Online Timed Pattern Matching using Derivatives

Dogan Ulus†, Thomas Ferrère†, Eugene Asarin††, and Oded Maler†

† VERIMAG, Université Grenoble-Alpes & CNRS, France
†† IRIF, Université Paris Diderot, France

April 7, 2016

TACAS 2016, Eindhoven

The Problem of Finding All Sprints

- Assume you're a football manager interested in formal methods.
- You want to **find all sprints** by a player **formally**.
- You have a sprint specification:
  - A period of **high acceleration** followed by a period of **high speed** for **1 second at least.**
- How to solve?

Hint: This is a problem of timed pattern matching.

The Problem of Finding All Sprints

- Assume you're a football manager interested in formal methods.
- You want to **find all sprints** by a player **formally**.
- You have a sprint specification:
  - A period of **high acceleration** followed by a period of **high speed** for **1 second at least.**
- How to solve?

Hint: This is a problem of timed pattern matching.

The Problem of Finding All Sprints

- Given data and pattern, we perform timed pattern matching.

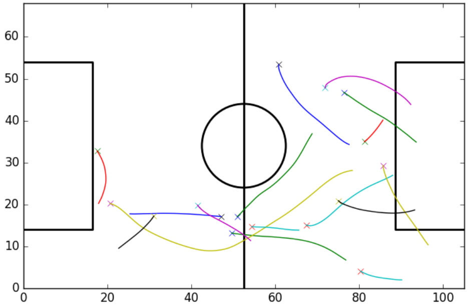


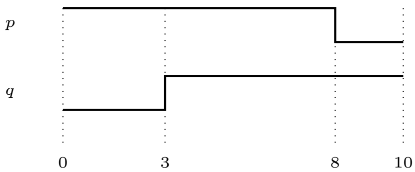
Figure: All sprints by Olcan Adin, Galatasaray v Sivasspor, 16 Jan 2016.

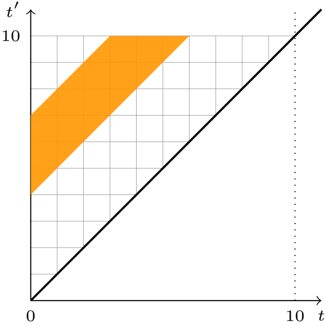
Timed Pattern Matching

- Finding **all** segments of a **dense-time** signal that satisfy a **timed regular expression**.
- For given an expression φ and a signal w:

$$\mathcal{M}(\varphi, w) = \{ \ (t, t') \ \mid \ w[t, t'] \models \varphi \ \}$$

- Example:
  - A pattern $\varphi = \langle p \cdot q \rangle_{[4,7]}$.
  - A signal w over p and q.



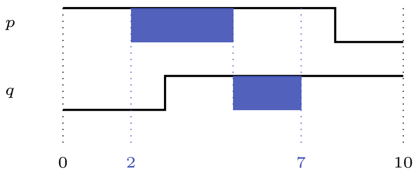


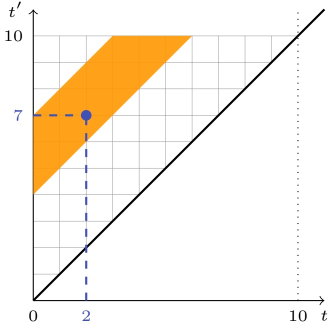
Timed Pattern Matching

- Finding **all** segments of a **dense-time** signal that satisfy a **timed regular expression**.
- For given an expression φ and a signal w:

$$\mathcal{M}(\varphi, w) = \{ \ (t, t') \ | \ w[t, t'] \models \varphi \ \}$$

- Example:
  - A pattern $\varphi = \langle p \cdot q \rangle_{[4,7]}$.
  - A signal w over p and q.





Timed Pattern Matching

- Previously we presented an offline procedure in [UFAM'14].
- We now develop an online procedure to
  - Perform matching on streams.
  - Monitor and alert in real time.
  - Have better memory performance.
- Our online procedure is based on the concept of **derivatives**.

Recipe Analogy

The original concept of derivatives can be explained by an analogy:

- A recipe is a list of actions to be done sequentially.
- To-do list for an eggplant puree:
  - Wash and prick the eggplants with a fork.
  - Bake eggplants for 25 mins.
  - Smash eggplants.
  - Add flour.
  - Add milk and mix it well.
- Each time, you complete an action, you delete the item.
- Do everything correct and you will end up **the empty list**.
  (You are missing the point if you think of the puree.)
- Obviously this is an online acceptance procedure.

