Online Timed Pattern Matching using Derivatives

Dogan Ulus[1], Thomas Ferrère[1], Eugene Asarin[2], and Oded Maler[1]

[1] VERIMAG, Université Grenoble-Alpes/CNRS, France
[2] IRIF, Université Paris Diderot/CNRS, France

Abstract. Timed pattern matching consists in finding all segments of a dense-time Boolean signal that match a pattern defined by a timed regular expression. This problem has been formulated and solved in [17] via an offline algorithm that takes the signal and expression as inputs and produces the set of all matches, represented as a finite union of two-dimensional zones. In this work we develop an *online* version of this approach where the input signal is presented incrementally and the matching is computed incrementally as well.
Naturally, the concept of derivatives of regular expressions due to Brzozowski [6] can play a role in defining what remains to match after having read a prefix of the signal. However the adaptation of this concept is not a straightforward for two reasons: the dense infinite-state nature of timed behaviors and the fact that we are interested in matching, not only in prefix acceptance. To resolve these issues we develop an alternative theory of signals and expressions based on *absolute time* and show how derivatives are defined and computed in this setting. We then implement an online timed pattern matching algorithm based on these results.

1 Introduction

Timed regular expressions (TRE), introduced in [3, 4], constitute a formalism for expressing patterns in timed behaviors in a compact and natural way. They augment classical regular expressions with timing constraints and as such they provide an alternative specification style to real-time temporal logics such as MTL [10]. We believe that such expressions have numerous applications in many domains such as runtime verification, robotics, medical monitoring and circuit analysis [9, 7].

For a given expression φ and input signal w, *timed pattern matching* means computing the match set $\mathcal{M}(\varphi, w)$ consisting of all pairs (t, t') of time instants such that the segment of w between t and t' satisfies the expression φ. In [17] we showed how to compute $\mathcal{M}(\varphi, w)$ offline, assuming the input signal to be completely available before the matching. In this paper we develop an *online* matching algorithm where the input is presented incrementally and matches are computed on the fly. An online procedure can be used to monitor real systems during their actual executions (in contrast with monitoring simulations) and alert the user in real time. In addition, an online procedure can reduce memory requirements, discarding signals and intermediate matches when those are no longer needed.

The online pattern matching procedure that we develop in this paper is built upon the notion of derivatives of regular expressions, introduced by Brzozowski in 1964 [6].

In essence, the derivative of an expression with respect to a letter or word, tells us what remains to be observed in order to reach acceptance. In this sense it is very similar to the tableaux construction used to build automata from temporal logic formulae. Derivatives provide an elegant solution for problems of language membership [14], pattern matching [13, 16] and automaton construction [1, 5, 6] and have been observed to be naturally suitable for monitoring behaviors of systems [15, 12]. The original notion of the derivative that we recall in Section 2 is based on discrete time and requires a careful adaptation to dense time. Moreover, as we will explain, matching is more complex than acceptance (of the word or its prefixes) and this has some implications on associating derivatives with rewrite rules.

In Section 3 we modify the definition of signals, one of the commonly-used formalisms to express timed behaviors, so as to lift the theory of derivatives to the timed setting. Signals (and sequences) are traditionally defined to start at time zero and when two signals are concatenated as in $w = u \cdot v$, the second argument v is shifted forward in time, to start at the end of u. In contrast, we define signals in absolute time, each having its own fixed starting point. In this setting concatenation becomes a *partial* function, defined only when the domains of definition of the two signals fit. We also introduce a special place holder symbol ✓ and define extended signals where all letters in some prefix have been replaced by this symbol.

We then adapt timed regular expressions to represent sets of extended signals using the absolute time semantics. The regular expressions of [3, 4, 17] are obtained as a syntactic sub-class denoting "pure" ✓-free signals, used for the initial specification. The more general expressions are used to represent intermediate stages during the incremental computation of the match set.

In Section 4 we introduce our main technical contribution: the definition and computation of derivatives of left-reduced timed regular expressions with respect to a constant signal of arbitrary duration and all its factors. We apply this result to solve the problem of online timed pattern matching in Section 5 where we observe an input signal consisting of a finite concatenation of constant signals. We give a complete example of a run of our algorithm and briefly mention our implementation and its performance.

2 Preliminaries

Let Σ^* be the set of all finite words over alphabet Σ with ϵ denoting the empty word. A language $\mathcal{L}$ over Σ is a subset of Σ^*. The syntax of regular expressions over Σ is given by the following grammar:

$$r := \varnothing \mid \epsilon \mid a \mid r_1 \cdot r_2 \mid r_1 \vee r_2 \mid r^*$$

where $a \in \Sigma$. A regular expression r specifies a regular language $[\![r]\!]$, inductively defined as follows:

$$\begin{array}{rlrl} [\![\varnothing]\!] &= \emptyset & [\![r_1 \cdot r_2]\!] &= [\![r_1]\!] \cdot [\![r_2]\!] \\ [\![\epsilon]\!] &= \{\epsilon\} & [\![r_1 \vee r_2]\!] &= [\![r_1]\!] \cup [\![r_2]\!] \\ [\![a]\!] &= \{a\} & [\![r^*]\!] &= [\![r]\!]^* \end{array}$$

In some cases it is important to determine whether or not the language of a regular expression r contains the empty word ϵ. For this purpose an empty word extraction function ν (also known as the nullability predicate) is defined such as

$$\nu(r) = \begin{cases} \epsilon & \text{if } \epsilon \in [\![r]\!] \\ \varnothing & \text{otherwise} \end{cases}$$

This function which extracts ϵ from r if it exists, is computed inductively by the following rules:

$$\begin{array}{rr} \nu(\varnothing) = \varnothing & \nu(r_1 \cdot r_2) = \nu(r_1) \cdot \nu(r_2) \\ \nu(\epsilon) = \epsilon & \nu(r_1 \vee r_2) = \nu(r_1) \vee \nu(r_2) \\ \nu(a) = \varnothing & \nu(r^*) = \epsilon \end{array}$$

Definition 1 (Derivative). *The derivative of a language $\mathcal{L}$ with respect to a word u is defined as*

$$D_u(\mathcal{L}) := \{\, v \in \Sigma^* \mid u \cdot v \in \mathcal{L} \}.$$

In [6] Brzozowski applied the notion of derivatives to regular expressions and proved that the derivative $D_a(r)$ of an expression r with respect to a letter a can be computed recursively using the following syntactic rewrite rules:

$$\begin{array}{rr} D_a(\varnothing) = \varnothing & D_a(r_1 \cdot r_2) = D_a(r_1) \cdot r_2 \ \vee \ \nu(r_1) \cdot D_a(r_2) \\ D_a(\epsilon) = \varnothing & D_a(r_1 \vee r_2) = D_a(r_1) \vee D_a(r_2) \\ D_a(a) = \epsilon & D_a(r^*) = D_a(r) \cdot r^* \\ D_a(b) = \varnothing & \end{array}$$

These rules are extended for words by letting $D_{a \cdot w}(r) = D_w(D_a(r))$. By definition, membership $w \in \mathcal{L}$ is equivalent to $\epsilon \in D_w(\mathcal{L})$. Hence to check, for example, whether abc is in the language of the expression $\varphi = a^* \cdot (b \cdot c)^*$ we compute $D_{abc}(\varphi) = D_c(D_b(D_a(\varphi)))) = (b \cdot c)^*$ as follows:

$$a^* \cdot (b \cdot c)^* \xrightarrow[D_a]{} a^* \cdot (b \cdot c)^* \xrightarrow[D_b]{} c \cdot (b \cdot c)^* \xrightarrow[D_c]{} (b \cdot c)^*,$$

and since $\nu((b \cdot c)^*) = \epsilon$, $abc \in [\![\varphi]\!]$.

