Runtime Enforcement of Regular Timed Properties

Srinivas Pinisetty, Yliès Falcone\textsuperscript{2}, Thierry Jéron\textsuperscript{1}, Hervé Marchand\textsuperscript{1}

INRIA Rennes - Bretagne Atlantique, France
Université Grenoble I, Laboratoire d’Informatique de Grenoble, France

SAC-SVT 2014 (Gyeongju, Korea),
extended to Science of Computer Programming
Runtime verification and enforcement (monitors)

Runtime verification and enforcement:

- Monitor observing a stream $\sigma$ of system events (e.g., trace, log, messages).
- Requested property $\varphi$.
- No system model.

**Runtime verification**

- Input: stream of events.
- Modified to satisfy the property.
- Output: stream of events.

---

Does $\sigma$ satisfy $\varphi$? 
Output: stream of **verdicts**.
Runtime verification and enforcement (monitors)

Runtime verification and enforcement:

- Monitor observing a stream $\sigma$ of system events (e.g., trace, log, messages).
- Requested property $\varphi$.
- No system model.

**Runtime verification**

- Verdicts $w \in \mathbb{D}^\infty$.
- Event $\sigma \in \Sigma^\infty$.
- Does $\sigma$ satisfy $\varphi$?
- Output: stream of *verdicts*.

**Runtime enforcement**

- Input: stream of events.
- Modified to satisfy the property.
- Output: stream of *events*.
- Event $o \subseteq \sigma$.
- Event $o \models \varphi!$.
Specifying Timed Properties

Runtime Enforcement of Regular Timed Properties

Conclusions and FW

Motivations for \textit{timed} enforcement

Specifying the timing behavior

Allowing to specify desired behaviors of a system with constraints on both the \textbf{order of events} and \textbf{timing}.

- After action “a”, action “b” should occur \textit{with a delay of at least 5 time units between them}.

- The system should allow consecutive requests \textit{with a delay of at least 10 time units between any two requests}. 
Motivations for *timed* enforcement

Specify the timing behavior

Allowing to specify desired behaviors of a system with constraints on both the **order of events** and **timing**.

- After action “a”, action “b” should occur with a delay of at least 5 time units between them.
- The system should allow consecutive requests with a delay of at least 10 time units between any two requests.

Application domains

- Real-time embedded systems, monitoring hardware failures, communication protocols, web services, etc.
- Examples of monitor usage:
  - firewall to prevent DOS attacks ensuring minimal delay between input events;
  - checking pre-conditions of a service in web applications.
Related work on monitoring

Runtime Enforcement of Untimed properties

Related work on monitoring

Runtime Enforcement of **Untimed** properties


Runtime **Verification** of Timed properties

Efforts mainly to verify timed properties at runtime:

- Safe runtime verification of real-time properties – C. Colombo et al.
**Problem and Contributions**

\[ \varphi: \text{timed property} \]

- **timed events**
- Enforcement Monitor
- **timed events**

\[ o \leq \sigma \]

\[ o \models \varphi! \]

\[ \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^* \]

Factory, Falcone, Jéron, Marchand (INRIA, UJF)
Problem and Contributions

A formal framework for runtime enforcement of timed properties

- Enforcement mechanisms defined at **several abstraction levels** to ease the design and implementation.
- Work as **delayers**: either **increase input dates** or **suppress events** in order to satisfy the property.
- General definitions for all regular properties.
Outline

1. Specifying Timed Properties

2. Runtime Enforcement of Regular Timed Properties
   - Requirements on an Enforcement Mechanism
   - Functional Definition of an Enforcement Mechanism
   - Operational Description of an Enforcement Mechanism
   - Algorithmic Description of an Enforcement Mechanism

3. Conclusions and Future Work
Specifying Timed Properties

Runtime Enforcement of Regular Timed Properties
- Requirements on an Enforcement Mechanism
- Functional Definition of an Enforcement Mechanism
- Operational Description of an Enforcement Mechanism
- Algorithmic Description of an Enforcement Mechanism

Conclusions and Future Work
Specify timed properties

- Input/output streams of timed events modelled as timed words in $\text{tw}(\Sigma) = (\mathbb{R}_{\geq 0} \times \Sigma)^*$ for an alphabet of actions $\Sigma$ and increasing dates $\sigma = (t_1, a_1) \cdot (t_2, a_2) \cdots (t_n, a_n)$.

