Online Timed Pattern Matching using Automata[⋆]

Alexey Bakhirkin[1], Thomas Ferrère[2], Dejan Nickovic[3], Oded Maler[1], and Eugene Asarin[4]

[1] Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, 38000 Grenoble, France
[2] IST Austria
[3] AIT Austrian Institute of Technology
[4] IRIF, Université Paris Diderot

Abstract. We provide a procedure for detecting the sub-segments of an incrementally observed Boolean signal w that match a given temporal pattern φ. As a pattern specification language, we use timed regular expressions, a formalism well-suited for expressing properties of concurrent asynchronous behaviors embedded in metric time. We construct a timed automaton accepting the timed language denoted by φ and modify it slightly for the purpose of matching. We then apply zone-based reachability computation to this automaton while it reads w, and retrieve all the matching segments from the results. Since the procedure is automaton based, it can be applied to patterns specified by other formalisms such as timed temporal logics reducible to timed automata or directly encoded as timed automata. The procedure has been implemented and its performance on synthetic examples is demonstrated.

1 Introduction and Motivation

Complex cyber-physical systems and reactive systems in general exhibit temporal behaviors that can be viewed as dense-time signals or discrete-time sequences and time-series. The correctness and performance of such systems is based on properties satisfied by these behaviors. In formal verification, a system model is used to generate *all* possible behaviors and check for their inclusion in the language defined by the specifications. In runtime verification, interpreted as lightweight simulation-based verification, property satisfaction by *individual* system behaviors is checked. In many situations, we would like to monitor the ongoing behavior of a real system, already deployed and running, rather than traces of a simulation model during design time. In this context we want to detect property violation and other patterns of interest such as suspicious activities, degradation of performance and other alarming signs known to precede

[⋆] This research was supported in part by the Austrian Science Fund (FWF) under grants S11402-N23 (RiSE/SHiNE) and Z211-N23 (Wittgenstein Award), and by the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement nr. 306595 "STATOR".

unpleasant or even catastrophic situations. The detection of such patterns and the reaction to them can be the basis of another level of supervisory control that reacts to situations as they occur without achieving the challenging and often impossible task of verifying offline against all possible scenarios. In fact, in the software engineering literature for safety-critical systems there is an actuator-monitor decomposition (*safety bag*) where one module computes the reaction while the other checks the results. Such an architecture has been proposed recently as a way to handle autonomous systems such as driver-free cars [15].

Properties traditionally used in verification often correspond to complete behaviors or their prefixes that start from time zero. In pattern matching one is interested in segments that may start and end in any time point. To be useful, the detection of patterns should be done *online*, as soon as they occur. In this paper we provide an automaton-based online pattern matching procedure where patterns are specified using timed regular expressions, a formalism suitable for describing patterns in concurrent and asynchronous behaviors. Before moving to our contribution, let us make a brief survey of some well-known classical results on string pattern matching. A regular expression φ defines a regular language $\mathcal{L}(\varphi)$ consisting of sequences that match it. It can be transformed into a non-deterministic automaton $\mathcal{A}_\varphi$ that recognizes $\mathcal{L}(\varphi)$ in the sense that $\mathcal{L}(\varphi)$ is exactly the set of words w that admit a run from an initial to a final state. [19] The existence for such an accepting run in $\mathcal{A}_\varphi$ can be checked by exploring the finitely many such runs associated with w. As a byproduct of classifying w, the automaton can classify any prefix $w[0..j]$ of w for $j < k$, just by observing the states reachable by the runs at time j. It is worth noting that the determinization procedure (subset construction) due to Rabin and Scott [21] corresponds to a breadth-first exploration of these runs for all words, combined with the fact that runs that reach the same state can be merged (see Figure 3). A simple modification of $\mathcal{A}_\varphi$, to our knowledge first described in [23], allows also to find all sub-segments $w[i..j]$ of w that match φ. First, a counter t' is used which increments each time a symbol is read – such a counter exists anyway in any implementation of a membership tester as a pointer to the current symbol in the sequence. Then a new initial state s is added to $\mathcal{A}_\varphi$, and in this state the automaton can self-loop indefinitely and postpone the selection of the point in time when it starts reading w. When it moves to the original initial state of $\mathcal{A}_\varphi$, it records the start time in an auxiliary variable t (see Figures 1 and 2 for an example). Then whenever there is a run reaching an accepting state with $t = r$ and $t' = r'$, one can conclude that $w[r..r']$ matches φ. This modification, also pointed to in [1], enabled the later implementation of fast and reliable string matching tools [2,20] which are now standard. The application of regular expressions since extends beyond text processing but also occurs, e.g. in DNA analysis and programming languages.

To reason about the dynamic behaviors of complex systems, cyber-physical or not, one needs a language richer than traditional expressions [26]. To start with, the behaviors of such systems, unlike sequential text, are multi-dimensional by nature, involving several state variables and components working in parallel. In

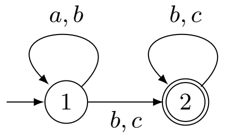

Fig. 1. A non-deterministic automaton for $(a \cup b)^* \cdot (b \cup c)^+$.

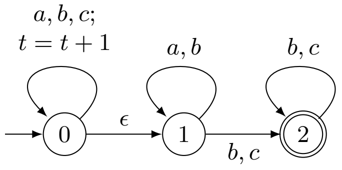

Fig. 2. A non-deterministic matching automaton for $(a \cup b)^* \cdot (b \cup c)^+$. Note the new initial state and the counter t.

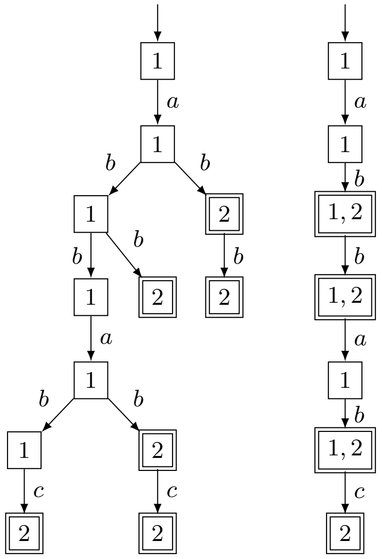

Fig. 3. Runs of the automaton in Fig. 1. *Left* – all runs for the word *abbabc*. *Right* – breadth-first on-the-fly subset construction.