It is of course not a coincidence that this procedure resembles the reading of the word by an automaton where derivatives correspond to states and those that contain ϵ correspond to accepting states. Hence we can report membership in $[\![\varphi]\!]$ of w as well as the membership of all its prefixes. We can do it incrementally as new letters arrive.

Matching is more involved as we are interested in the membership of all factors of w, starting at arbitrary positions. Thus, having read j letters of w, the state of a matching algorithm should contain all the derivatives by $w[i..j]$, $i \leq j$. When letter $j + 1$ is read, these derivatives are updated to become derivatives by $w[i..j + 1]$, new matches are extracted and a new process for matches that start at $j + 1$ is spawned. Table 2 illustrates the systematic application of derivatives to find segments of $w = abcbc$ that match $\varphi = a^* \cdot (b \cdot c)^*$. The table is indexed by the start position (rows) and end position (columns) of the segments with respect to which we derive. Derivatives that contain ϵ correspond to matches and their time indices constitute the match

set $\{(1,1),(1,3),(1,5),(2,3),(2,5),(4,5)\}$. In a discrete finite-state setting there are finitely many such derivatives but this is not the case for timed systems.[1]

Symbols		a	b	c	b	c
Positions		1	2	3	4	5
1	$\varphi \xrightarrow[D_a]{}$	$a^* \cdot (b \cdot c)^*$	$\xrightarrow[D_b]{} c \cdot (b \cdot c)^* \xrightarrow[D_c]{}$	$(b \cdot c)^*$	$\xrightarrow[D_b]{} c \cdot (b \cdot c)^* \xrightarrow[D_c]{}$	$(b \cdot c)^*$
2		φ	$\xrightarrow[D_b]{} c \cdot (b \cdot c)^* \xrightarrow[D_c]{}$	$(b \cdot c)^*$	$\xrightarrow[D_b]{} c \cdot (b \cdot c)^* \xrightarrow[D_c]{}$	$(b \cdot c)^*$
3			$\varphi \xrightarrow[D_c]{}$	$\varnothing$	$\xrightarrow[D_b]{} \varnothing \xrightarrow[D_c]{}$	$\varnothing$
4				φ	$\xrightarrow[D_b]{} c \cdot (b \cdot c)^* \xrightarrow[D_c]{}$	$(b \cdot c)^*$
5					$\varphi \xrightarrow[D_c]{}$	$\varnothing$

Table 1. Pattern matching using derivatives for $w = abcbc$ and $\varphi = a^* \cdot (b \cdot c)^*$. Entry (i, j) represents the derivative with respect to $w[i, j]$. Derivatives containing ϵ are shaded with green. The state of an online matching algorithm after reading j symbols is represented in column j.

In dense time, the analogue of the arrival of a new letter is the arrival of a constant segment of the signal $w[t_1, t_2]$. When this occurs, the state of the algorithm should be updated to capture all derivatives by segments of the form $w[t, t_2]$ for $t < t_2$ and all matches ending in some $t < t_2$ should be extracted. The technique for representing and manipulating such an uncountable number of derivative together with their corresponding time segments is the main contribution of this paper.

3 Signals, Timed Languages and Expressions

We consider an alphabet $\Sigma = \mathbb{B}^m$ which is the set of valuations of a set of propositional variables $P = \{p_1 \ldots, p_m\}$. We define signals not as free floating objects but anchor them in absolute time.

Definition 2 (Signals). *A signal over an alphabet Σ is a piecewise-constant function $w : [t_1, t_2) \longrightarrow \Sigma$, where $t_1 \leq t_2 \in \mathbb{R}_{\geq 0}$ and w admits a finite number of discontinuities. The time domain of the signal and its beginning and end times are denoted as*

$$\mathrm{dom}(w) = [t_1, t_2) = [\tau_1(w), \tau_2(w)).$$

The empty signal ϵ is the unique signal satisfying $\mathrm{dom}(w) = \emptyset$. The duration of w is $|w| = \tau_2(w) - \tau_1(w)$ and $|\epsilon| = 0$. We often view the boundary points of a signal as a pair, $\tau(w) = (\tau_1(w), \tau_2(w))$.

[1] To keep the survey within a reasonable size and avoid tedious repetitions, the description here is not fully rigorous, using the same notation for the *semantic* notion of a left quotient, which is unique for every language and word, and the *syntactic* notion of a derivative of a regular expression. The derivation of the minimal automaton from a regular expression, for example, requires additional rewrite rules to detect equivalence between different regular expressions.

We use $w[t,t']$ to denote the restriction of w to an interval $[t,t') \subseteq \mathit{dom}(w)$ and let $\mathit{Sub}(w) = \{w[t,t'] \mid \tau_1(w) \leq t < t' \leq \tau_2(w)\}$ be the set of sub-signals (factors, segments) of w. Concatenation is restricted to signals that meet, that is, one ends where the other starts.

Definition 3 (Meets and Concatenation). *Signal w_1 meets signal w_2 when $w_1 = \epsilon$ or $w_2 = \epsilon$ or $\tau_2(w_1) = \tau_1(w_2)$. Concatenation is a partial function such that $w_1 \cdot w_2$ is defined only if w_1 meets w_2:*

$$w_1 \cdot w_2(t) = \begin{cases} w_1(t) & \text{if } t \in \mathrm{dom}(w_1) \\ w_2(t) & \text{if } t \in \mathrm{dom}(w_2) \end{cases}$$

The empty signal ϵ is the neutral element for concatenation: $\epsilon \cdot w = w \cdot \epsilon = w$. The set of signals thus defined can be made a monoid by making concatenation total by introducing a new element $\perp$ and letting $w_1 \cdot w_2 = \perp$ when the signals do not meet. The newly introduced element is an absorbing zero satisfying $\perp \cdot w = w \cdot \perp = \perp$.

The variability (logical length) of a signal w is the minimal n such that w can be written as $w = w_1 \cdot w_2 \cdots w_n$ where each w_i is a constant signal. We use notations $\Sigma^{(*)}$, $\Sigma^{(+)}$ and $\Sigma^{(n)}$ to denote the set of all signals, non-empty signals and signals of variability n, respectively. In particular, $\Sigma^{(1)}$ is the set of all constant signals. Sets of signals are referred to as signal languages on which Boolean operations as well as concatenation and star are defined naturally. Finally we extend the *time restriction* operation of [4] which constrains the duration of signals, to apply also to their time domains. The language ${}^{K}_{J}\langle\mathcal{L}\rangle_{I}$ where I, J, K are intervals of non-negative reals, is a subset of $\mathcal{L}$ consisting of signals with duration in I, beginning in $t_1 \in J$ and ending in $t_2 \in K$. We omit the corresponding interval when there is no restriction on beginning, ending or duration.

