns over their temporal behaviors is an important task while we reason about systems and the environment.

Timed regular expressions (TREs) [2] extend regular expressions, a well-established formalism for specifying sequences of symbols, with the notion of real-time and timing constraints. Many patterns requiring both qualitative and quantitative temporal properties can be specified by TREs in a compact and natural way. Given a TRE that specifies a temporal pattern and a real-time behavior the problem of timed pattern matching is defined as locating all segments that satisfy the expression. This problem has been solved by an offline algorithm in [14]. It is further endowed with an online algorithm that incrementally matches patterns over streaming behaviors [15].

In this paper, we describe Montre a new tool for timed pattern matching whose applications are numerous and diverse. First of all, Montre can naturally check execution traces of software and hardware systems against real time properties specified in TRE (e.g. [5,7]), thus complementing temporal logic based
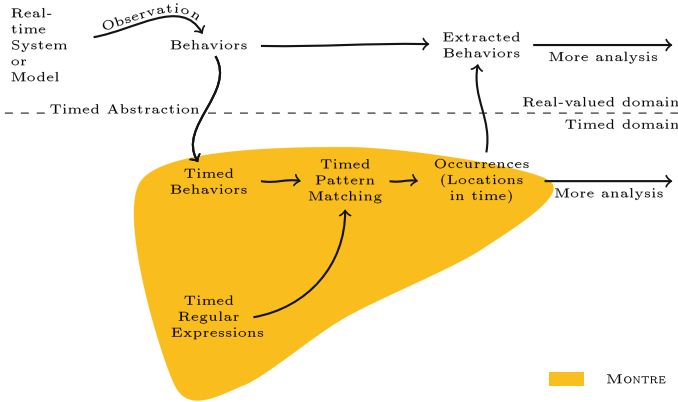
Fig. 1. The work flow and extent of the monitoring tool MONTRE

property checkers such as [1,4,11]. Further, MONTRE can be used for specification mining such as [3,8] as matching is a basic task for mining. Outside the verification context, MONTRE has a potential use in temporal data mining [10] and (vehicle or human) trajectory data mining [9,16] as it can label time segments with meaningful tags such as overtaking (another car) or sprinting. To illustrate our tool in action, we present such an example from the domain of sports analytics in Sect. 3 where we find all sprints of a soccer player.

2 Tool Description

The tool MONTRE essentially incorporates online and offline timed pattern matching algorithms extended with some practical features such as anchors and a Boolean layer. It takes a timed behavior and a timed regular expression as inputs, and produces a finite set of two dimensional zones representing the (possibly uncountable) set of segments that watch the pattern. MONTRE provides a standard text-based interface for easy integration with other tasks such as data preparations and visualization as we consider them necessary but outside the scope of MONTRE. In Fig. 1, we illustrate the work flow and extent of MONTRE, and we give details for each component in the following.

Timed Behaviors. A timed behavior is a sequence of time segments where each segment has a duration value and is associated with a set of propositional variables that hold continuously in the segment. In general, we assume all propositions are concurrent. For example, $(3, pq); (2, q); (2, p)$ is a timed behavior with 3 segments over propositions p and q. It means that p and q evaluate to true for the first 3 time units, then q is true for 2 more time units, and then p is true for 2 time units again. We assume behaviors start at time 0; therefore, the example behavior can be alternatively stated such that p holds from 0 to 3 and then 5 to 7 while q holds from 0 to 5.

Table 1. Montre timed regular expression syntax

Construct	Description
p	A propositional variable.
!P	Boolean NOT operation on P.
P || Q	Boolean OR operation on P and Q.
P && Q	Boolean AND operator on P and Q.
P	occurs on (t, t') if P holds from t to t' continuously.
<:P	occurs on a time period (t, t') if P occurs on (t, t') and there exists a rising edge for P at t.
P:>	occurs on a time period (t, t') if P occurs on (t, t') and there exists a falling edge for P at t'.
<:P:>	occurs on a time period (t, t') if P occurs on (t, t') and there exists a rising edge for P at t as well as a falling edge for P at t'.
E;F	occurs on a time period (t, t') if E occurs on (t, t'') and F occurs on (t'', t') for $t \leq t'' \leq t'$.
E|F	occurs on a time period (t, t') if either E or F occurs on (t, t').
E&F	occurs on a time period (t, t') if E and F occur on (t, t') concurrently.
E%(m,n)	occurs on a time period (t, t') if E occurs on (t, t') and the duration of the occurrence is in the specified range such that $\mathtt{m} \leq t' - t \leq \mathtt{n}$.
E*	Zero-or-more repetition of E.
E+	One-or-more repetition of E.