principle, it is possible to express patterns in the behavior of such systems using a global product alphabet, but such a flattening of the structure is impractical for both readability and complexity reasons. Instead we use a more symbolic variant of regular expressions (first used in the timed setting in [27], see also the proposal [13] to add timing to the regular expressions of the industrial specification language PSL which admits variables) where one can refer to state variables and write expressions like $p \cdot q$ rather than $(pq \cup p\bar{q}) \cdot (pq \cup \bar{p}q)$. Needless to say, the whole practice of verification, starting with temporal logic specifications, via compositional system descriptions to their symbolic model-checking [9] is based on such an approach, which seems to be less developed in formal language theory. To reason about what happens in various parts of the system, we also employ intersection in our syntax. In the one-dimensional untimed case it does not increase the expressive power, but affects the complexity of online membership testing since the minimal DFA translating such expressions can be exponentially larger [12]. The second observation is that system components need not be synchronized and they may operate on different time scales. Consequently, reasoning in discrete time with a pre-selected time step is wasteful and we use instead the timed regular expressions of [4,5], a formalism tailored for specifying properties of timed behaviors such as Boolean signals [4] or time-event sequences [5], where value changes and events can occur anywhere along the time axis. Thus the expression $p \cap (\mathsf{true} \cdot q \cdot \mathsf{true})$ is matched by any segment of a Boolean signal where p holds continuously and a burst of q occurs anywhere inside this segment. Using duration constraints we can refine the patterns, for example the expression $p \cap (\mathsf{true} \cdot \langle q\rangle_{[a,\infty]} \cdot \mathsf{true})$ considers only q-bursts that last for at least a time. The problem of timed pattern matching has been introduced and solved in [27]: find all the sub-segments of a multi-dimensional Boolean signal of a bounded variability that match a timed regular expression. This work was automaton-free and worked inductively on the structure of the formula in an offline manner. An online version of that procedure has been developed in [28] based on a novel extension of Brzozowski derivatives [8] to timed regular expressions and dense time signals. Both works, which have been implemented in the tool Montre [25] did not use the full syntax of timed regular expressions and hence did not match the expressive power of timed automata (see [14]). In this paper we explore an alternative automaton-based procedure whose scope of application is wider than the expressions used in [27,28] as it works with any timed language definable by a timed automaton and is agnostic about the upstream pattern specification language. Let us mention another recent automaton-based approach is the one of [29], a real-time extension of the Boyer-Moore pattern matching method. In contrast to our work, the procedure in [29] works on TA defined over time-event sequences and it requires pre-computing the region graph from the TA specification. The same authors improve this result in [30], by using a more efficient variant of the Boyer-Moore algorithm and by replacing the region automaton by the more efficient zone-simulation graph.

The essence of our contribution is the following. Starting with an expression φ, we build a non deterministic timed automaton $\mathcal{A}_\varphi$ which accepts $\mathcal{L}(\varphi)$. Then by a small modification, similar to the discrete case, we convert it to a matching automaton $\mathcal{A}'_\varphi$ with two additional clocks x_0 and x_s that record, respectively the time since the beginning of the signal (absolute time) and the time since we started reading it. Then, given a bounded variability Boolean signal we compute the reachability tree of the automaton whose nodes are pairs of the form (q, Z) where Z is a zone in the extended clock space of $\mathcal{A}'_\varphi$. This tree captures all (possibly uncountably many) runs induced by w. By projecting zones associated with accepting states of the automaton on x_0 and $x_0 - x_s$, we retrieve the matches. We combine this procedure with incremental observation of the input signal to obtain an online matching procedure. We implemented this procedure using the zone library of IF [7].

2 Preliminaries

Signals Let $\mathbb{B} = \{0, 1\}$ and let $P = \{p_1, \ldots, p_n\}$ be a set of propositions. A state over P is an element of 2^P or equivalently a Boolean vector $u \in \mathbb{B}^n$ assigning the truth value for any $p \in P$. The time domain $\mathbb{T}$ is taken to be the set of non-negative reals. A Boolean signal w of duration $|w| = d$ is a right-continuous function $w : [0, d) \to \mathbb{B}^n$ whose value at time t is denoted by $w[t]$. We use $w[t..t']$ to denote the part of signal w between absolute times t and t'. The concatenation of two signals w and w' of respective durations d and d' is the signal ww' of duration $d + d'$ defined as $ww'[t] = w[t]$ for all $t < d$, $ww'[t] = w'[t - d]$ for all

$t \in [d, d+d')$. We consider signals of finite variability, which can be written as a finite concatenation of constant signals. The empty signal of duration zero is denoted ϵ. We use $w \| w'$ to denote the parallel composition of two signals of the same duration defined over disjoint sets of propositions.

Timed Regular Expressions To specify sets of signals we use the timed regular expressions (TRE) of [4], augmented with the use of a structured alphabet represented by a set P of atomic propositions [27]. The set of state constraints, denoted by $\Sigma(P)$ is simply the set of Boolean formulas over P. The syntax of TRE is given by the grammar

$$\varphi ::= \epsilon \mid \sigma \mid \varphi \cup \varphi \mid \varphi \cap \varphi \mid \varphi \cdot \varphi \mid \varphi^+ \mid \langle \varphi \rangle_I \mid \exists p . \varphi$$

where $\sigma \in \Sigma(P)$ and $I \subseteq \mathbb{T}$ is an integer-bounded interval. As customary, iterations of φ are denoted in exponent with $\varphi^0 \equiv \epsilon$ and $\varphi^k \equiv \varphi^{k-1} \cdot \varphi$ for $k \geq 1$. The Kleene star is defined as $\varphi^* \equiv \epsilon \cup \varphi^+$.

Any TRE φ is associated with a set of signals, the timed language $\mathcal{L}(\varphi)$, via the following inductive definitions:

$$\begin{array}{ll}
\mathcal{L}(\epsilon) = \{\epsilon\} & \mathcal{L}(\varphi_1 \cdot \varphi_2) = \{w_1 \cdot w_2 \mid w_i \in \mathcal{L}(\varphi_i), i = 1, 2\} \\
\mathcal{L}(\sigma) = \{w \mid \forall t \in [0, |w|)\ w[t] \models \sigma\} & \mathcal{L}(\varphi^+) = \bigcup_{k=1}^{\infty} \mathcal{L}(\varphi^k) \\
\mathcal{L}(\varphi_1 \cup \varphi_2) = \mathcal{L}(\varphi_1) \cup \mathcal{L}(\varphi_2) & \mathcal{L}(\langle \varphi \rangle_I) = \{w \mid w \in \mathcal{L}(\varphi) \wedge |w| \in I\} \\
\mathcal{L}(\varphi_1 \cap \varphi_2) = \mathcal{L}(\varphi_1) \cap \mathcal{L}(\varphi_2) & \mathcal{L}(\exists p . \varphi) = \{w \mid \exists w' \text{ over } \{p\}, w \| w' \in \mathcal{L}(\varphi)\}
\end{array}$$

Note that signals in $\mathcal{L}(\sigma)$ need not be a constant, for example $\mathcal{L}(p_i)$ consists of all signals in which p_i is constantly true but p_j, for $j \neq i$ may go up and down. Note also that the semantics of σ does not specify any duration, and in this sense it resembles σ^* in classical regular expressions. The duration restriction is expressed using the $\langle \varphi \rangle_I$ operation. The $\exists p . \varphi$ operation corresponds to the renaming operation in [4] which has been proven in [14] to be necessary in order to match the expressive power of timed automata.