We are interested in representing a family of sub-signals of a n-variability signal $w = w_1 \dots w_n$ starting in segment i and ending in segment j, that is, $\mathit{Sub}_{[i:j]}(w) := \{w[t,t'] \mid t \in \mathit{dom}(w_i) \text{ and } t' \in \mathit{dom}(w_j)\}$. It can be easily verified that

$$\mathit{Sub}_{[i:j]}(w) = \mathit{Sub}(w_i) \cdot w_{i+1} \cdots \mathit{Sub}(w_j) = \mathit{Sub}(w_i) \cdot \mathit{Sub}(w_{i+1}) \cdots \mathit{Sub}(w_j).$$

In the classical discrete setting, the derivative D_a is associated with a rewrite rule $a \to \epsilon$ and a word w is accepted if it can be transformed into ϵ by successive rewritings. For the purpose of timed matching we need a more length-preserving view where reading a corresponds to a rule $a \to \checkmark$ where $\checkmark$ is a special place-holder that indicates that a has been processed. Acceptance then corresponds to the rewriting of w into a signal $w' : \mathit{dom}(w) \mapsto \checkmark$. We let $\Sigma_\checkmark = \Sigma \cup \{\checkmark\}$ and define extended signals which are signals over $\Sigma_\checkmark$, as well as some subclasses of those.

Definition 4 (Extended Signals). *An extended signal over alphabet Σ is a function $w : [t,t') \to \Sigma_\checkmark$. An extended signal w is left-reduced if $w \in \checkmark^{(*)} \cdot \Sigma^{(*)}$. A left-reduced signal w is* pure *if $w \in \Sigma^{(*)}$ and* reduced *if $w \in \checkmark^{(*)}$.*

We use initial Greek letters to denote reduced signals and hence a left-reduced signal w will be written as $w = \alpha \cdot v$ where α is a reduced signal and v is a pure signal.

Definition 5 (Left Reduction). *A reduction rule $R(u)$ for a signal $u \in \Sigma^{(*)}$ is a pair (u, γ) such that $\gamma \in \checkmark^{(*)}$ and $\mathrm{dom}(u) = \mathrm{dom}(\gamma)$. The left reduction of a left-reduced signal language $\mathcal{L}$ with respect to u is:*

$$\delta_u(\mathcal{L}) := \{\ \alpha\gamma w \mid \alpha u w \in \mathcal{L},\ \alpha \in \checkmark^{(*)} \text{ and } w \in \Sigma^{(*)}\}$$

We use operation $\delta_u(\mathcal{L})$ in a similar way $D_u(\mathcal{L})$ is used in the classical setting but with one important difference. When $v = D_u(w)$ the length of the word is reduced, that is, $|v| = |w| - |u|$, while when $v = \delta_u(w)$ the domains (and hence durations) of v and w are the same. Consequently, unlike the classical case where membership of w in $\mathcal{L}$ amounts to $\epsilon \in D_w(\mathcal{L})$, here the membership is equivalent to $\gamma \in \delta_w(\mathcal{L})$ where γ is a reduced signal of the same domain as w. It is not difficult to check that $\delta_{u_1 \cdot u_2}(\mathcal{L}) = \delta_{u_2}(\delta_{u_1}(\mathcal{L}))$ and sometimes we denote by δ_S the left reduction with respect to a set of signals.

Example 1. Consider a signal language $\mathcal{L} = \{w_1, w_2\}$ such that

$$w_1(t) = \begin{cases} a & \text{if } t \in [0,3) \\ b & \text{if } t \in [3,5) \end{cases} \qquad w_2(t) = \begin{cases} a & \text{if } t \in [0,2) \\ b & \text{if } t \in [2,5) \end{cases}$$

In Figure 1 we illustrate a left reduction operation $\delta_{u_3}(\delta_{u_2}(\delta_{u_1}(\mathcal{L}))) = \{w_1'''\}$ with respect to $u = u_1u_2u_3$ with $u_1 : [0,1) \mapsto a$, $u_2 : [1,3) \mapsto a$ and $u_3 : [3,5) \mapsto b$. Since w_1''' is a reduced signal and $\tau(u) = \tau(w_1''')$, $u \in \mathcal{L}$.

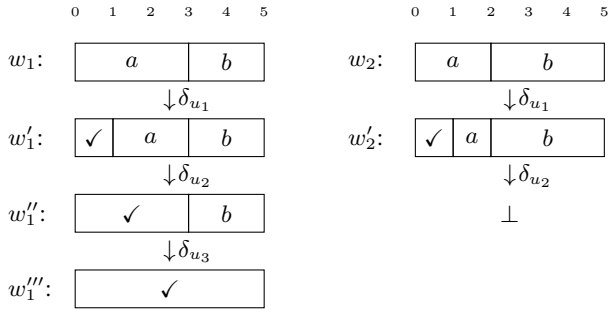

Fig. 1. A left reduction example.

We now introduce timed regular expressions to describe sets of signals and extended signals using the absolute time semantics. Note that the intersection operator, which is considered a syntactic sugar in the classical theory, adds expressiveness in the timed setting [4].

Definition 6 (Extended Timed Regular Expressions). *Extended timed regular expressions are defined by the following grammar:*

$$\varphi := \varnothing \mid \epsilon \mid p \mid \checkmark \mid \varphi_1 \cdot \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi^* \mid {}^{K}_{J}\langle\varphi\rangle_I$$

where p is a proposional variable in P and I, J, K are intervals of $\mathbb{R}_{\geq 0}$.

The semantics of the expressions is defined by the following rules (we use $a \models p$ to denote the fact that p holds at a):

$$
\begin{aligned}
[\![\varnothing]\!] &= \emptyset \\
[\![\epsilon]\!] &= \{\epsilon\} \\
[\![p]\!] &= \{w : [t,t') \to \Sigma \mid 0 \le t < t' \text{ and } \forall t'' \in [t,t').\ w(t'') \models p\} \\
[\![\checkmark]\!] &= \{w : [t,t') \to \{\checkmark\} \mid 0 \le t < t'\} \\
[\![\varphi \cdot \psi]\!] &= [\![\varphi]\!] \cdot [\![\psi]\!] \\
[\![\varphi \vee \psi]\!] &= [\![\varphi]\!] \cup [\![\psi]\!] \\
[\![\varphi \wedge \psi]\!] &= [\![\varphi]\!] \cap [\![\psi]\!] \\
[\![\varphi^*]\!] &= \textstyle\bigcup_{i=0}^{\infty} [\![\varphi]\!]^i \\
[\![{}^K_J\langle\varphi\rangle_I]\!] &= \{w \mid w \in [\![\varphi]\!],\ |w| \in I,\ w \neq \epsilon \to (\tau_1(w) \in J \wedge \tau_2(w) \in K)\}
\end{aligned}
$$

A signal language is regular if it can be represented by a timed regular expression.