Timed Regular Expressions. An atomic timed regular expression corresponds to a Boolean expression over a set of propositions, denoted by letters p, q, r. These propositions can stand for predicates over real-valued variables. Usual Boolean operators (!), (||), (&&) are used to build Boolean expressions. We say that an atomic expression occurs on a time period (t, t') if the corresponding Boolean expression holds from t to t' continuously. Complex timed regular expressions are built from other expressions by using TRE operators: sequential composition (concatenation) (;), time restriction (%), choice (|), coincidence (&) and zero-or-more repetition (*). Further, we add one-or-more repetition (+) and two anchoring (<: and :>) operators to the set of operators. Typically parentheses are used to group expressions. We summarize all Boolean and TRE operations in Montre in Table 1.

Zones. For a proposition p that holds from t_1 to t_2, all sub-periods of (t_1, t_2) satisfy the expression p. As shown in Fig. 2-(i), such a set of matches $\{(t, t') \mid t_1 \leq t < t' \leq t_2\}$ can be represented on a two-dimensional plane as a triangular zone. Then the match set of any atomic expression would be a union of such triangular zones. A triangular zone is a special case of zones, which constitutes a restricted class of convex polygons defined by orthogonal and diagonal constraints as shown Fig. 2-(ii). Zones are basic data objects for timed pattern matching as unions of zones are closed under Boolean and regular operations. It follows that the match set of any timed regular expression over a timed behavior can be representable by a finite union of zones.
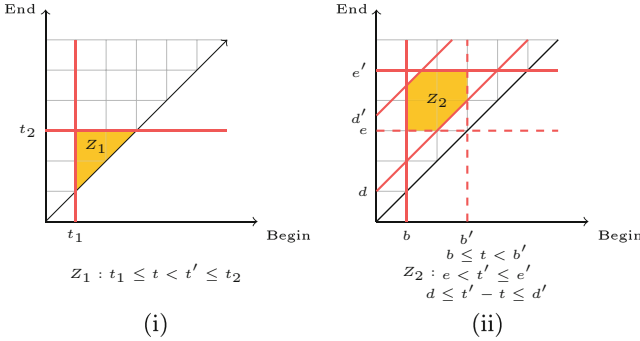
$$Z_1 : t_1 \leq t < t' \leq t_2$$

(i)

$$Z_2 : \begin{array}{c} b \leq t < b' \\ e < t' \leq e' \\ d \leq t' - t \leq d' \end{array}$$

(ii)

Fig. 2. (i) A triangular zone. (ii) A zone in general.

Implementation. MONTRE is a command line program[1] that uses structured text files for input/output specification. When invoked MONTRE parses the timed regular expression passed as an argument and starts to reads the input file. According to flags set by the user MONTRE would run in either online or offline mode. For online mode it is possible that the input can be given interactively using the command line or directed from another process as usual. At its core, MONTRE contains our efficient zone manipulation library, `libmontre`, called dynamically by top-level online and offline timed pattern matching algorithms. As Boolean and regular operations over sets of zones are intensive numerical computations, we have implemented `libmontre` in C++. In the implementation, we use an integer-valued time model where all time values are represented by integers for efficiency and accuracy reasons. For the majority of applications, integers give us sufficient precision and range; and a proper scaling can be found.

We implement timed pattern matching algorithms in Pure[2], a functional programming language based on term rewriting with a support for native code compilation and native calls to dynamic libraries. For the online algorithm [15], built upon derivatives of regular expressions [12,13], we extensively use the rewriting functionality when deriving an expression with respect to a newly observed segment. The offline algorithm [14] is a recursive computation over the syntax tree of the expression; therefore, the role of Pure's rewriting engine is minimal. The worst case complexity is polynomial in the size of input behavior and expression for the offline approach. For the online approach it is polynomial in the size of the behavior and exponential in expression. In practice, however, we realistically assume patterns to be much shorter than behaviors and somewhat sparse in them. Then we expect a linear-time performance in the size of input behavior for both algorithms. Under these assumptions, MONTRE can process timed behaviors with a size of 1M segments in a few seconds (offline) and a few hundred seconds (online).