Timed Automata We use a variant of timed automata, finite-state automata extended with real-valued clock variables, as acceptors of timed languages over signals. Unlike the automata introduced originally in [3] and used in [5], which are event-based with alphabet symbols associated with transitions, we use a state-based approach [4] where signal values are associated with time passage inside states. Let $X = \{x_1, \ldots, x_m\}$ be a set of clock variables. A *clock constraint* is a Boolean combination of inequalities of the form $x \bowtie c$ where $c \in \mathbb{N}$ is a constant, $x \in X$ is a clock variable, and $\bowtie \in \{<, \leq, =, \geq, >\}$ is a comparison sign. The set of clock constraints over X is written $\Phi(X)$. A *valuation* $v \in \mathbb{T}^m$ associates any $x \in X$ with a delay denoted $v(x) \in \mathbb{T}$.

Definition 1 (Timed Automaton). *A timed automaton over signals is a tuple $\mathcal{A} = (P, X, L, S, i, o, \Delta)$ with locations L, initial locations $S \subseteq L$, input labeling $i : L \to \Sigma(P)$, output labeling $o : L \to \Phi(X)$, and set of edges $\Delta \subseteq L \times \Phi(X) \times 2^X \times L$.*

A state (configuration) of the automaton is a pair (ℓ, v) where ℓ is a location and v is a clock valuation. The behavior of the automaton while reading a signal consists of an alternation of two types of steps:

- A time step $(\ell, v) \overset{w}{\rightsquigarrow} (\ell, v+r)$ where the automaton consumes a signal w of duration r while advancing all clocks in the same pace provided that the signal satisfies continuously the state invariant specified by the input label: $\forall t \in [0, r)\ w[t] \models i(\ell)$;
- A discrete step $(\ell, v) \xrightarrow{\delta} (\ell', v')$ for some transition $\delta = (\ell, \varphi, R, \ell') \in \Delta$ such that $v \models \varphi$ (clocks satisfy transition guard) and $v' = v[R \leftarrow 0]$ (clocks in R are reset while taking the transition).

A *run* of automaton $\mathcal{A}$ over a signal w is a sequence

$$(\ell_0, 0) \overset{w_0}{\rightsquigarrow} (\ell_0, v_0) \xrightarrow{\delta_1} (\ell_1, v_1') \overset{w_1}{\rightsquigarrow} (\ell_1, v_1) \xrightarrow{\delta_2} \dots \xrightarrow{\delta_n} (\ell_n, v_n') \overset{w_n}{\rightsquigarrow} (\ell_n, v_n)$$

of discrete and time steps such that $w = w_1 w_2 \dots w_n$, starting with an initial configuration, that is, $\ell_0 \in S$ and $v_0(x) = 0$ for all $x \in X$. A run is accepting if it ends in an accepting configuration: $v_n \models o(\ell_n)$. The language $\mathcal{L}(\mathcal{A})$ is the set of signals admitting an accepting run.

The object that we are going to compute is the *match set* of a signal w with respect to a timed language $\mathcal{L}$ defined either by an automaton $\mathcal{A}$ or a timed regular expression: $\mathcal{M}(w, \mathcal{L}) = \{(t, t') : w[t..t'] \in \mathcal{L}\}$. In [27] it has been proved that for an expression φ, $\mathcal{M}(w, \mathcal{L}(\varphi))$ is a finite union of two-dimensional zones. Our results extend it to languages accepted by timed automata where (t, t') belong to the match set if $w[t..t']$ admits an accepting run in $\mathcal{A}$.

Translating TRE into Automata We now demonstrate, following [4], how a timed regular expression translates into a timed automaton accepting the same language. The construction is rather straightforward and the reader is referred to [4] for a more lengthy intuitive presentation. The construction of an automaton $\mathcal{A}_\varphi = (P, X_\varphi, L_\varphi, S_\varphi, i_\varphi, o_\varphi, \Delta_\varphi)$ accepting $\mathcal{L}(\varphi)$, is obtained by structural induction. In the description that follows we assume that automata given by induction hypothesis have disjoint sets of locations, but may share the same clocks except for the case of intersection.

- Empty word: $\mathcal{A}_\epsilon$ is defined by letting $X_\epsilon = \{x\}$, $L_\epsilon = S_\epsilon = \{\ell\}$, $i_\epsilon(\ell) \equiv \mathsf{true}$, $o_\epsilon(\ell) \equiv (x = 0)$, and $\Delta_\epsilon = \emptyset$.
- State expressions: $\mathcal{A}_\sigma$ is defined by $X_\sigma = \emptyset$, $L_\sigma = S_\sigma = \{\ell\}$, $i_\sigma(\ell) \equiv \sigma$, $o_\sigma(\ell) \equiv \top$, and $\Delta_\sigma = \emptyset$.
- Union: $\mathcal{A}_{\varphi \cup \psi}$ is defined as the component-wise union of $\mathcal{A}_\varphi$ and $\mathcal{A}_\psi$.
- Intersection: $\mathcal{A}_{\varphi \cap \psi}$ is given by $X_{\varphi \cap \psi} = X_\varphi \uplus X_\psi$, $L_{\varphi \cap \psi} = L_\varphi \times L_\psi$, $S_{\varphi \cap \psi} = S_\varphi \times S_\psi$, with input labels $i_{\varphi \cap \psi}(\ell, m) = i_\varphi(\ell) \wedge i_\psi(m)$ for every $\ell \in L_\varphi, m \in L_\psi$, similarly for output labels. For a pair of edges $(\ell, \beta, Q, \ell') \in \Delta_\varphi$, $(m, \gamma, R, m') \in \Delta_\psi$, $\mathcal{A}_{\varphi \cap \psi}$ has the edges $((\ell, m), \beta, Q, (\ell', m))$, $((\ell, m), \gamma, R, (\ell, m'))$, and $((\ell, m), \beta \wedge \gamma, Q \cup R, (\ell', m'))$.
- Concatenation: we let $X_{\varphi \cdot \psi} = X_\varphi \cup X_\psi$, $L_{\varphi \cdot \psi} = L_\varphi \cup L_\psi$ and $S_{\varphi \cdot \psi} = S_\varphi$. Labels are given by $i_{\varphi \cdot \psi}(\ell) \equiv i_\varphi(\ell)$ if $\ell \in \mathcal{L}_\varphi$, $i_{\varphi \cdot \psi}(\ell) \equiv i_\psi(\ell)$ otherwise; $o_{\varphi \cdot \psi}(\ell) \equiv \mathsf{false}$ if $\ell \in L_\varphi$, $o_{\varphi \cdot \psi}(\ell) \equiv o_\psi(\ell)$ otherwise. Edges are given by $\Delta_{\varphi \cdot \psi} = \Delta_\varphi \cup \Delta_\psi \cup \{(f, o_\varphi(f), X_\psi, s) \mid f \in L_\varphi, s \in S_\psi\}$.
- Iteration: $\mathcal{A}_{\varphi^+}$ is obtained from $\mathcal{A}_\varphi$ by adding edges $(f, o_\varphi(f), X_\varphi, s)$ for every pair $f \in L_\varphi$ and $s \in S_\varphi$.