The syntax in Definition 6 allows to define sets including extended signals with arbitrary interleavings of letters and ✓. Below we define three syntactic classes of expressions. The first class, called pure (or original) timed regular expressions, corresponds almost the same syntax of expressions seen in [3, 4, 17]. Pure expressions are ✓-free and do not place any restriction on the absolute beginning and ending values over their sub-expressions. The second class is reduced timed regular expressions which is formed using the ✓symbol only. Lastly we have left-reduced timed regular expressions, obtained as compositions of reduced and pure expressions satisfying some conditions.

Definition 7 (Syntactic Classes). *A timed regular expression φ belongs to the classes of* reduced*,* pure *or* left-reduced *timed regular expressions if functions $r?$, $p?$ or $lr?$, respectively, evaluate to* true *in the following table.*

Case	*Reduced* $r?(\varphi)$	*Pure* $p?(\varphi)$	*Left-reduced* $lr?(\varphi)$
$\varnothing$	$\top$	$\top$	$\top$
ϵ	$\top$	$\top$	$\top$
p	$\bot$	$\top$	$\top$
$\checkmark$	$\top$	$\bot$	$\top$
$\varphi_1 \cdot \varphi_2$	$r?(\varphi_1) \wedge r?(\varphi_2)$	$p?(\varphi_1) \wedge p?(\varphi_2)$	$lr?(\varphi_1) \wedge p?(\varphi_2) \vee r?(\varphi_1) \wedge lr?(\varphi_2)$
$\varphi_1 \wedge \varphi_2$	$r?(\varphi_1) \wedge r?(\varphi_2)$	$p?(\varphi_1) \wedge p?(\varphi_2)$	$lr?(\varphi_1) \wedge lr?(\varphi_2)$
$\varphi_1 \vee \varphi_2$	$r?(\varphi_1) \wedge r?(\varphi_2)$	$p?(\varphi_1) \wedge p?(\varphi_2)$	$lr?(\varphi_1) \wedge lr?(\varphi_2)$
φ^*	$r?(\varphi)$	$p?(\varphi)$	$r?(\varphi) \vee p?(\varphi)$
${}^K_J\langle\varphi\rangle_I$	$r?(\varphi)$	$p?(\varphi) \wedge J = K = [0,\infty)$	$lr?(\varphi)$

Trivially any reduced expression ψ and any pure expression φ represent reduced and pure signal languages such that $[\![\psi]\!] \subseteq \checkmark^{(*)}$ and $[\![\varphi]\!] \subseteq \Sigma^{(*)}$. For left-reduced expressions we do not allow concatenation and star operations on arbitrary left-reduced expressions as in Definition 7 because left-reduced languages are not closed under concatenation. By doing that we have the following result.

Proposition 1. *The language $[\![\varphi]\!]$ of a left-reduced timed regular expression φ is an extended signal language such that $[\![\varphi]\!] \subseteq \checkmark^{(*)} \cdot \Sigma^{(*)}$.*

Proof. For the concatenation $\varphi_1 \cdot \varphi_2$ we have two possibilities: (1) $[\![\varphi_1]\!] \subset \checkmark^{(*)} \cdot \Sigma^{(*)}$ and $[\![\varphi_2]\!] \subset \Sigma^{(*)}$; (2) $[\![\varphi_1]\!] \subset \checkmark^{(*)}$ and $[\![\varphi_2]\!] \subset \checkmark^{(*)} \cdot \Sigma^{(*)}$. For both possibilities, we have $[\![\varphi_1 \cdot \varphi_2]\!] = [\![\varphi_1]\!] \cdot [\![\varphi_2]\!] \subset \checkmark^{(*)} \cdot \Sigma^{(*)}$. Other cases are straightforward by following the definitions.

A comprehensive study on regular algebra extended with intersection operation can be found in [2]. We now mention some algebraic rules relative to the time restriction operator. It is shown in [17] how the right hand side of following equations can be computed from the corresponding left hand side.

$$ {}^{K_1}_{J_1}\langle\checkmark\rangle_{I_1} \cdot {}^{K_2}_{J_2}\langle\checkmark\rangle_{I_2} = {}^{K_3}_{J_3}\langle\checkmark\rangle_{I_3} \quad \text{and} \quad {}^{K_1}_{J_1}\langle\checkmark\rangle_{I_1} \wedge {}^{K_2}_{J_2}\langle\checkmark\rangle_{I_2} = {}^{K_3}_{J_3}\langle\checkmark\rangle_{I_3} $$

for some intervals I_3, J_3 and K_3, and

$$ (\bigvee_{i=1}^{m} {}^{K_i}_{J_i}\langle\checkmark\rangle_{I_i})^{+} = \bigvee_{i=1}^{n} {}^{K'_i}_{J'_i}\langle\checkmark\rangle_{I'_i} \quad \text{for some } m, n \in \mathbb{N} $$

Therefore we can simplify timed regular expressions further using these equations and procedures.

4 Derivatives of Left-Reduced Timed Regular Expressions

We now introduce, semantically and syntactically, a derivative operation for left-reduced signal languages and expressions based on the left reduction operation. Since our goal is to solve the dense time matching problem, we have to operate on sets of signals and define derivatives more symbolically. Therefore we define the derivative Δ_v to correspond to the left reduction with respect to all factors of v.

Definition 8 (Dense Derivative). *The derivative $\Delta_v(\mathcal{L})$ of a left-reduced language $\mathcal{L}$ with respect to a constant signal $v \in \Sigma^{(1)}$ is defined as follows:*

$$ \Delta_v(\mathcal{L}) := \bigcup_{u \in \mathit{Sub}(v)} \delta_u(\mathcal{L}) $$

As mentioned previously, reduced signals will provide the output of our matching procedure. Their existence will be the witness of a match and their time domains will indicate its position in the signal.

Definition 9 (Extraction). *The extraction* $\mathrm{xt}(\mathcal{L})$ *of a left-reduced signal language $\mathcal{L}$ is*

$$ \mathrm{xt}(\mathcal{L}) := \{\, \alpha \mid \alpha \in \checkmark^{(*)} \cap \mathcal{L} \} $$

The following result shows that xt can be computed syntactically for left-reduced timed regular expressions.