Recipe Analogy

The original concept of derivatives can be explained by an analogy:

- A recipe is a list of actions to be done sequentially.
- To-do list for an eggplant puree:
  - Wash and prick the eggplants with a fork.
  - Bake eggplants for 25 mins.
  - Smash eggplants.
  - Add flour.
  - Add milk and mix it well.
- Each time, you complete an action, you delete the item.
- Do everything correct and you will end up **the empty list**.
  (You are missing the point if you think of the puree.)
- Obviously this is an online acceptance procedure.

Recipe Analogy

The original concept of derivatives can be explained by an analogy:

- A recipe is a list of actions to be done sequentially.
- To-do list for an eggplant puree:
  - Wash and prick the eggplants with a fork.
  - Bake eggplants for 25 mins.
  - Smash eggplants.
  - Add flour.
  - Add milk and mix it well.
- Each time, you complete an action, you delete the item.
- Do everything correct and you will end up **the empty list**.

  (You are missing the point if you think of the puree.)
- Obviously this is an online acceptance procedure.

Recipe Analogy

The original concept of derivatives can be explained by an analogy:

- A recipe is a list of actions to be done sequentially.
- To-do list for an eggplant puree:
  - Wash and prick the eggplants with a fork.
  - Bake eggplants for 25 mins.
  - Smash eggplants.
  - Add flour.
  - Add milk and mix it well.
- Each time, you complete an action, you delete the item.
- Do everything correct and you will end up **the empty list**.
  (You are missing the point if you think of the puree.)
- Obviously this is an online acceptance procedure.

Recipe Analogy

The original concept of derivatives can be explained by an analogy:

- A recipe is a list of actions to be done sequentially.
- To-do list for an eggplant puree:
  - Wash and prick the eggplants with a fork.
  - Bake eggplants for 25 mins.
  - Smash eggplants.
  - Add flour.
  - Add milk and mix it well.
- Each time, you complete an action, you delete the item.
- Do everything correct and you will end up **the empty list**.
  (You are missing the point if you think of the puree.)
- Obviously this is an online acceptance procedure.

Recipe Analogy

The original concept of derivatives can be explained by an analogy:

- A recipe is a list of actions to be done sequentially.
- To-do list for an eggplant puree:
  - Wash and prick the eggplants with a fork.
  - Bake eggplants for 25 mins.
  - Smash eggplants.
  - Add flour.
  - Add milk and mix it well.
- Each time, you complete an action, you delete the item.
- Do everything correct and you will end up **the empty list**.
  (You are missing the point if you think of the puree.)
- Obviously this is an online acceptance procedure.

Recipe Analogy

The original concept of derivatives can be explained by an analogy:

- A recipe is a list of actions to be done sequentially.
- To-do list for an eggplant puree:
  - Wash and prick the eggplants with a fork.
  - Bake eggplants for 25 mins.
  - Smash eggplants.
  - Add flour.
  - Add milk and mix it well.
- Each time, you complete an action, you delete the item.
- Do everything correct and you will end up **the empty list**.
  (You are missing the point if you think of the puree.)
- Obviously this is an online acceptance procedure.

Derivatives

Definition

The derivative of a language $\mathcal{L}$ over Σ^* with respect to a word u is defined as

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

Language Membership

$$w \in \mathcal{L} \quad \text{iff} \quad \epsilon \in D_w(\mathcal{L})$$

- Compute **derivatives** and check **empty word containment**.
- Pattern matching is more than that:
  Membership queries for **all subwords**.

Derivatives of Regular Expressions

- RE syntax:

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

- The derivative of a regular expression r with respect to a letter a can be found recursively by these rewrite rules. (Brzozowski 1964)

Empty Word Extraction

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

Derivatives of Regular Expressions

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

Matching using Derivatives

Example: Let $\varphi = a^*(bc)^*$ and $w = abcbc$.

Symbols	a	b	c	b	c
Positions	1	2	3	4	5
1 $\quad \varphi \xrightarrow[D_a]{}$	$a^*(bc)^*$				

$$\epsilon \in D_a(\varphi) \rightarrow a \in \varphi$$

Matching using Derivatives

Example: Let $\varphi = a^*(bc)^*$ and $w = abcbc$.

Symbols	a	b	c	b	c
Positions	1	2	3	4	5

1 $\quad \varphi \xrightarrow[D_a]{} \boxed{a^*(bc)^*} \xrightarrow[D_b]{} c(bc)^*$

$$\epsilon \notin D_{ab}(\varphi) \rightarrow ab \notin \varphi$$

Matching using Derivatives

Example: Let $\varphi = a^*(bc)^*$ and $w = abcbc$.