– Duration constraint: $\mathcal{A}_{\langle\varphi\rangle_I}$ is obtained by using a fresh clock $x \notin X_\varphi$ and replacing output labels φ with $\varphi \wedge (x \in I)$ in every location.
– Existential quantification: $\mathcal{A}_{\exists p.\varphi}$ is obtained by replacing input labels σ with $\sigma[p \leftarrow \mathsf{true}] \vee \sigma[p \leftarrow \mathsf{false}]$ in every location.

The above procedure yields the following:

Theorem 1 (TRE ⇒ TA). *For any TRE of containing m atomic expressions and n duration constraints, one can construct an equivalent timed automaton with n clocks and 2^m locations.*

The exponential blow-up in the number of locations is solely due to the intersection operator, with repeated application of the product construction, and would otherwise vanish. For a proof of the other direction, TA ⇒ TRE, see [4,5].

3 Membership and Matching using Timed Automata

In this section we present a zone-based algorithm for testing membership of a signal in the language accepted by a timed automaton; then show how it can be extended to find and extract the match set of the signal in that language.

3.1 Checking Acceptance by Non-Deterministic Timed Automata

The automata constructed from expressions, as well as other typical timed automata, are non-deterministic. Part of this non-determinism is dense, coming from modeling duration uncertainty using intervals, and also from different factorizations of a signal segment into two concatenated expressions. Unlike classical automata, it can be shown that some timed automata cannot be determinized [3] and even determinizability of a given automaton is an undecidable problem [11]. However, these results are concerned with converting the non-deterministic TA into a deterministic one, equivalent with respect to *all* possible inputs which include signals of arbitrary variability and hence the number of clocks cannot be bounded. In contrast, exploring all the (uncountably many) possible runs of a non-deterministic timed automaton while reading a *given* signal of finite duration and variability is feasible, as has already been demonstrated [24,18,16] in the context of testing, using what has been termed on-the-fly subset construction. The procedure described in the sequel shows how despite their dense non-determinism, timed automata can be effectively used as membership testers for bounded-variability signals and eventually as online pattern matching devices. To this end we use a variant of the standard zone-based reachability algorithm for simulating uncountably many runs in parallel. This algorithm underlies all timed automata verification tools [10,6,7], see [31,17]. The procedure computes the simulation/reachability graph whose nodes are symbolic states of the form (ℓ, Z) where ℓ is a location and Z is a zone in the space of clock valuations.

Definition 2 (Zone). *Let X be a set of clock variables. A* difference constraint *is an inequality of the form $x - y \prec c$ for $x, y \in X$, $c \in \mathbb{T}$, and $\prec \in \{<, \leq\}$. A* zone *is a polytope definable as a conjunction of clock and difference constraints.*

Zones are known to be closed under intersection, projection, resets and forward time projection defined as $Z^{\nearrow} = \{v + t \mid v \in Z \wedge t \geq 0\}$. These operations are implemented as simple operations on the difference bound matrix (DBM) representing the zone.

Let $\mathcal{A} = (P, X, L, S, i, o, \Delta)$ be a timed automaton and let $\mathcal{A}'$ be the automaton obtained from $\mathcal{A}$ by adding an auxiliary clock x_0 which is never reset since the beginning and hence it keeps track of the absolute time. We will consider zones in the extended clock space, and denote extended clock valuations as (v_0, v). It is not hard to see that a configuration $(\ell, (v_0, v))$ is reachable in $\mathcal{A}'$ iff the input prefix $w[0..v_0]$ admits a run to (ℓ, v) in $\mathcal{A}$.

Definition 3 (Discrete Successor). *Let Z be a zone and let $\delta = (\ell, \varphi, R, \ell') \in \Delta$ be a transition. The δ-successor of Z is the zone*

$$Succ^{\delta} Z = \{v' : \exists v \in Z \ v \models \varphi \wedge v' = v[R \leftarrow 0]\}$$

While doing zone-based time passage in a zone, we need to restrict ourselves to segments of the signal that satisfy the input constraints of the location and this is made possible through the use of absolute time.

Definition 4 (Temporal Scope). *The temporal scope of a signal w in location ℓ is the set of time points where w satisfies the input constraint of ℓ:*

$$\mathcal{J}(\ell, w) = \{t : w[t] \models i(\ell)\}.$$

For a bounded variability signal, $\mathcal{J}(\ell, w)$ is a sequence $J_1, \ldots, J_k$ of disjoint intervals of the form $J_i = [\alpha_i, \beta_i)$.

When a symbolic state (ℓ, Z) is reached via a discrete transition, we need to split Z into zones on which time can progress, using the following operation.

$$E(\ell, Z) = \{Z \wedge \alpha \leq x_0 \leq \beta \mid [\alpha, \beta) \in \mathcal{J}(\ell, w)\}.$$

The procedure *Succ* (Algorithm 1) computes the successors of a symbolic state by one discrete transition and one time passage. The whole reachability algorithm (Algorithm 2) applies this procedure successively to all reachable symbolic states. It accepts as arguments the automaton $\mathcal{A}$, the signal w, and the set I of states from which to start the exploration. When calling *Reach* for the first time, we set I to be $\{(\ell, 0) \mid \ell \in S\}$. When *Reach* terminates, it outputs the set Q_{reach} of reachable symbolic states. From Q_{reach}, we can extract the set of accepting states by intersecting its elements with the output labels of locations: $Q_{\text{acc}} = \{(\ell, Z_a) \mid Z_a = Z \wedge o(\ell) \wedge Z_a \neq \emptyset \wedge (\ell, Z) \in Q\}$. If a configuration $(\ell, (v, v_0))$ is reachable and accepting (belongs to some element of Q_{acc}), then the prefix $w[0..v(x_0)]$ is accepted by the automaton.