Theorem 1 (Extraction Computation). *For a given left-reduced timed regular expression φ, applying the following rules recursively yields an expression ψ such that $[\![\psi]\!] = \mathrm{xt}([\![\varphi]\!])$.*

$$
\begin{aligned}
\mathrm{xt}(\varnothing) &= \varnothing & \mathrm{xt}(\psi_1 \cdot \psi_2) &= \mathrm{xt}(\psi_1) \cdot \mathrm{xt}(\psi_2)\\
\mathrm{xt}(\epsilon) &= \epsilon & \mathrm{xt}(\psi_1 \vee \psi_2) &= \mathrm{xt}(\psi_1) \vee \mathrm{xt}(\psi_2)\\
\mathrm{xt}(p) &= \varnothing & \mathrm{xt}(\psi_1 \wedge \psi_2) &= \mathrm{xt}(\psi_1) \wedge \mathrm{xt}(\psi_2)\\
\mathrm{xt}(\checkmark) &= \checkmark & \mathrm{xt}({}^{K}_{J}\langle\psi\rangle_{I}) &= {}^{K}_{J}\langle\mathrm{xt}(\psi)\rangle_{I}\\
& & \mathrm{xt}(\psi^*) &= (\mathrm{xt}(\psi))^*
\end{aligned}
$$

Proof. We proceed by induction and only look at the case of concatenation, other cases are similar. For any expressions φ_1, φ_2 it holds

$$
\begin{aligned}
[\![\mathrm{xt}(\varphi_1 \cdot \varphi_2)]\!] &= \{\alpha \mid \alpha \in \checkmark^{(*)} \text{ and } \alpha \in [\![\varphi_1 \cdot \varphi_2]\!]\}\\
&= \{\alpha_1\alpha_2 \mid \alpha_1, \alpha_2 \in \checkmark^{(*)},\ \alpha_1 \in [\![\varphi_1]\!] \text{ and } \alpha_2 \in [\![\varphi_2]\!]\}\\
&= \{\alpha_1 \mid \alpha_1 \in \checkmark^{(*)} \text{ and } \alpha_1 \in [\![\varphi_1]\!]\} \cdot \{\alpha_2 \mid \alpha_2 \in \checkmark^{(*)} \text{ and } \alpha_2 \in [\![\varphi_2]\!]\}\\
&= [\![\mathrm{xt}(\varphi_1)]\!] \cdot [\![\mathrm{xt}(\varphi_2)]\!]
\end{aligned}
$$

Example 2. Consider a left-reduced expression $\varphi := \left\langle {}^{[0,3]}_{[0,3]}\langle\checkmark\rangle_{[0,3]} \cdot p^* \right\rangle_{[0,2]}$. Applying Theorem 1 we extract from φ a reduced expression ψ such that $\psi = \left\langle {}^{[0,3]}_{[0,3]}\langle\checkmark\rangle_{[0,3]} \right\rangle_{[0,2]}$. Expression ψ can be simplified further to ${}^{[0,3]}_{[0,3]}\langle\checkmark\rangle_{[0,2]}$.

We now state our main result concerning derivatives of left-reduced timed regular expressions.

Theorem 2 (Derivative Computation). *Given a left-reduced timed regular expression φ and a constant signal $v : [t, t') \mapsto a$, applying the following rules yields an expression ψ such that $[\![\psi]\!] = \Delta_v([\![\varphi]\!])$.*

$$
\begin{aligned}
\Delta_v(\varnothing) &= \varnothing\\
\Delta_v(\epsilon) &= \varnothing\\
\Delta_v(\checkmark) &= \varnothing\\
\Delta_v(p) &= \begin{cases} \Gamma \vee \Gamma \cdot p & \text{if } a \models p \text{ where } \Gamma := {}^{[t,t']}_{[t,t']}\langle\checkmark\rangle_{[0,t'-t]} \\ \varnothing & \text{otherwise} \end{cases}\\
\Delta_v(\psi_1 \cdot \psi_2) &= \Delta_v(\psi_1) \cdot \psi_2 \ \vee\ \mathrm{xt}\big(\psi_1 \vee \Delta_v(\psi_1)\big) \cdot \Delta_v(\psi_2)\\
\Delta_v(\psi_1 \vee \psi_2) &= \Delta_v(\psi_1) \vee \Delta_v(\psi_2)\\
\Delta_v(\psi_1 \wedge \psi_2) &= \Delta_v(\psi_1) \wedge \Delta_v(\psi_2)\\
\Delta_v({}^{K}_{J}\langle\psi\rangle_{I}) &= {}^{K}_{J}\langle\Delta_v(\psi)\rangle_{I}\\
\Delta_v(\psi^*) &= \mathrm{xt}(\Delta_v(\psi))^* \cdot \Delta_v(\psi) \cdot \psi^*
\end{aligned}
$$

Proof. By semantic definition $\Delta_v(\varphi) = \{\ \alpha\gamma w \mid \alpha u w \in [\![\varphi]\!] \text{ and } (u, \gamma) \in \mathrm{RSub}(v)\}$ where $\mathrm{RSub}(v) := \{\ R(u) \mid u \in \mathrm{Sub}(v)\}$. We proceed by induction on the structure of φ. In the following we tend to use languages and expressions interchangeably, when in the interest of readability. Consider the cases:

- For $\varphi = \varnothing$, $\varphi = \epsilon$ and $\varphi = \checkmark$: for all cases $\alpha u w \notin [\![\varphi]\!]$ therefore $\Delta_v(\varphi) = \varnothing$.

- For $\varphi = p$: It needs that $\alpha = \epsilon$ and $u \in [\![p]\!]$. Then, $\alpha uw \in [\![p]\!]$ can be satisfied if either $w = \epsilon$ or $w \in [\![p]\!]$. By applying definitions, we get

$$\begin{aligned}\Delta_v(p) &= \{\, \gamma \mid u \in [\![p]\!] \text{ and } (u,\gamma) \in \text{RSub}(v)\} \cup \\ &\qquad \{\, \gamma w \mid u \in [\![p]\!],\ w \in [\![p]\!] \text{ and } (u,\gamma) \in \text{RSub}(v)\} \\ &= \Gamma \vee \Gamma \cdot \{w \mid w \in [\![p]\!]\} \\ &= \Gamma \vee \Gamma \cdot p\end{aligned}$$

  where the expression Γ is ${}^{[t,t']}_{[t,t']}\langle\checkmark\rangle_{[0,t'-t]}$. Hence, we have $\Delta_v(p) = \Gamma \vee \Gamma \cdot p$ if $u \in [\![p]\!]$, otherwise $\Delta_v(p) = \varnothing$. The condition $u \in [\![p]\!]$ can be easily checked by testing $a \models p$.
- For $\varphi = \varphi_1 \cdot \varphi_2$: $\alpha uw \in [\![\varphi_1 \cdot \varphi_2]\!]$ should be satisfied. There are three possibilities to split αuw in dense time:
  - It can be split up into $\alpha uw_1 \in [\![\varphi_1]\!]$ and $w_2 \in [\![\varphi_2]\!]$.

$$\begin{aligned}\Delta_v(\varphi) &= \{\alpha\gamma w_1 w_2 \mid \alpha u w_1 \in [\![\varphi_1]\!],\ w_2 \in [\![\varphi_2]\!] \text{ and } (u,\gamma) \in \text{RSub}(v)\} \\ &= \{\alpha\gamma w_1 \mid \alpha u w_1 \in [\![\varphi_1]\!] \text{ and } (u,\gamma) \in \text{RSub}(v)\} \cdot \{w_2 \mid w_2 \in [\![\varphi_2]\!]\} \\ &= \Delta_v(\varphi_1) \cdot \varphi_2\end{aligned}$$