- Enforced timed property: any regular timed language $\varphi \subseteq \text{tw}(\Sigma)$, i.e., accepted by a Timed Automaton $A_\varphi$. 
Specifying timed properties

- Input/output streams of **timed events** modelled as **timed words** in \( \text{tw}(\Sigma) = (\mathbb{R}_{\geq 0} \times \Sigma)^* \) for an alphabet of **actions** \( \Sigma \) and increasing **dates** \( \sigma = (t_1, a_1) \cdot (t_2, a_2) \cdot \cdots (t_n, a_n) \).

- Enforced timed property: any **regular timed language** \( \varphi \subseteq \text{tw}(\Sigma) \), i.e., accepted by a **Timed Automaton** \( A_\varphi \).

Safety, co-safety and response properties specified by TAs
Specifying timed properties

- Input/output streams of timed events modelled as timed words in $\text{tw}(\Sigma) = (\mathbb{R}_{\geq 0} \times \Sigma)^*$ for an alphabet of actions $\Sigma$ and increasing dates $\sigma = (t_1, a_1) \cdot (t_2, a_2) \cdots (t_n, a_n)$.

- Enforced timed property: any regular timed language $\varphi \subseteq \text{tw}(\Sigma)$, i.e., accepted by a Timed Automaton $\mathcal{A}_\varphi$.

Safety, co-safety and response properties specified by TAs

**Safety**: nothing bad should ever happen (prefix closed).

\[ \Sigma = \{\text{req}\} \]

"A delay of 5 t.u. between any two requests."
Specifying timed properties

- Input/output streams of timed events modelled as timed words in $\text{tw}(\Sigma) = (\mathbb{R}_{\geq 0} \times \Sigma)^*$ for an alphabet of actions $\Sigma$ and increasing dates $\sigma = (t_1, a_1) \cdot (t_2, a_2) \cdots (t_n, a_n)$.
- Enforced timed property: any regular timed language $\varphi \subseteq \text{tw}(\Sigma)$, i.e., accepted by a Timed Automaton $A_\varphi$.

Safety, co-safety and response properties specified by TAs

Co-safety: something good will eventually happen within a finite amount of time (extension closed).

$\Sigma = \{\text{req, gr}\}$

“A request, and then a grant should arrive between 10 and 15 t.u.”
Specifying timed properties

- Input/output streams of timed events modelled as timed words in 
  \[ \text{tw}(\Sigma) = (\mathbb{R}_{\geq 0} \times \Sigma)^* \] 
  for an alphabet of actions \( \Sigma \) and increasing dates 
  \( \sigma = (t_1, a_1) \cdot (t_2, a_2) \cdot \ldots \cdot (t_n, a_n) \).

- Enforced timed property: any regular timed language \( \varphi \subseteq \text{tw}(\Sigma) \), 
  i.e., accepted by a Timed Automaton \( A_{\varphi} \).

Safety, co-safety and response properties specified by TAs

Response: any property.

\[ \Sigma = \{ \text{req, gr} \} \]

“Requests and grants should alternate in this order with a delay between 15 and 20 t.u between the request and the grant.”
Example: response property

\[ \Sigma = \{\text{req}, \text{gr}\} \]

- \( (3, \text{req}) \cdot (18, \text{gr}) \cdot (23, \text{req}) \cdot (42, \text{gr}) \]

Response properties are neither prefix nor extension closed.
Semantics of TA

The semantics of a timed automaton $\mathcal{A} = (L, l_0, X, \Sigma, \Delta, F)$, is a timed transition system $\llbracket \mathcal{A} \rrbracket = (Q, q_0, \Gamma, \rightarrow, Q_F)$ with states of the form $q = (l, \nu)$ (location, valuation of clocks), transitions $(l, \nu) \xrightarrow{(\delta,a)} (l', \nu + \delta[Y \leftarrow 0])$ if $\nu + \delta \models g$ for some $l \xrightarrow{(g,a,Y)} l'$ in $\mathcal{A}$.