Theorem 2 (Termination). *Given a finite-variability signal, Algorithm 2 terminates.*

Algorithm 1 $\mathit{Succ}(\ell, Z)$

```
**Require:** A timed automaton $\mathcal{A}$ and symbolic state $(\ell, Z)$;
**Require:** An input signal $w$ for which $\mathcal{J}(\ell, w)$ has been computed for every location.
**Ensure:** The set $Q$ of successors of $(\ell, Z)$ by one transition and one time step.
  $Q := \emptyset$
  $Q_1 := \{\mathit{Succ}^{\delta}(\ell, Z) \mid \delta \in \Delta\}$ {Discrete successors}
  for all non-empty $(\ell', Z') \in Q_1$ do
    $Q_2 := \{\ell'\} \times E(\ell', Z')$ {Compute the sub-zones of $Z'$ in which time can progress}
    for all non-empty $(\ell'', Z'') \in Q_2$ do
      $Q := Q \cup \{(\ell'', Z''^{\nearrow} \wedge x_0 \leq \beta_i)\}$ {Apply time passage until the corresponding
      upper bound}
    end for
  end for
  return $Q$
```

Algorithm 2 $\mathit{Reach}(\mathcal{A}, w, I)$

```
**Require:** A timed automaton $\mathcal{A}$, signal $w$, set of initial states $I$;
**Ensure:** The set $Q_{\text{reach}}$ of all symbolic states reachable while reading $w$
  $Q_{\text{reach}} := P := I$ {Initialization of visited and pending symbolic states}
  while $P \neq \emptyset$ do
    pick and remove $(\ell, Z) \in P$
    $Q_1 := \mathit{Succ}(\ell, Z)$
    for all $(\ell', Z') \in Q_1$ do
      if $(\ell', Z') \notin Q_{\text{reach}}$ then
        $Q_{\text{reach}} := Q_{\text{reach}} \cup \{(\ell', Z')\}$ {Add to visited}
        $P := P \cup \{(\ell', Z')\}$ {Add to pending}
      end if
    end for
  end while
  return $Q_{\text{reach}}$
```

Termination follows from the fact that the set of symbolic states is finite in our case. We can scale the signal so that all the switching points come at integer times, then, we can use zones with integer coefficients in Algorithm 2. The largest possible value of a clock and thus the largest constant that can appear in a reachable symbolic state is the duration of the signal, hence the number of possible symbolic states is finite.

Theorem 3 (Completeness). *There exists a run of the automaton $\mathcal{A}$:*

$$(\ell_0, 0) \overset{w_0}{\rightsquigarrow} (\ell_0, v_0) \xrightarrow{\delta_1} (\ell_1, v_1') \overset{w_1}{\rightsquigarrow} (\ell_1, v_1) \xrightarrow{\delta_2} \ldots \xrightarrow{\delta_n} (\ell_n, v_n') \overset{w_n}{\rightsquigarrow} (\ell_n, v_n)$$

if and only if the configuration (ℓ_n, v_n) belongs to Q_{reach}.

By induction on the number of discrete transitions, every reachable configuration is eventually visited by the algorithm as part of some symbolic state.

Algorithm 3 *ReachOnline*$(\mathcal{A}, w_i, Q_r^{i-1})$

```
Require: A timed automaton $\mathcal{A}$, signal segment $w_i$ defined on $[t_i, t_{i+1})$, previous set
    of reachable states $Q_r^{i-1}$;
Ensure: The set $Q_r^i$ contains the states reachable while reading $w_i$
  $I = \{(\ell, Z') \mid Z' = Z \wedge x_0 = t_i \wedge Z' \neq \emptyset \wedge (\ell, Z) \in Q_r^{i-1}\}$ {States that are reachable
    when $w_i$ starts}
  return $Reach(\mathcal{A}, w_i, I)$
```

3.2 Checking Acceptance Online

Algorithm 2 can be used to perform reachability computation in an online way. We can arbitrarily split the input signal w into segments $w_1, w_2, \cdots, w_n$ and present them one by one to the procedure *ReachOnline* (Algorithm 3). After processing segment w_i, the procedure returns the set of states reachable after reading $w_1 \cdots w_i$. From the previous set of reachable states, *ReachOnline* extracts the states which are reachable at the start of the new segment (those where the absolute time clock satisfies $x_0 = |w_1 \cdots w_i|$) and passes them on as initial states for *Reach*. More formally, we build the sequence

$$\begin{aligned}
Q_r^1 &= ReachOnline(A, w_1, \{(\ell, 0) \mid \ell \in S\}), \\
Q_r^2 &= ReachOnline(A, w_2, Q_r^1), \\
&\ldots \\
Q_r^n &= ReachOnline(A, w_n, Q_r^{n-1})
\end{aligned}$$

and take the set of reachable states Q_{reach} to be $\bigcup_{i=1}^n Q_r^i$.

One useful property of *ReachOnline* in terms of memory use is that when processing a segment of the signal, it does not need to store previously processed segments and, after the new initial states are extracted, it does not need to store previously computed reachable states. Another property of *ReachOnline* is that it does not care about how we split w into segments. Segments may have different duration, different number of switching points, etc. The way we split the signal affects the performance though. As the segment size gets smaller, the number of times Algorithm 2 is called increases, but the cost to process a segment decreases. The influence of this parameter on performance is discussed in Section 4.

3.3 From Acceptance to Matching

To compute the segments of w which are accepted by an automaton $\mathcal{A}$, we first construct a matching automaton $\mathcal{A}'$ similar to the one used in the discrete case. It can stay indefinitely in an added initial state before it moves to an initial state of $\mathcal{A}$, resets a clock x_s and starts reading the remaining part of w. The automaton also uses the absolute time clock x_0 used for acceptance, see Figure 4.

Definition 5 (Matching Timed Automaton). *Let $\mathcal{A} = (P, X, L, S, i, o, \Delta)$ be a timed automaton. Then the corresponding matching automaton is $\mathcal{A}' =$*

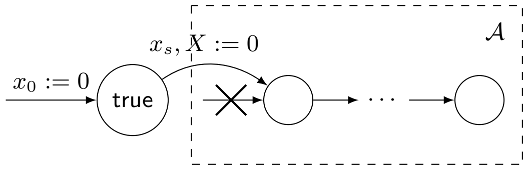

Fig. 4. Matching automaton $\mathcal{A}'$ for a property automaton $\mathcal{A}$.