---

[1] Available at http://github.com/doganulus/montre.
[2] Available at http://purelang.bitbucket.io.

3   An Illustrative Example

We present an example use of Montre on a data set obtained by tracking positions of players in a real soccer match. In this example, we find all sprints performed by a single player where a sprint is formally specified by a timed regular expression over speed and acceleration behaviors. The data are obtained by a computer vision algorithm with a frame rate of 10 Hz so we have a xy-coordinate for each player on the field at every 100 milliseconds. Therefore we use milliseconds as our base time unit for behaviors and expressions.

In order to specify a pattern for sprints, we need to address two issues in order: (1) how to categorize continuous speed and acceleration axes, and (2) which composition of these categories defines a sprinting effort best. Clearly, there are no universal answers for these questions so we rely on the study [6] in the following. First, we partition speed and acceleration axes into four categories (near-zero, low, medium, and high), and we associate a letter for each category in Table 2. For example, a period of medium speed, denoted by `r`, means the speed value resides between 3.7 and 6 m/s during the period.

Often a sprint effort is characterized by any movement above a certain speed threshold for a limited time. This gives us our first sprint pattern such that a period of high speed between 1–10 s, formally written as follows:

$$(<:s:>)\%(1000,10000) \tag{P1}$$

Above we use anchor operators from both sides on the proposition `s` to obtain only maximal periods that satisfy `s`; otherwise, any sub-period satisfies the pattern as well. The operator `%` specifies that the duration is restricted to be in 1000 and 10000 milliseconds. Alternatively we may want to find other efforts starting with high acceleration but not reaching top speeds necessarily. This gives us our second sprint pattern such that a period of high acceleration followed by a period of medium or high speed between 1–10 s, formally written as follows:

$$(<:g);(<:(r||s):>)\%(1000,10000) \tag{P2}$$

Notice that we do not use the right-anchor on `g`. This allows a medium or high speed period to overlap with a high acceleration period as it is usually the case that they are concurrent. Writing an equivalent pattern using classical

Table 2. Speed and acceleration thresholds [6].

Symbol	Label	Speed thresholds ($m \cdot s^{-1}$)	Symbol	Label	Acceleration thresholds ($m \cdot s^{-2}$)
s	High	> 6.0	g	High	>1.60
r	Medium	3.7 - 6	f	Medium	1.17 - 1.60
q	Low	2 - 3.7	e	Low	0.57 - 1.17
p	Near Zero	0 - 2	d	Near Zero	-0.57 - 0.57

(a) Entire movement   (b) `P1`

(c) `P2`   (d) `P3`
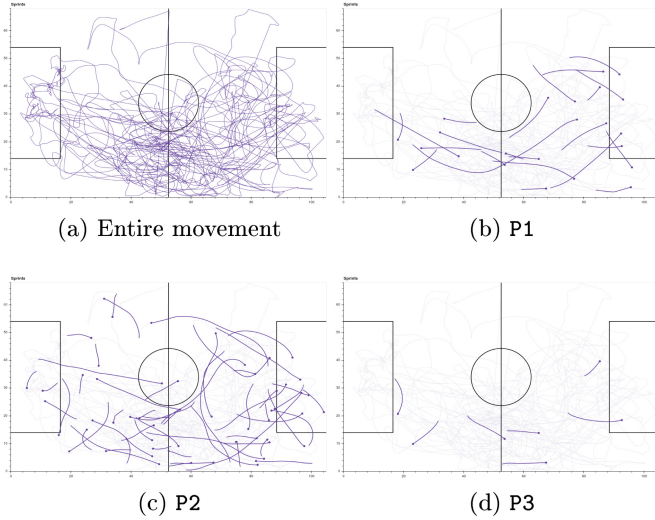
Fig. 3. The trajectory of a soccer player for 45 min on the field, and his sprinting periods found by MONTRE for patterns `P1-P3`.