Partition of states

The set of states $Q$ of $\llbracket \mathcal{A} \rrbracket$ can be partitioned into $Q = G^c \cup G \cup B^c \cup B$ where $Q_F = G^c \cup G$ and $Q \setminus Q_F = B^c \cup B$ and

- $G^c = Q_F \cap \text{pre}^*(Q \setminus Q_F)$, currently good states,
- $G = Q_F \setminus G^c = Q_F \setminus \text{pre}^*(Q \setminus Q_F)$, good states,
- $B^c = (Q \setminus Q_F) \cap \text{pre}^*(Q_F)$, currently bad states,
- $B = (Q \setminus Q_F) \setminus \text{pre}^*(Q_F)$, bad states.
Major Challenges for enforcement

- for safety properties: on-line correction for each event.
Major Challenges for enforcement

- for safety properties: on-line correction for each event.
- for co-safety properties: wait for an event allowing correction.
Major Challenges for enforcement

- for safety properties: on-line correction for each event.
- for co-safety properties: wait for an event allowing correction.
- response properties: mix of both
1. Specifying Timed Properties

2. Runtime Enforcement of Regular Timed Properties
   - Requirements on an Enforcement Mechanism
   - Functional Definition of an Enforcement Mechanism
   - Operational Description of an Enforcement Mechanism
   - Algorithmic Description of an Enforcement Mechanism

3. Conclusions and Future Work
Summary of the approach

What can an enforcement mechanism do?

- **CANNOT** change the order of events.
- **CANNOT** insert events.
- **CAN** increase the dates of events so satisfy \( \varphi \).
- **CAN** suppress events if no future can satisfy \( \varphi \).

\( \Rightarrow \) the enforcement monitor is a “delayer with suppression”. 
Summary of the approach

- **Requirements** for any enforcement mechanism for $\varphi$
- **Functional definition:**
  - description of the global input/output behavior,
  - satisfying requirements;
- **Enforcement monitor:**
  - timed operational behavior as a rule-based transition system,
  - refining the functional definition;
- **Implementation:** translation of the EM semantic rules into algorithms.
Requirements for any enforcement mechanism for $\varphi$

**Functional definition:**
- description of the global input/output behavior,
- satisfying requirements;

**Enforcement monitor:**
- timed operational behavior as a rule-based transition system,
- refining the functional definition;

**Implementation:** translation of the EM semantic rules into algorithms.
Summary of the approach

- **Requirements** for any enforcement mechanism for $\varphi$
- **Functional definition:**
  - description of the global input/output behavior,
  - satisfying requirements;
- **Enforcement monitor:**
  - timed operational behavior as a rule-based transition system,
  - refining the functional definition;
- **Implementation:** translation of the EM semantic rules into algorithms.
Summary of the approach

- **Requirements** for any enforcement mechanism for $\varphi$
- **Functional definition:**
  - description of the global input/output behavior,
  - satisfying requirements;
- **Enforcement monitor:**
  - timed operational behavior as a rule-based transition system,
  - refining the functional definition;
- **Implementation**: translation of the EM semantic rules into algorithms.
Preliminaries: orders on timed words

**Subsequence:** \((3, a) \cdot (8, c) \Rightarrow (3, a) \cdot (5, b) \cdot (8, c)\)
\[w' \Rightarrow w\text{ if } w'\text{ obtained from } w\text{ by suppressions.}\]

**Delaying order** \(\succ_d\): \((4, a) \cdot (7, b) \cdot (9, c) \succ_d (3, a) \cdot (5, b) \cdot (8, c)\).
\[\sigma' \succ_d \sigma\text{ if they have same untimed projections but dates in } \sigma'\text{ exceed corresponding dates in } \sigma.\]

**Delaying subsequence order** \(\triangleleft_d\): \((4, a) \cdot (9, c) \triangleleft_d (3, a) \cdot (5, b) \cdot (8, c)\)
\[\sigma' \triangleleft_d \sigma \overset{\text{def}}{=} \exists \sigma'' \in \text{tw}(\Sigma) : \sigma'' \triangleleft \sigma \land \sigma' \succ_d \sigma''\]
i.e., \(\sigma'\) obtained from \(\sigma\) by first suppressing some actions, and then increasing the dates of the actions to be kept.

**Lexical order** \(\preceq_{\text{lex}}\): \((3, a) \cdot (5, b) \cdot (8, c) \preceq_{\text{lex}} (3, a) \cdot (6, b) \cdot (7, c)\).
\[\varepsilon \preceq_{\text{lex}} \varepsilon, \text{ and for two events with identical actions } (t, a) \text{ and } (t', a),\]
\[ (t, a) \cdot \sigma \preceq_{\text{lex}} (t', a) \cdot \sigma' \text{ if } t \leq t' \lor (t = t' \land \sigma \preceq_{\text{lex}} \sigma').\]
An **enforcement function** for \( \varphi \) is a function \( E_\varphi : \text{tw}(\Sigma) \rightarrow \text{tw}(\Sigma) \) which transforms timed words, and satisfies the following requirements:

- **Physical constraint:**
  \[
  \forall x \in \text{tw}(\Sigma): E_\varphi(x) \sqsubseteq 0 \rightarrow E_\varphi(x) \sqsubseteq 0.
  \]
  (where \( \sqsubseteq \) is the prefix ordering)
  i.e., the output cannot be undone.