$(P, X', L', S', i', o', \Delta')$, *where* $X' = X \cup \{x_0, x_s\}$*;* $\mathcal{L}' = \mathcal{L} \cup \{\ell_s\}$*;* $S' = \{\ell_s\}$, $i' = i \cup \{\ell_s \mapsto \textsf{true}\}$, $o' = o \cup \{\ell_s \mapsto \textsf{false}\}$*;* $\Delta' = \Delta \cup \{(\ell_s, \textsf{true}, X' - \{x_0\}, \ell) \mid \ell \in S\}$.

The start time of reading the segment is constantly maintained by the difference $x_0 - x_s$. As a generalization of the case of acceptance, w admits a run that ends in an extended configuration $(\ell, (v, v_0, v_s))$ in $\mathcal{A}'$ iff the signal segment $w[v_0 - v_s..v_0]$ admits a run in $\mathcal{A}$ that leads to (ℓ, v). Thus Algorithms 2 and 3 applied to $\mathcal{A}'$ compute the reachable symbolic states in the extended clock space. Projecting zones associated with accepting locations on x_0 and x_s we can can extract the matches. From Theorems 2 and 3 it follows that the for a given TRE and expression, the match set can be described by a finite set of zones. This extends the result obtained in [27] to arbitrary TA.

3.4 Example

Let us illustrate the matching algorithm with a simple example. As the pattern specification, we use the expression $\varphi = p{\cdot}q$, and we translate it to the automaton shown in Fig. 5. As input, we use the signal w from Fig. 6. The signal is split into two segments: w_1 defined in the interval $[0, 4)$, and w_2 defined in the interval $[4, 8)$. We run the matching algorithm presenting it one segment at a time.

When the segment w_1 arrives, we start the exploration with the symbolic state $(\ell_\top, x_0 = 0)$ and immediately apply the time transition to it. We can stay in the location $\ell_\top$ until the end of the segment, thus we add to the reachability tree the state $s_1 = (\ell_\top, x_0 \in [0, 4])$. Next, from s_1, we execute the discrete transition that leads to ℓ_p. Changing the location, resetting the clock x_s and constraining the zone to the interval where p holds, produces the state $(\ell_p, x_0 \in [1, 3] \wedge x_s = 0 \wedge x_0 - x_s \in [1, 3])$. Then, applying time elapse until the end of p produces the state $s_2 = (\ell_p, x_0 \in [1, 3] \wedge x_s = [0, 2] \wedge x_0 - x_s \in [1, 3])$ that we add to the reachability tree. Finally, we execute from s_2 the discrete transition to ℓ_q. After changing the location and restricting to the interval where q holds, we get the state $(\ell_q, x_0 \in [2, 3] \wedge x_s \in [0, 2] \wedge x_0 - x_s \in [1, 3])$. After applying time elapse until the end of q (the end of the fragment in this case), we get the state $s_3 = (\ell_q, x_0 \in [2, 4] \wedge x_s \in [0, 3] \wedge x_0 - x_s \in [1, 3])$ that we add to the tree. We would like to point out again that time transitions from different states are not synchronized. When s_2 was created, we allowed x_0 to advance until time 3. When executing the transition to ℓ_q, we discover that it could happen "in the past", between time 2 and 3; but also that after taking the transition, we can

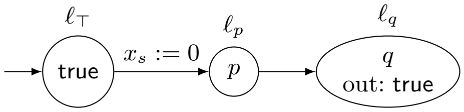

Fig. 5. Matching automaton for the expression $p \cdot q$.

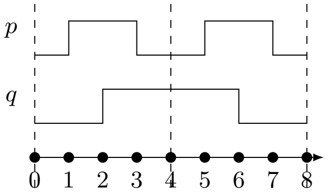

Fig. 6. Example of a signal.

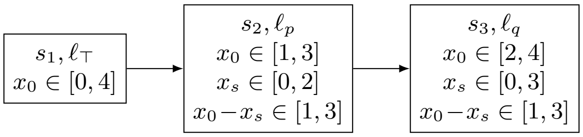

Fig. 7. Reachable symbolic states after reading the first fragment of the signal.

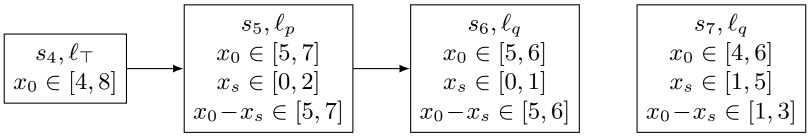

Fig. 8. Reachable symbolic states after reading the second fragment of the signal. States corresponding to the previous fragment were discarded.

stay in ℓ_q until time 4. At this point, there are no more states to explore, and we report the matches in the observed signal prefix. In this example, the matches are described by the state s_3 intersected with its output label true. Possible values of $x_0 - x_s$ in s_3 are the possible start times of the match, and possible values of x_0 are the end times. For us, the match should start between time 1 and 3, while p holds, and end between time 2 and 4, while q holds.

We now proceed to read w_2. We extract from the reachability tree all states that correspond to reading the first segment until the end, that is, states with valuations that lie on the hyperplane $x_0 = 4$. From state s_0, we extract $(\ell_\top, x_0 = 4)$. Applying time transition to it results in the state $s_4 = (\ell_\top, x_0 \in [4,8])$ that we add to the tree. From state s_3, we extract $(\ell_q, x_0 = 4 \wedge x_s \in [1,3] \wedge x_0 - x_s \in [1,3])$. Applying time transition to it results in the state $s_7 = (\ell_q, x_0 \in [4,6] \wedge x_s \in [1,5] \wedge x_0 - x_s \in [1,3])$ that we add to the tree. At this point, we can discard the previous segment of the signal and the reachability tree corresponding to it; they will no longer be used. Then, we restart the exploration from s_4 and s_7, which discovers two more states: s_5 and s_6. Both s_6 and s_7 correspond to ℓ_q and describe newly discovered matches. State s_6 corresponds to the matches that start and end between time 5 and 6, when q still holds and p holds again. State s_7 corresponds to matches that start between time 11 and 3, that is, in the previous segment) and end before time 6 in the current segment.