  - It can be split up into $\alpha_1 \in [\![\varphi_1]\!]$ and $\alpha_2 uw \in [\![\varphi_2]\!]$.

$$\begin{aligned}\Delta_v(\varphi) &= \{\alpha_1\alpha_2\gamma w \mid \alpha_1 \in [\![\varphi_1]\!],\ \alpha_2 uw \in [\![\varphi_2]\!] \text{ and } (u,\gamma) \in \text{RSub}(v)\} \\ &= \{\alpha_1 \mid \alpha_1 \in [\![\varphi_1]\!]\} \cdot \{\alpha_2\gamma w \mid \alpha_2 uw \in [\![\varphi_2]\!] \text{ and } (u,\gamma) \in \text{RSub}(v)\} \\ &= \text{xt}(\varphi_1) \cdot \Delta_v(\varphi_2)\end{aligned}$$

  - It can be split up into $\alpha u_1 \in [\![\varphi_1]\!]$ and $u_2 w \in [\![\varphi_2]\!]$. For this case, it is required by definitions that φ_1 is a left-reduced expression and φ_2 is a pure expression. This is the most involved case requiring to split reducing signals.

$$\begin{aligned}\Delta_v(\varphi) &= \{\alpha\gamma_1\gamma_2 w \mid \alpha u_1 \in [\![\varphi_1]\!],\ u_2 w \in [\![\varphi_2]\!] \text{ and } (u_1u_2, \gamma_1\gamma_2) \in \text{RSub}(v)\} \\ &= \{\alpha\gamma_1\gamma_2 w \mid \alpha u_1 \in [\![\varphi_1]\!],\ u_2 w \in [\![\varphi_2]\!],\ (u_1,\gamma_1) \in \text{RSub}(v), \\ &\qquad (u_2,\gamma_2) \in \text{RSub}(v) \text{ and } (u_1,\gamma_1) \text{ meets } (u_2,\gamma_2)\} \\ &= \{\alpha\gamma_1 \mid \alpha u_1 \in [\![\varphi_1]\!] \text{ and } (u_1,\gamma_1) \in \text{RSub}(v)\} \cdot \\ &\qquad \{\gamma_2 w \mid u_2 w \in [\![\varphi_2]\!] \text{ and } (u_2,\gamma_2) \in \text{RSub}(v)\} \\ &= \text{xt}(\Delta_v(\varphi_1)) \cdot \Delta_v(\varphi_2)\end{aligned}$$

  Thus $\Delta_v(\varphi_1 \cdot \varphi_2)$ can be found by the disjunction of these three cases. Then, by rearranging the last two cases, we obtain the equality claimed in the theorem.
- For $\varphi = \psi^*$: assume without loss of generality $\epsilon \notin \psi$. Then

$$\begin{aligned}\Delta_v(\psi^*) &= \Delta_v(\epsilon) \vee \Delta_v(\psi \cdot \psi^*) \\ &= \Delta_v(\psi) \cdot \psi^* \ \vee\ \text{xt}(\psi) \cdot \Delta_v(\psi^*) \ \vee\ \text{xt}(\Delta_v(\psi)) \cdot \Delta_v(\psi^*) \\ &= \Delta_v(\psi) \cdot \psi^* \ \vee\ \text{xt}(\Delta_v(\psi)) \cdot \Delta_v(\psi^*) \\ &= [\epsilon \ \vee\ X \ \vee\ X^2 \ \vee \cdots \vee\ X^\infty] \cdot \Delta_v(\psi) \cdot \psi^* \text{ where } X = \text{xt}(\Delta_v(\psi)) \\ &= \text{xt}(\Delta_v(\psi))^* \cdot \Delta_v(\psi) \cdot \psi^*\end{aligned}$$

- Time restriction and Boolean operations follow definitions straightforwardly.

Corollary 1. *The derivative $\Delta_v(\varphi)$ of a left-reduced timed regular expression φ with respect to a constant signal v is a left-reduced timed regular expression.*

Proof. Theorem 2 shows that only finite number of regular operations is required to find the derivative and these equations satisfy requirements in Definition 7.

We extend derivatives for arbitrary signals by letting $\Delta_\epsilon(\varphi) = \varphi$ and

$$\Delta_{v\cdot w}(\varphi) = \Delta_w(\Delta_v(\varphi)).$$

Lemma 1. *The derivative $\Delta_w(\varphi)$ of a left-reduced timed regular expression φ with respect to a signal $w = w_1 \dots w_n$ with n segments is equivalent to the left reduction of φ with respect to the set of sub-signals of w beginning in dom(w_1) and ending in dom(w_n).*

$$\Delta_w(\varphi) = \bigcup_{u \in \mathit{Sub}_{[1:n]}(w)} \delta_u([\![\varphi]\!])$$

Proof. Using definitions we directly have

$$\begin{aligned}
\Delta_w(\varphi) &= \Delta_{w_n}(\Delta_{w_{n-1}}(\dots(\Delta_{w_1}(\varphi)))) \\
&= \delta_{\mathsf{Sub}(w_n)}(\delta_{\mathsf{Sub}(w_{n-1})}(\dots(\delta_{\mathsf{Sub}(w_1)}([\![\varphi]\!])))) \\
&= \delta_{\mathsf{Sub}(w_1)\cdot\mathsf{Sub}(w_2)\dots\mathsf{Sub}(w_n)}([\![\varphi]\!]) \\
&= \delta_{\mathsf{Sub}_{[1:n]}}([\![\varphi]\!])
\end{aligned}$$

5 Application to Online Timed Pattern Matching

In this section we solve the problem of online timed pattern matching by applying concepts and results introduced in previous sections. Our online matching procedure assumes the input signal w to be presented incrementally as follows. Let $w = w_1 w_2 \dots w_n$ be an n-variability signal and at each step j we read a new segment $w_j : [t_j, t'_j) \mapsto a_j$ where $a_j \in \Sigma$. After reading a new segment w_j we may have new matches ending in dom(w_j) in addition to previously found matches. Therefore we define an incremental match set $\mathcal{M}_j(\varphi, w)$ consisting of matches ending in dom(w_j) and we say that $\mathcal{M}_j(\varphi, w)$ is the output of j^{th} incremental step.

$$\mathcal{M}_j(\varphi, w) := \{\, \tau(s) \mid s \in [\![\varphi]\!],\ s \in \mathit{Sub}_{[i:j]}(w) \text{ and } 1 \le i \le j \}$$

We then define the state of the online timed pattern matching procedure at the step j as a left-reduced timed regular expression.

Definition 10 (The State of Online Procedure). *Given a pure timed regular expression φ the state of the online timed pattern matching procedure after reading a prefix $w_{1..j}$ of the input signal is:*

$$\psi_j := \bigvee_{1 \le i \le j} \Delta_{w_{i..j}}(\varphi)$$

Then, starting with $\psi_0 = \varphi$, we update the state upon reading w_{j+1} by letting

$$\psi_{j+1} = \Delta_{w_{j+1}}(\psi_j) \vee \Delta_{w_{j+1}}(\varphi)$$

Now we show that the extraction of reduced signals from state ψ_j provides the match set $\mathcal{M}_j(\varphi, w)$. We do not make a distinction here between a reduced signal α and its time domain $\tau(\alpha)$ as they stand for the same thing.