- **Soundness:**
  \[
  \forall x \in \text{tw}(\Sigma): E_\varphi(x) \sqsubseteq E_\varphi'(x).
  \]
  i.e., the output is either empty or satisfies \( E_\varphi \).

- **Transparency:**
  \[
  \forall x \in \text{tw}(\Sigma): E_\varphi(x) \sqsubseteq d E_\varphi'(x).
  \]
  i.e., the output ia a delaying subsequence of the input.
An enforcement function for $\varphi$ is a function $E_\varphi : \text{tw}(\Sigma) \rightarrow \text{tw}(\Sigma)$ which transforms timed words, and satisfies the following requirements:

**Physical constraint:** $\forall \sigma, \sigma' \in \text{tw}(\Sigma) : \sigma \preceq \sigma' \implies E_\varphi(\sigma) \preceq E_\varphi(\sigma')$.

(Where $\preceq$ is the prefix ordering)

i.e., the output cannot be undone.
An enforcement function for $\varphi$ is a function $E_\varphi : \text{tw}(\Sigma) \rightarrow \text{tw}(\Sigma)$ which transforms timed words, and satisfies the following requirements:

**Physical constraint:** $\forall \sigma, \sigma' \in \text{tw}(\Sigma) : \sigma \preceq \sigma' \implies E_\varphi(\sigma) \preceq E_\varphi(\sigma').$

(where $\preceq$ is the prefix ordering)
i.e., the output cannot be undone.

**Soundness:** $\forall \sigma \in \text{tw}(\Sigma) : E_\varphi(\sigma) \models \varphi \lor E_\varphi(\sigma) = \epsilon.$
i.e., the output is either empty or satisfies $\varphi.$
An **enforcement function** for $\varphi$ is a function $E_\varphi : \text{tw}(\Sigma) \rightarrow \text{tw}(\Sigma)$ which transforms timed words, and satisfies the following requirements:

**Physical constraint:** $\forall \sigma, \sigma' \in \text{tw}(\Sigma) : \sigma \preceq \sigma' \implies E_\varphi(\sigma) \preceq E_\varphi(\sigma')$. (where $\preceq$ is the prefix ordering)

i.e., the output cannot be undone.

**Soundness:** $\forall \sigma \in \text{tw}(\Sigma) : E_\varphi(\sigma) \models \varphi \lor E_\varphi(\sigma) = \epsilon$.

i.e., the output is either empty or satisfies $\varphi$.

**Transparency:** $\forall \sigma \in \text{tw}(\Sigma) : E_\varphi(\sigma) \triangleleft_d \sigma$.

i.e., the output is a delaying subsequence of the input.
Transparency

![Graph showing time vs. actions]

- **Time** axis labeled from 1 to 5.
- **Actions** labeled as $a_1$, $a_2$, $a_3$, $a_4$.
- Input events marked with red circles.
- Output events marked with blue stars.

```plaintext
output

input

Pinisetty, Falcone, Jéron, Marchand (INRIA, UJF)
```
Specifying Timed Properties

Runtime Enforcement of Regular Timed Properties

Conclusions and FW

Transparency (2)

Safety, co-safety and response
Functional definition: general scheme

\[ E_\varphi : \text{tw}(\Sigma) \rightarrow \text{tw}(\Sigma). \]

\[ E(\sigma) \models \varphi \quad \text{Enforcement function} \]
**Functional definition: general scheme**

\[ E_\varphi : \text{tw}(\Sigma) \rightarrow \text{tw}(\Sigma). \]
Functional definition: general scheme

\[ E_\varphi : tw(\Sigma) \rightarrow tw(\Sigma). \]

Input and output functions are realized by the observation function:

\[ \text{obs}(\sigma, t) = \max \{ \sigma' \mid \sigma' \preceq \sigma \land \text{time}(\sigma') \leq t \}. \]
Functional definition: principle

\[ E_\varphi : tw(\Sigma) \rightarrow tw(\Sigma) \text{ is defined as } E_\varphi(\sigma) = \Pi_1 (\text{store}_\varphi(\sigma)) \]

where \( \text{store}_\varphi(\sigma) = (\sigma_s, \sigma_c) \) describes how the input stream is transformed:

- \( \sigma_s \) is the computed output (to be released);
- \( \sigma_c \) is a sub-seq. of the suffix of the input stream for which output dates cannot be computed (e.g., co-safety, response).