Symbols	a	b	c	b	c
Positions	1	2	3	4	5

1 $\quad \varphi \xrightarrow[D_a]{} \boxed{a^*(bc)^*} \xrightarrow[D_b]{} c(bc)^* \xrightarrow[D_c]{} \boxed{(bc)^*}$

$$\epsilon \in D_{abc}(\varphi) \rightarrow abc \in \varphi$$

Matching using Derivatives

Example: Let $\varphi = a^*(bc)^*$ and $w = abcbc$.

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

$$\epsilon \notin D_{abcb}(\varphi) \rightarrow abcb \notin \varphi$$

Matching using Derivatives

Example: Let $\varphi = a^*(bc)^*$ and $w = abcbc$.

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

$$\epsilon \in D_{abcbc}(\varphi) \to abcbc \in \varphi$$

Matching using Derivatives

Example: Let $\varphi = a^*(bc)^*$ and $w = abcbc$.

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

Below all segments of w that satify the expression φ:

$$\mathcal{M}(\varphi, w) = \{(1,1), (1,3), (1,5), (2,3), (2,5), (4,5)\}$$

Transition to Timed

- Each action takes some time.
- Duration of actions can be constrained to be between m an n time units such that

$$\langle \mathsf{Actions} \rangle_{[m,n]}$$

- The expression specifying all correct timings for our recipe:

$$\Big\langle \mathsf{Wash} \cdot \langle \mathsf{Bake} \rangle_{[23,27]} \cdot \mathsf{Smash} \cdot \mathsf{Flour} \cdot \mathsf{Milk} \Big\rangle_{[0,60]}$$

- For online matching such specifications, you have to:
  - Remember how much time passed for actions.
  - Do it for uncountable number of start points.

Recipe Analogy with Time

- Adding absolute timestamps to our recipe.
- Very punctual recipe:

6h00 - 6h05	Wash and prick the eggplants with a fork.
6h05 - 6h30	Bake eggplants.
6h30 - 6h45	Smash eggplants.
6h45 - 6h48	Add flour.
6h48 - 6h50	Add milk and mix it well.

- Now, when we complete an action, we put a checkmark.
- You will end up **a fully checkmarked list** with stamps.
- This is more informative: Now we can directly say the puree is cooked between 6h00 and 6h50.
- WHY: It simplifies a lot when we deal with uncountable sets.

Recipe Analogy with Time

- Adding absolute timestamps to our recipe.
- Very punctual recipe:

6h00 - 6h05	✓
6h05 - 6h30	Bake eggplants.
6h30 - 6h45	Smash eggplants.
6h45 - 6h48	Add flour.
6h48 - 6h50	Add milk and mix it well.

- Now, when we complete an action, we put a checkmark.
- You will end up **a fully checkmarked list** with stamps.
- This is more informative: Now we can directly say the puree is cooked between 6h00 and 6h50.
- WHY: It simplifies a lot when we deal with uncountable sets.

Recipe Analogy with Time

- Adding absolute timestamps to our recipe.
- Very punctual recipe:

6h00 - 6h05	✓
6h05 - 6h30	✓
6h30 - 6h45	Smash eggplants.
6h45 - 6h48	Add flour.
6h48 - 6h50	Add milk and mix it well.

- Now, when we complete an action, we put a checkmark.
- You will end up **a fully checkmarked list** with stamps.
- This is more informative: Now we can directly say the puree is cooked between 6h00 and 6h50.
- WHY: It simplifies a lot when we deal with uncountable sets.

Recipe Analogy with Time

- Adding absolute timestamps to our recipe.
- Very punctual recipe:

6h00 - 6h05 ✓
6h05 - 6h30 ✓
6h30 - 6h45 ✓
6h45 - 6h48 Add flour.
6h48 - 6h50 Add milk and mix it well.

- Now, when we complete an action, we put a checkmark.
- You will end up **a fully checkmarked list** with stamps.
- This is more informative: Now we can directly say the puree is cooked between 6h00 and 6h50.
- WHY: It simplifies a lot when we deal with uncountable sets.

Recipe Analogy with Time

- Adding absolute timestamps to our recipe.
- Very punctual recipe:

6h00 - 6h05	✓
6h05 - 6h30	✓
6h30 - 6h45	✓
6h45 - 6h48	✓
6h48 - 6h50	Add milk and mix it well.

- Now, when we complete an action, we put a checkmark.
- You will end up **a fully checkmarked list** with stamps.
- This is more informative: Now we can directly say the puree is cooked between 6h00 and 6h50.
- WHY: It simplifies a lot when we deal with uncountable sets.