Theorem 3. *Given a state ψ_j of an online matching procedure for expression φ and a signal w, the incremental match set $\mathcal{M}_j(\varphi, w)$ is found by the extraction of the state:*

$$\mathcal{M}_j(\varphi, w) = \text{xt}(\psi_j)$$

Proof. Following Definition 10 and Lemma 1 we know the state ψ_j represents a reduced language $\delta_S(\varphi)$ of φ with respect to a set of signals S satisfying $s \in \text{Sub}(w)$ and $\tau_2(s) \in \text{dom}(w_j)$. A reduced signal α in $\delta_S(\varphi)$ indicates the existence of a signal $s \in S$ such that $\tau(s) = \tau(\alpha)$ and $s \in [\![\varphi]\!]$, thus s is a match. Then we can find the match set $\mathcal{M}_j$ by extracting all reduced signals from the state ψ_j.

Theorem 3 allows us to have a complete procedure for online timed pattern matching for given φ and an input signal $w = w_1 \dots w_n$ summarized below:

1. Extract φ to see if the empty word is a match.
2. For $1 \leq j \leq n$ repeat:
   (a) Update the state of the matching ψ_j by deriving the previous state ψ_{j-1} with respect to w_j and adding a new derivation $\Delta_{w_j}(\varphi)$ to the state for matches starting in segment j.
   (b) Extract ψ_j to get matches ending in segment j.

We present an example of online pattern matching for the timed regular expression $\varphi := \langle p \cdot q \rangle_{[4,7]}$ and input signal $w := w_1 w_2 w_3$ with $w_1 : [0, 3) \mapsto \{p \wedge \neg q\}$, $w_2 : [3, 8) \mapsto \{p \wedge q\}$ and $w_3 : [8, 10) \mapsto \{\neg p \wedge q\}$ over propositional variables p and q shown in Figure 2. In Table 2 we depict the step-by-step computation of the match set $\mathcal{M}(\varphi, w)$ after reading the next segment from w. For Step 1 the state ψ_1 is equal to the derivative of φ with respect to w_1 such that $\psi_1 = \langle \Gamma_1 \cdot q \rangle_{[4,7]} \vee \langle \Gamma_1 \cdot p \cdot q \rangle_{[4,7]}$ where $\Gamma_1 = {}^{[0,3]}_{[0,3]}\langle \checkmark \rangle_{[0,3]}$. The extraction $\text{xt}(\psi_1)$ is empty therefore we do not have any match ending in $\text{dom}(w_1) = [0, 2)$. For Step 2 where $\Gamma_2 = {}^{[3,8]}_{[3,8]}\langle \checkmark \rangle_{[0,5]}$ the extraction of

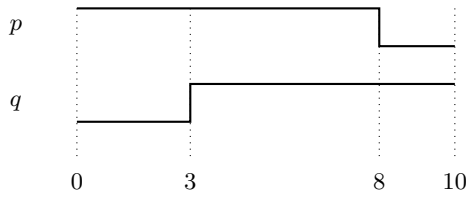

Fig. 2. A signal $w := w_1 w_2 w_3$ over variables p and q.

Symbols		$\{p \wedge \neg q\}$	$\{p \wedge q\}$	$\{\neg p \wedge q\}$
Segments		$[0,3)$	$[3,8)$	$[8,10)$
$[0,3)$	$\langle p \cdot q \rangle_I$	$\xrightarrow[\Delta_{w_1}]{} \langle \Gamma_1 \cdot q \rangle_I \vee \langle \Gamma_1 \cdot p \cdot q \rangle_I$	$\xrightarrow[\Delta_{w_2}]{} \langle \Gamma_1 \cdot \Gamma_2 \rangle_I \vee \langle \Gamma_1 \cdot \Gamma_2 \cdot q \rangle_I \vee \langle \Gamma_1 \cdot \Gamma_2 \cdot p \cdot q \rangle_I$	$\xrightarrow[\Delta_{w_3}]{} \langle \Gamma_1 \cdot \Gamma_2 \cdot \Gamma_3 \rangle_I \vee \langle \Gamma_1 \cdot \Gamma_2 \cdot \Gamma_3 \cdot q \rangle_I$
$[3,8)$		$\langle p \cdot q \rangle_I$	$\xrightarrow[\Delta_{w_2}]{} \langle \Gamma_2 \rangle_I \vee \langle \Gamma_2 \cdot q \rangle_I \vee \langle \Gamma_2 \cdot p \cdot q \rangle_I$	$\xrightarrow[\Delta_{w_3}]{} \langle \Gamma_2 \cdot \Gamma_3 \rangle_I \vee \langle \Gamma_2 \cdot \Gamma_3 \cdot q \rangle_I$
$[8,10)$			$\langle p \cdot q \rangle_I$	$\xrightarrow[\Delta_{w_3}]{} \varnothing$

Table 2. Timed pattern matching using derivatives for $w = w_1 w_2 w_3$ and $\varphi = \langle p \cdot q \rangle_I$. Entries represent the derivative with respect to $w_{i..j}$. Reduced expressions, indicating matched segments, are shaded with green. ($I = [4,7]$, $\Gamma_1 = {}^{[0,3]}_{[0,3]}\langle\checkmark\rangle_{[0,3]}$, $\Gamma_2 = {}^{[3,8]}_{[3,8]}\langle\checkmark\rangle_{[0,5]}$ and $\Gamma_3 = {}^{[8,10]}_{[8,10]}\langle\checkmark\rangle_{[0,2]}$).

the state is equal to $\mathrm{xt}(\psi_2) = \langle \Gamma_1 \cdot \Gamma_2 \rangle_{[4,7]} \vee \langle \Gamma_2 \rangle_{[4,7]} = {}^{[4,8]}_{[0,3]}\langle\checkmark\rangle_{[4,7]} \vee {}^{[7,8]}_{[3,4]}\langle\checkmark\rangle_{[4,5]}$. Similarly, for Step 3 where $\Gamma_3 = {}^{[8,10]}_{[8,10]}\langle\checkmark\rangle_{[0,2]}$, the extraction of the state is equal to $\mathrm{xt}(\psi_3) = \langle \Gamma_1 \cdot \Gamma_2 \cdot \Gamma_3 \rangle_{[4,7]} \vee \langle \Gamma_2 \cdot \Gamma_3 \rangle_{[4,7]} = {}^{[8,9]}_{[1,3]}\langle\checkmark\rangle_{[5,7]} \vee {}^{[8,9]}_{[4,6]}\langle\checkmark\rangle_{[4,5]}$. In Figure 3 we illustrate corresponding segments (t, t') extracted in Steps 2 and 3 where solid regions show the actual outputs for the corresponding step.