\text{store}_\varphi(\sigma) \text{ is inductively defined:}

suppose \( \text{store}_\varphi(\sigma) = (\sigma_s, \sigma_c) \) and a new input \((t, a)\) is received, 3 cases:

- if \( \sigma_s \cdot \sigma_c \cdot (t, a) \) allows to satisfy \( \varphi \) by delaying \( \sigma_c \cdot (t, a) \) (\( Q_F = G^C \cup G \) reached by some minimal delaying \( K \)): \( (\sigma_s \cdot K, \epsilon) \)
- if no continuation of \( \sigma_s \cdot \sigma_c \cdot (t, a) \) allows to satisfy \( \varphi \) (\text{Bad} reached by all delayings), \((t, a)\) suppressed: \( (\sigma_s, \sigma_c) \)
- otherwise some continuation may still allow to satisfy \( \varphi \) (\( Q_F \) is reachable): \( (\sigma_s, \sigma_c \cdot (t, a)) \).
Functional definition: formal definition

\[ E_\varphi : \text{tw}(\Sigma) \to \text{tw}(\Sigma) \] is defined as \[ E_\varphi(\sigma) = \Pi_1(\text{store}_\varphi(\sigma)) \]

where \( \text{store}_\varphi : \text{tw}(\Sigma) \to \text{tw}(\Sigma) \times \text{tw}(\Sigma) \) is inductively defined as

\[
\text{store}_\varphi(\epsilon) = (\epsilon, \epsilon) \text{ and }
\]

\[
\text{store}_\varphi(\sigma \cdot (t, a)) = \begin{cases} 
(\sigma_s \cdot \min_{\geq \text{lex}, \text{end}} \kappa_\varphi(\sigma_s, \sigma'_c), \epsilon) & \text{ if } \kappa_\varphi(\sigma_s, \sigma'_c) \neq \emptyset, \\
(\sigma_s, \sigma_c) & \text{ if } \kappa_{\text{pref}}(\varphi)(\sigma_s, \sigma'_c) = \emptyset, \\
(\sigma_s, \sigma'_c) & \text{ otherwise,}
\end{cases}
\]

with

\[
\kappa_\varphi(\sigma_s, \sigma'_c) = \text{CanD}(\sigma'_c) \cap \sigma_s^{-1} \cdot \varphi \\
\text{or } \sigma_s^{-1} \cdot \text{pref}(\varphi) \text{ for } \kappa_{\text{pref}}(\varphi)
\]

and

\[
\text{CanD}(\sigma'_c) = \{ w \in \text{tw}(\Sigma) \mid w \geq_d \sigma'_c \land \text{start}(w) \geq \text{end}(\sigma'_c) \}
\]
i.e., sequences delaying \( \sigma'_c \) and starting after \( t \)

\[
\sigma_s^{-1} \cdot \varphi = \{ w \in \text{tw}(\Sigma) \mid \sigma_s \cdot w \models \varphi \}
\]
i.e., what remains to be satisfied from \( \varphi \) after reading \( \sigma_s \)
Functional definition: Example and Evolution over time

Let \( \text{obs}(\sigma, t) = \max_{\subseteq} \{ \sigma' \mid \sigma' \preceq \sigma \land \text{time}(\sigma') \leq t \} \).

\[ \Sigma = \{ op_1, op_2, op \}. \]
\[ \sigma = (2, op_1) \cdot (3, op_1) \cdot (3.5, op) \cdot (6, op_2). \]
Let \( \text{obs}(\sigma, t) = \max_\prec \{ \sigma' \mid \sigma' \preceq \sigma \land \text{time}(\sigma') \leq t \} \).

\[
\begin{align*}
\Sigma &= \{ \text{op}_1, \text{op}_2, \text{op} \}. \\
\sigma &= (2, \text{op}_1) \cdot (3, \text{op}_1) \cdot (3.5, \text{op}) \cdot (6, \text{op}_2).
\end{align*}
\]