Recipe Analogy with Time

- Adding absolute timestamps to our recipe.
- Very punctual recipe:

6h00 - 6h05 ✓
6h05 - 6h30 ✓
6h30 - 6h45 ✓
6h45 - 6h48 ✓
6h48 - 6h50 ✓

- Now, when we complete an action, we put a checkmark.
- You will end up **a fully checkmarked list** with stamps.
- This is more informative: Now we can directly say the puree is cooked between 6h00 and 6h50.
- WHY: It simplifies a lot when we deal with uncountable sets.

Recipe Analogy with Time

- Adding absolute timestamps to our recipe.
- Very punctual recipe:

6h00 - 6h05	✓
6h05 - 6h30	✓
6h30 - 6h45	✓
6h45 - 6h48	✓
6h48 - 6h50	✓

- Now, when we complete an action, we put a checkmark.
- You will end up **a fully checkmarked list** with stamps.
- This is more informative: Now we can directly say the puree is cooked between 6h00 and 6h50.
- WHY: It simplifies a lot when we deal with uncountable sets.

Recipe Analogy with Time

- Adding absolute timestamps to our recipe.
- Very punctual recipe:

6h00 - 6h05 ✓
6h05 - 6h30 ✓
6h30 - 6h45 ✓
6h45 - 6h48 ✓
6h48 - 6h50 ✓

- Now, when we complete an action, we put a checkmark.
- You will end up **a fully checkmarked list** with stamps.
- This is more informative: Now we can directly say the puree is cooked between 6h00 and 6h50.
- WHY: It simplifies a lot when we deal with uncountable sets.

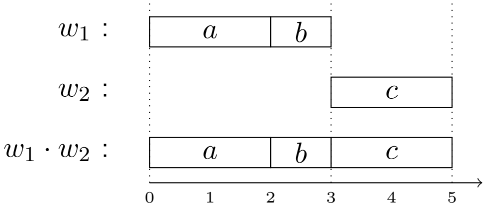
Signals in absolute time

- A signal is a piecewise-constant function over Σ:

$$w : [t, t') \to \Sigma$$

starting at t and ending at t' with a duration $t' - t$.

- The concatenation $w_1 \cdot w_2$ is defined only if w_1 meets w_2:



- All signals: $\Sigma^{(*)}$, Non-empty: $\Sigma^{(+)}$, Constant: $\Sigma^{(1)}$.

Extended Signals

- The special symbol $\checkmark$ extends the alphabet Σ.
- $\Sigma_{\checkmark} = \Sigma \cup \{\checkmark\}$
- An *extended* signal w if $w \in \Sigma_{\checkmark}^{(*)}$.
- Some classes of extended signals:

$$\begin{aligned} \text{A } \textbf{pure} \text{ signal } w & \quad \text{if } w \in \Sigma^{(*)} \\ \text{A } \textbf{reduced} \text{ signal } w & \quad \text{if } w \in \checkmark^{(*)} \\ \text{A } \textbf{left-reduced} \text{ signal } w & \quad \text{if } w \in \checkmark^{(*)} \cdot \Sigma^{(*)} \end{aligned}$$

- Pure - Original recipe
- Reduced - Fully checkmarked recipe
- Left-reduced - Partially checkmarked recipe

Extended Timed Regular Expressions

- Let P a set of propositions.
- Representing extended signal languages over $\Sigma = \mathbb{B}^{|P|}$.
- The syntax:

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

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

- The semantics:

$$\begin{array}{rcl} & \dots & \\ [\![p]\!] & = & \{w : [t,t') \to \Sigma \mid t < t' \text{ and } \forall t'' \in [t,t').\ w(t'') \models p\} \\ [\![\checkmark]\!] & = & \{w : [t,t') \to \checkmark \mid t < t'\} \\ & \dots & \\ [\![{}^{K}_{J}\langle \varphi \rangle_I]\!] & = & \{w \mid w \in [\![\varphi]\!],\ |w| \in I,\ w \neq \epsilon \to (\tau_1(w) \in J \wedge \tau_2(w) \in K)\} \end{array}$$

Left Reduction

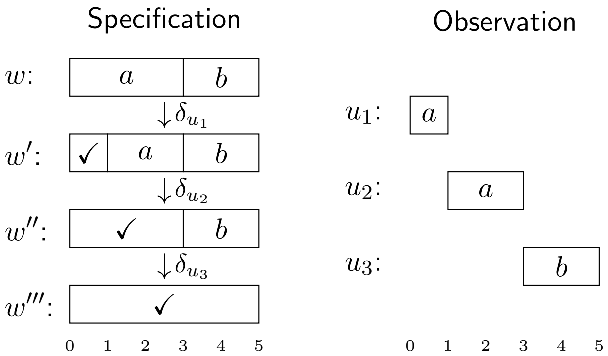


Figure: A left reduction example.