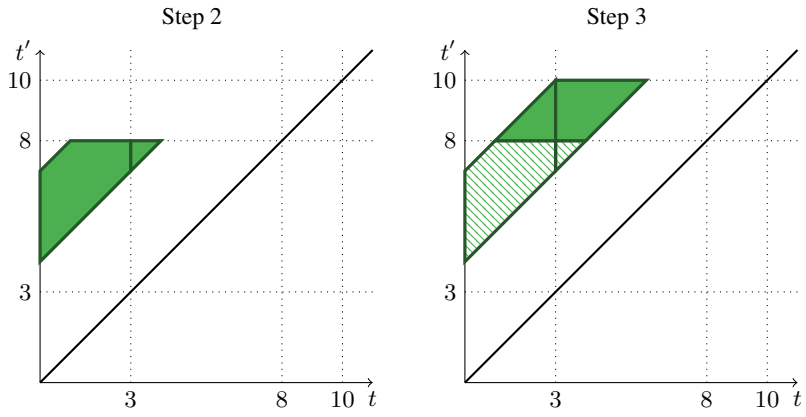

Fig. 3. A graphical representation of online timed pattern matching presented in Table 2 with t and t' denoting, respectively, the beginning and end of the match.

We implemented our procedure using the functional term rewriting language PURE and C++. Besides derivative and extraction rules we introduced in this paper, our im-

plementation includes some basic algebraic rewrite rules as well as simplification rules for reduced expressions given in Section 3. We perform our experiments on a 3.3GHz machine for a set of test patterns and we depict performance results of the online procedure in comparison with the offline procedure in [17] in Table 3. For typical cases, experiments suggest a linear time performance with respect to the number of segments in the input for both algorithms. Although the online procedure runs slower than the offline procedure, it requires less memory and the memory usage does not depend on the input size as expected.

	Offline Algorithm Input Size			Online Algorithm Input Size		
Test Patterns	100K	500K	1M	100K	500K	1M
p	0.06/17	0.27/24	0.51/33	6.74/14	29.16/14	57.87/14
$p \cdot q$	0.08/21	0.42/46	0.74/77	8.74/14	42.55/14	81.67/14
$\langle p \cdot q \cdot \langle p \cdot q \cdot p \rangle_I \cdot q \cdot p \rangle_J$	0.23/28	1.09/77	2.14/140	28.07/14	130.96/14	270.45/14
$(\langle p \cdot q \rangle_I \cdot r) \wedge (p \cdot \langle q \cdot r \rangle_J)$	0.13/23	0.50/51	1.00/86	15.09/15	75.19/15	148.18/15
$p \cdot (q \cdot r)^*$	0.11/20	0.49/37	0.96/60	11.53/15	52.87/15	110.58/15

Table 3. Execution times/Memory usage (in seconds/megabytes)

6 Conclusions

The contribution of the paper is both theoretical and practical. From a theoretical standpoint we have tackled the difficult problem of exporting the concept of derivatives from discrete to timed behaviors, languages and expressions. To this end we introduced a new approach to handle signals in absolute time, yielding a new type of monoid with interesting properties that by itself is worth investigating in the future. We have shown that such derivatives can be computed syntactically using left-reduced timed regular expressions and that all the matches of the expressions in the signal can be extracted from this representation.

We have used these results to implement a novel procedure for online pattern matching for timed behavior that can be used to monitor systems in real time and detect occurrences of complex patterns. Our procedure consumes a constant segment from the input signal and reports a set of matches ending in that segment before processing the next segment. The algorithm can be applied, of course, to the discrete case where words are viewed as signals that can change their values only at integer times. Despite the overhead, our algorithm might be advantageous for words that have long periods of stuttering if a delay in the detection of matching can be tolerated.

We believe that this procedure has a lot of potential applications in detecting temporal patterns at different time scales. It can be used, for example to detect patterns in music as in [8], in cardiac behavior or in speech. To this end the expression should be

extended with predicates over real numbers [7] as in the passage from MTL to STL (signal temporal logic) [11]. Other potential application domains could be the detection of congestions in traffic or in communication network and the analysis of execution logs of organizations information systems or web servers, for example to detect internet robots or customers who are about to abandon our web site.

Acknowledgement: This work was partially supported by the French ANR projects EQINOCS and CADMIDIA and benefitted from useful comments made by anonymous referees.

References

1. Valentin M. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theoretical Computer Science*, 155(2):291–319, 1996.
2. Valentin M. Antimirov and Peter D. Mosses. Rewriting extended regular expressions. *Theoretical Computer Science*, 143(1):51–72, 1995.
3. Eugene Asarin, Paul Caspi, and Oded Maler. A Kleene theorem for timed automata. In *Logic in Computer Science (LICS)*, pages 160–171, 1997.
4. Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *Journal of ACM*, 49(2):172–206, 2002.
5. Gérard Berry and Ravi Sethi. From regular expressions to deterministic automata. *Theoretical Computer Science*, 48(3):117–126, 1986.
6. Janusz A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11(4):481–494, 1964.
7. Thomas Ferrère, Oded Maler, Dejan Nickovic, and Dogan Ulus. Measuring with timed patterns. In *Computer Aided Verification CAV*, pages 322–337, 2015.
8. Jean-Louis Giavitto and José Echeveste. Real-time matching of Antescofo temporal patterns. In *Principles and Practice of Declarative Programming (PPDP)*, pages 93–104, 2014.
9. John Havlicek and Scott Little. Realtime regular expressions for analog and mixed-signal assertions. In *Formal Methods in Computer-Aided Design (FMCAD)*, pages 155–162, 2011.
10. Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
11. Oded Maler, Dejan Nickovic, and Amir Pnueli. Checking temporal properties of discrete, timed and continuous behaviors. In *Pillars of Computer Science*, volume 4800 of *Lecture Notes in Computer Science*, pages 475–505. Springer Berlin Heidelberg, 2008.
12. Katell Morin-Allory and Dominique Borrione. On-line monitoring of properties built on regular expressions. In *Forum on specification and Design Languages, (FDL)*, pages 249–255, 2006.
13. Scott Owens, John H. Reppy, and Aaron Turon. Regular-expression derivatives re-examined. *Journal of Functional Programming*, 19(2):173–190, 2009.
14. Grigore Rosu and Mahesh Viswanathan. Testing extended regular language membership incrementally by rewriting. In *Rewriting Techniques and Applications (RTA)*, pages 499–514, 2003.
15. Koushik Sen and Grigore Rosu. Generating optimal monitors for extended regular expressions. *Electronic Notes Theoretical Computer Science*, 89(2):226–245, 2003.
16. Martin Sulzmann and Pippijn van Steenhoven. A flexible and efficient ML lexer tool based on extended regular expression submatching. In *Compiler Construction (CC)*, pages 174–191, 2014.
17. Dogan Ulus, Thomas Ferrère, Eugene Asarin, and Oded Maler. Timed pattern matching. In *Formal Modeling and Analysis of Timed Systems (FORMATS)*, pages 222–236, 2014.