4 Implementation and Experiments

We implemented a prototype of the algorithm in C++, using the zone library of the tool IF [7]. We evaluate the performance of the prototype using a number of patterns and periodic signals of different length. We summarize the experimental results in Table 1. The columns "Expression" and "Signal" give the expression and the shape of the signal. Different expressions and signal shapes are discussed in more detail below. To present time-related parameters in a uniform way, we measure them in integer time units. The column "Seg" gives the length of the signal segment (in time units) that is presented at once to the reachability algorithm. We run every experiment with 2-3 segment lengths: presenting the whole signal at once ("offline") and presenting a fixed number of time units, based on the period of the signal. The last three columns show the results for different length of the signal: 10K, 100K, and 1 million time units. A cell of the table shows the run time of the matching algorithm in seconds and the number of explored symbolic states. The three parameters: signal length, signal shape, and segment length influence the performance of the algorithm in a connected way. The longer are the stable periods of the signal, the fewer switching points it has within a given length; but at the same time, if the segment length is small, longer stable periods become split into more segments. Time figures were obtained on a PC with a Core i7-3630QM and 8GB RAM.

Signals We use three different periodic signal shapes. The signal $wave_{2}$ has two components, p_0 and p_1, which are square waves with the period of 2 time units, p_1 being the negation of p_0. The signal $wave_{200}$ has four components, p_0 to p_3, which are square waves with the period of 200 time units, shifted in time by different amount. The signal $wave_{30/32}$ has two components, p_0 and p_1. The component p_0 has the period of 30 time units, in every period it has hi value for 5 time units. The component p_1 has the period of 32 time units, in every period it has hi value for 4 time units.

Simple Expressions Expressions φ_1 to φ_4 are examples of basic regular operators: concatenation, disjunction, and duration constraint.

Intersection Example Intersection allows to assert that multiple properties should hold during the same interval. To evaluate it, we use the expression

$$\varphi_5 = (\langle p\rangle_{[4,5]} \cdot \neg p) \cap (\neg q \cdot \langle q\rangle_{[4,5]}) \cap (\mathsf{true} \cdot \langle p \wedge q\rangle_{[1,2]} \cdot \mathsf{true})$$

It denotes a pattern where p holds at the beginning and between 4 to 5 time units, q holds at the end between 4 to 5 time units, and in between p and q hold together for at least 1 to 2 time units. This could be an example of one resource (such as power source) replacing another in a redundant architecture. We cannot express this property without intersection; it would require duration constraints with unbalanced parentheses [4].

Quantification Example Existential quantification allows to express synchronization with a signal which is not part of the input, but is itself described by a

Table 1. Evaluation results.

Expression	Signal	Seg	Signal length		
			10K	100K	1M
$\varphi_1 = p_0 \cdot p_1$	$wave_2$	offline	0.1s, 10K	0.96s, 100K	16s, 1M
		1	0.14s, 30K	1.4s, 300K	21s, 3M
	$wave_{200}$	signal	< 0.01s, 100	0.02s, 1K	0.1s, 10K
		100	< 0.01s, 350	0.03s, 3.5K	0.18, 35K
		25	0.02s, 2.4K	0.1s, 24K	0.84s, 240K
$\varphi_2 = \langle p_0 \rangle_{[0,20]} \cdot \langle p_1 \rangle_{[0,20]} \cdot \langle p_2 \rangle_{[0,20]}$	$wave_{200}$	offline	< 0.01s, 150	0.03s, 1.5K	0.22s, 15K
		100	< 0.01s, 450	0.06s, 4.5K	0.4s, 45K
$\varphi_3 = (p_0 \cup p_1) \cdot (p_2 \cup p_3)$	$wave_{200}$	offline	< 0.01s, 300	0.03s, 3K	0.24s, 30K
		100	0.01s, 900	0.05s, 9K	0.4s, 90K
$\varphi_4 = \langle p_1 \cdot (p_0 \cdot p_2)^+ \rangle_{[0,1000]}$	$wave_{200}$	offline	< 0.01s, 250	0.03s, 2.5K	0.26s, 25K
		100	< 0.01s, 500	0.04s, 5K	0.3s, 50K
φ_5 (see text)	$wave_{30/32}$	offline	0.02s, 800	0.1s, 8K	1s, 80K
		30	0.03s, 1.4K	0.14s, 14K	1.4s, 140K
φ_6 (prefix match, see text)	$wave_{200}$	offline	0.01s, 400	0.04, 4K	0.24s, 40K
		100	0.03s, 6.3K	2.7s, 500K	TO

regular expression. To evaluate quantification, we use the expression

$$\varphi_6 = \exists r. \big((\langle \neg r \rangle_{[98,98]} \cdot \langle r \rangle_{[1,3]})^+ \\ \cap \ (\neg p \cdot (\neg p \wedge \neg r) \cdot (p \wedge r) \cdot p \cdot (p \wedge \neg r) \cdot (\neg p \wedge r))^+ \big)$$

It denotes a signal p that changes its value on the rising edge of a virtual clock, denoted by r, that occurs every 100 ± 1 time units (note how we use $(\neg p \wedge \neg r) \cdot (p \wedge r)$ to synchronize the rising edges of p and r). In the experiments, we use this property for prefix matching. We fix the start of the match to the start of the signal and use our algorithm to find matching prefixes.

Discussion The run time of the matching algorithm is determined by the number of symbolic states that it explores, which depends on the structure of the expression, the input signal, and the way the signal is split into segments when presented to the matching algorithm. In our experiments, we focused on the case when the length of a segment given to the algorithm is greater than or equal to the length of a stable state of the signal. For example, for the signal $wave_{200}$, we normally observe two cases: when the algorithm receives the whole signal immediately, and when the signal is split into segments 100 time units in length, which is the half-period of the signal. In this setup, for a variety of regular expressions we observe two properties of the algorithm: (i) the number of explored configurations (and thus the runtime) is linear in the length of the signal; and (2) going from offline to online matching (with the length of a segment greater or equal to the length of a stable state) increases the number of explored configurations only by a small constant factor. That said, one can always come up with adversarial examples, where the match set (and thus the number of explored configurations) requres at least quadratic number of zones in the length of the signal. One way to construct adversarial examples is to synchronize the start and end of a match with some event, e.g., a raising or a falling edge. In our experiments, this happens in the property φ_6. For the signal $wave_{200}$, every sequence of one or more full signal periods is a match, and the set of all matches is described by a quadratic number of zones. For this reason, we only do prefix matching in that experiment. Another way to construct adversarial examples is to perform "oversampling" and split every stable state of the input signal into a large number of segments. As a result, every zone in the match set may be split in a quadratic number of smaller zones, since the matches that start and/or end in different segments cannot be part of the same zone in the current algorithm. We can observe this effect for the property φ_1 and the signal $wave_{200}$. Reducing the segment length from 100 to 25 time units causes oversampling and increases the number of explored configurations by a factor of $8 = 4^2/2$. In future work, we wish to address this issue, as it is reminiscent of the issue of interleaving in reachability of timed automata, which was addressed in [22].