regular expressions over a product alphabet would be a very tedious task partly due to a requirement to handle such interleavings explicitly (and the lack of timing constraints). For TREs all propositions are considered to be concurrent by definition, which results in concise and intuitive expressions. Finally we give a third pattern to find rather short but intense sprints such that

$$(<:(f||g));((<:s:>)\%(1000,2000)) \tag{P3}$$

Then we visualize all sprints found by MONTRE for patterns `P1-P3` in Fig. 3 over the behavior of a single player during one half of the game (45 min.) containing 27 K data points that reduces to timed behaviors of 5K segments after pre-processing. Note that we used Python to prepare data and visualize results.

4   Conclusions

Timed regular expressions can define many timed properties and MONTRE is the first tool to check such properties and detect timed patterns. Its performance is satisfactory for such monitoring tasks but we note that there is still some room for optimization especially for the online algorithm. The example we presented illustrates a complete MONTRE experience from raw data to visualization. As seen defining good patterns and categories are important to achieve intended results but it is not always obvious what a good pattern is. Such patterns should be found in the future using (unsupervised) pattern mining methods. We believe MONTRE would provide a good starting point for such research as it encapsulates timed pattern matching with an easy-to-use interface.

References

1. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TaLiRo: a tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 254–257. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19835-9_21

2. Asarin, E., Caspi, P., Maler, O.: Timed regular expressions. J. ACM **49**(2), 172–206 (2002)

3. Cutulenco, G., Joshi, Y., Narayan, A., Fischmeister, S.: Mining timed regular expressions from system traces. In: Workshop on Software Mining, pp. 3–10 (2016)

4. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 167–170. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14295-6_17

5. Dustdar, S., Gambi, A., Krenn, W., Nickovic, D.: A pattern-based formalization of cloud-based elastic systems. In: Principles of Engineering Service-Oriented and Cloud Systems (PESOS), pp. 31–37 (2015)

6. Dwyer, D.B., Gabbett, T.J.: Global positioning system data analysis: velocity ranges and a new definition of sprinting for field sport athletes. The. J. Strength Cond. Res. **26**(3), 818–824 (2012)

7. Ferrère, T., Maler, O., Ničković, D., Ulus, D.: Measuring with timed patterns. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015 Part II. LNCS, vol. 9207, pp. 322–337. Springer, Cham (2015). doi:10.1007/978-3-319-21668-3_19

8. Jin, X., Donzé, A., Deshmukh, J.V., Seshia, S.A.: Mining requirements from closed-loop control models. In: Hybrid Systems: Computation and Control, (HSCC), pp. 43–52 (2013)

9. Mazimpaka, J.D., Timpf, S.: Trajectory data mining: a review of methods and applications. J. Spat. Inf. Sci. **2016**(13), 61–99 (2016)

10. Mitsa, T.: Temporal Data Mining. CRC Press, Boca Raton (2010)

11. Nickovic, D., Maler, O.: AMT: a property-based monitoring tool for analog systems. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 304–319. Springer, Heidelberg (2007). doi:10.1007/978-3-540-75454-1_22

12. Owens, S., Reppy, J.H., Turon, A.: Regular-expression derivatives re-examined. J. Funct. Program. **19**(2), 173–190 (2009)

13. Roşu, G., Viswanathan, M.: Testing extended regular language membership incrementally by rewriting. In: Nieuwenhuis, R. (ed.) RTA 2003. LNCS, vol. 2706, pp. 499–514. Springer, Heidelberg (2003). doi:10.1007/3-540-44881-0_35

14. Ulus, D., Ferrère, T., Asarin, E., Maler, O.: Timed pattern matching. In: Legay, A., Bozga, M. (eds.) FORMATS 2014. LNCS, vol. 8711, pp. 222–236. Springer, Cham (2014). doi:10.1007/978-3-319-10512-3_16

15. Ulus, D., Ferrère, T., Asarin, E., Maler, O.: Online timed pattern matching using derivatives. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 736–751. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49674-9_47

16. Zheng, Y.: Trajectory data mining: an overview. ACM Trans. Intell. Syst. Technol. (TIST) **6**(3), 29 (2015)