Left Reduction

- We introduce left reduction as a position (and duration) preserving derivative operation.

Definition (Left Reduction)

The left reduction of a language $\mathcal{L}$ with respect to u is:

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

where $\gamma \in \checkmark^{(*)}$ and $\mathrm{dom}(u) = \mathrm{dom}(\gamma)$.

Language Membership

$$u \in \mathcal{L} \quad \text{ iff } \quad \gamma \in \delta_u(\mathcal{L})$$

Derivatives of Timed Regular Expressions

- A symbolic computation of left reduction with respect to all factors of a constant signal v:

Theorem (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 $[\![\psi]\!] = \Delta_v([\![\varphi]\!])$.

$$
\begin{array}{rcl}
\Delta_v(\varnothing) & = & \varnothing \\
\Delta_v(\epsilon) & = & \varnothing \\
\Delta_v(\checkmark) & = & \varnothing \\
\Delta_v(p) & = & \begin{cases} \Gamma \vee \Gamma \cdot p & \textit{if } a \models p \textit{ where } \Gamma := {}^{[t,t']}_{[t,t']}\langle \checkmark \rangle_{[0,t'-t]} \\ \varnothing & \textit{otherwise} \end{cases} \\
\Delta_v(\psi_1 \cdot \psi_2) & = & \Delta_v(\psi_1) \cdot \psi_2 \;\vee\; \mathit{xt}(\psi_1 \vee \Delta_v(\psi_1)) \cdot \Delta_v(\psi_2) \\
\Delta_v(\psi_1 \vee \psi_2) & = & \Delta_v(\psi_1) \vee \Delta_v(\psi_2) \\
\Delta_v(\psi_1 \wedge \psi_2) & = & \Delta_v(\psi_1) \wedge \Delta_v(\psi_2) \\
\Delta_v({}^K_J\langle \psi \rangle_I) & = & {}^K_J\langle \Delta_v(\psi) \rangle_I \\
\Delta_v(\psi^*) & = & \mathit{xt}(\Delta_v(\psi))^* \cdot \Delta_v(\psi) \cdot \psi^*
\end{array}
$$

Online Timed Pattern Matching

Inputs/Outputs:

- The input φ is a timed regular expression.
- The input $w = w_1 w_2 \ldots w_n$ to be read incrementally.
- The procedure yields the set of matches ending in j^{th} segment at each step.

Full Procedure:

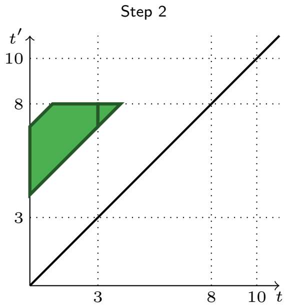
- Extract φ to see if the empty word is a match.
- For $1 \leq j \leq n$ repeat:
  - Update the state by deriving the previous state with respect to w_j and adding a new derivation $\Delta_{w_j}(\varphi)$ to the state for matches starting in j^{th} segment.
  - Extract the state to get matches ending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

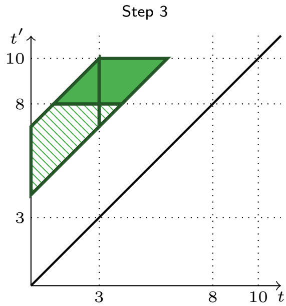
Example

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

- $\Gamma_1 = {}^{[0,3]}_{[0,3]}\langle \checkmark \rangle_{[0,3]}$
- $\Gamma_2 = {}^{[3,8]}_{[3,8]}\langle \checkmark \rangle_{[0,5]}$
- $\Gamma_3 = {}^{[8,10]}_{[8,10]}\langle \checkmark \rangle_{[0,2]}$

Example





- At each step, we report segments satisying the expression.

Performance - Online vs Offline

Test Patterns	Offline Algorithm Input Size			Online Algorithm Input Size		
	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

- Execution times/Memory usage (in seconds/megabytes).
- Both are linear for typical inputs.
- Online is 100 times slower but memory usage is constant.
- These numbers are sufficient for many applications.

Discussion

- We presented both theoretical and practical results:
- We formulated an algebraic approach in the timed theory.
  - The time passed represented by ($\checkmark$) symbol.
  - Timed derivatives
- We developed an online timed pattern matching procedure.
- Our procedure consumes a constant segment from the input signal and reports a set of matches ending in that segment.
- Do not worry, we have a tool:
  - `github.com/doganulus/montre` (Soon)
- Applications: Runtime verification, robotics, medical monitoring, ...

Thank you!!