Removing Inactive Clocks The cost of zone operations is in the worst case cubic in the number of clocks (normalization is cubic, but is not required for some operations), thus it is important to remove clocks as soon as they are no longer needed. For automata produced from TREs, this is not difficult to do, since every clock is tied to a duration constraint and thus has a clearly defined set of locations where it is active (x_0 and x_s are always active). When taking a transition, we erase (existentially quantify) clocks that are not active in the target location.

Introducing Clock Invariants The simple encoding of duration constraints that we describe may lead, during state exploration, to the creation of *doomed* symbolic states that may never lead to an accepting state. When time transition is applied to a state, we may increase a clock past a bound that will be much later checked by a guard of some transition. In the meantime, we may start exploring the successors of the doomed state, which are also doomed, then their successors, etc. To reduce the amount of such redundant work, in our implementation, we let locations have clock invariants. They are produced from the upper bounds of duration constraints and we use them to constrain the result of time transitions.

5 Conclusion

We presented a novel algorithm for timed pattern matching of Boolean signals. We are particularly interested in patterns described by timed regular expressions, but our result applies to arbitrary timed automata. The algorithm can be applied online, without restriction on how the input signal is split into incrementally presented segments. The prototype implementation shows promising results, but also points out some pessimistic scenarios. In future work, we plan to improve the performance of matching with the major goal being to improve the handling of small signal segments by adapting partial order reduction techniques; we also expect that some constant factor can be gained by improving the quality of the code. In another direction, we wish to perform a more in-depth case study to be able to adapt the algorithm to the specifics of real applications.

References

1. Aho, A.V., Hopcroft, J.E.: The design and analysis of computer algorithms. Pearson Education India (1974)
2. Aho, A.V., Kernighan, B.W., Weinberger, P.J.: The AWK programming language. Addison-Wesley Longman Publishing Co., Inc. (1987)
3. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical computer science 126(2), 183–235 (1994)
4. Asarin, E., Caspi, P., Maler, O.: A Kleene theorem for timed automata. In: Logic in Computer Science. pp. 160–171. IEEE (1997)
5. Asarin, E., Caspi, P., Maler, O.: Timed regular expressions. Journal of the ACM 49(2), 172–206 (2002)
6. Behrmann, G., David, A., Larsen, K.G., Hakansson, J., Petterson, P., Yi, W., Hendriks, M.: Uppaal 4.0. In: Quantitative Evaluation of Systems, 2006. QEST 2006. Third International Conference on. pp. 125–126. IEEE (2006)
7. Bozga, M., Fernandez, J.C., Ghirvu, L., Graf, S., Krimm, J.P., Mounier, L.: IF: An intermediate representation and validation environment for timed asynchronous systems. Formal Methods pp. 706–706 (1999)
8. Brzozowski, J.A.: Derivatives of regular expressions. Journal of the ACM (JACM) 11(4), 481–494 (1964)
9. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking (1999)
10. Daws, C., Olivero, A., Tripakis, S., Yovine, S.: The tool kronos. In: International Hybrid Systems Workshop. pp. 208–219. Springer (1995)
11. Finkel, O.: Undecidable problems about timed automata. In: Formal Modeling and Analysis of Timed Systems. pp. 187–199. Springer-Verlag (2006)
12. Gelade, W.: Succinctness of regular expressions with interleaving, intersection and counting. In: International Symposium on Mathematical Foundations of Computer Science. pp. 363–374. Springer (2008)
13. Havlicek, J., Little, S.: Realtime regular expressions for analog and mixed-signal assertions. In: Proceedings of the International Conference on Formal Methods in Computer-Aided Design. pp. 155–162. FMCAD Inc (2011)
14. Herrmann, P.: Renaming is necessary in timed regular expressions. In: International Conference on Foundations of Software Technology and Theoretical Computer Science. pp. 47–59. Springer (1999)
15. Koopman, P., Wagner, M.: Challenges in autonomous vehicle testing and validation. SAE International Journal of Transportation Safety 4(2016-01-0128), 15–24 (2016)
16. Krichen, M., Tripakis, S.: Conformance testing for real-time systems. Formal Methods in System Design 34(3), 238–304 (2009)
17. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. International Journal on Software Tools for Technology Transfer (STTT) 1(1), 134–152 (1997)
18. Larsen, K.G., Mikucionis, M., Nielsen, B.: Online testing of real-time systems using uppaal. In: FATES. pp. 79–94 (2004)
19. McNaughton, R., Yamada, H.: Regular expressions and state graphs for automata. IRE transactions on Electronic Computers (1), 39–47 (1960)
20. Pike, R.: The text editor sam. Software: Practice and Experience 17(11), 813–845 (1987)
21. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM journal of research and development 3(2), 114–125 (1959)

22. Salah, R.B., Bozga, M., Maler, O.: On interleaving in timed automata. In: CONCUR. pp. 465–476. Springer (2006)
23. Thompson, K.: Programming techniques: Regular expression search algorithm. Communications of the ACM 11(6), 419–422 (1968)
24. Tripakis, S.: Fault diagnosis for timed automata. In: Formal Techniques in Real-Time and Fault-Tolerant Systems. pp. 205–221. Springer (2002)
25. Ulus, D.: Montre: A tool for monitoring timed regular expressions. In: Computer Aided Verification. pp. 329–335. Springer (2017)
26. Ulus, D.: Pattern Matching with Time: Theory and Applications. Ph.D. thesis, University of Grenobles-Alpes (UGA) (2018)
27. Ulus, D., Ferrère, T., Asarin, E., Maler, O.: Timed pattern matching. In: Formal Modeling and Analysis of Timed Systems. pp. 222–236. Springer (2014)
28. Ulus, D., Ferrère, T., Asarin, E., Maler, O.: Online timed pattern matching using derivatives. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 736–751. Springer (2016)
29. Waga, M., Akazaki, T., Hasuo, I.: A boyer-moore type algorithm for timed pattern matching. In: Formal Modeling and Analysis of Timed Systems. pp. 121–139. Springer (2016)
30. Waga, M., Hasuo, I., Suenaga, K.: Efficient online timed pattern matching by automata-based skipping. In: Formal Modeling and Analysis of Timed Systems. pp. 224–243. Springer (2017)
31. Yovine, S.: Model checking timed automata. In: School organized by the European Educational Forum. pp. 114–152. Springer (1996)