<table>
<thead>
<tr>
<th>( t \in [0, 2[ )</th>
<th>( \text{obs}(\sigma_3, t) = \epsilon )</th>
<th>( \text{store}_{\varphi_3}(\text{obs}(\sigma_3, t)) = (\epsilon, \epsilon) )</th>
<th>( \text{obs}(E_{\varphi_3}(\text{obs}(\sigma_3, t))) = \text{obs}(\epsilon, t) = \epsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t \in [2, 3[ )</td>
<td>( \text{obs}(\sigma_3, t) = (2, \text{op}_1) )</td>
<td>( \text{store}_{\varphi_3}(\text{obs}(\sigma_3, t)) = (\epsilon, (2, \text{op}_1)) )</td>
<td>( \text{obs}(E_{\varphi_3}(\text{obs}(\sigma_3, t))) = \text{obs}(\epsilon, t) = \epsilon )</td>
</tr>
<tr>
<td>( t \in [3, 3.5[ )</td>
<td>( \text{obs}(\sigma_3, t) = (2, \text{op}_1) \cdot (3, \text{op}_1) )</td>
<td>( \text{store}_{\varphi_3}(\text{obs}(\sigma_3, t)) = (\epsilon, (2, \text{op}_1)) )</td>
<td>( \text{obs}(E_{\varphi_3}(\text{obs}(\sigma_3, t))) = \text{obs}(\epsilon, t) = \epsilon )</td>
</tr>
<tr>
<td>( t \in [3.5, 6[ )</td>
<td>( \text{obs}(\sigma_3, t) = (2, \text{op}_1) \cdot (3, \text{op}_1) \cdot (3.5, \text{op}) )</td>
<td>( \text{store}_{\varphi_3}(\text{obs}(\sigma_3, t)) = (\epsilon, (2, \text{op}_1) \cdot (3.5, \text{op})) )</td>
<td>( \text{obs}(E_{\varphi_3}(\text{obs}(\sigma_3, t))) = \text{obs}(\epsilon, t) = \epsilon )</td>
</tr>
<tr>
<td>( t \in [6, \infty[ )</td>
<td>( \text{obs}(\sigma_3, t) = (2, \text{op}_1) \cdot (3, \text{op}_1) \cdot (3.5, \text{op}) \cdot (6, \text{op}_2) )</td>
<td>( \text{store}_{\varphi_3}(\text{obs}(\sigma_3, t)) = ((6, \text{op}_1) \cdot (8, \text{op}) \cdot (10, \text{op}_2), \epsilon) )</td>
<td>( \text{obs}(E_{\varphi_3}(\text{obs}(\sigma_3, t))) = \text{obs}((6, \text{op}_1) \cdot (8, \text{op}) \cdot (10, \text{op}_2), t) )</td>
</tr>
</tbody>
</table>
Proposition: Enforcement function vs requirements

The proposed definition of enforcement function satisfies the **soundness**, **transparency**, and **optimality** requirements.
A rule-based transition system:
- configurations keep track of
  - The sequence which is corrected and can be released as output: $\sigma_{ms}$
  - The input sequence read by the EM, but yet to be corrected, except for events that are suppressed: $\sigma_{mc}$.
  - a clock $t$ indicating the current time instant,
  - the current state $q$ reached after processing the sequence already released followed by the timed word in memory i.e., $E_{\varphi}(\text{obs}(\sigma, t))$. 

$$E(\sigma, t) \models \varphi$$
A rule-based transition system:

- configurations keep track of
  - The sequence which is corrected and can be released as output: $\sigma_{ms}$
  - The input sequence read by the EM, but yet to be corrected, except for events that are suppressed: $\sigma_{mc}$.
  - a clock $t$ indicating the current time instant,
  - the current state $q$ reached after processing the sequence already released followed by the timed word in memory i.e., $E_\varphi(\text{obs}(\sigma, t))$

- an initial configuration
A rule-based transition system:
- configurations keep track of
  - The sequence which is corrected and can be released as output: $\sigma_{ms}$
  - The input sequence read by the EM, but yet to be corrected, except for events that are suppressed: $\sigma_{mc}$.
  - a clock $t$ indicating the current time instant,
  - the current state $q$ reached after processing the sequence already released followed by the timed word in memory i.e., $E_\varphi(\text{obs}(\sigma, t))$
- an initial configuration
- rule-based transitions executing enforcement operations (cf. next slide)
Enforcement monitor: operations

1. **store-$$\varphi$$**

when update returns *ok*, $$\varphi$$ is satisfiable by the released stream followed by $$\sigma_{ms}$$ and by $$w$$ which minimally delays $$\sigma_{mc} \cdot (t, a)$$. Sequence $$w$$ is appended to $$\sigma_{ms}$$. 
Enforcement monitor: operations

1. **store-φ**

   When update returns ok, φ is satisfiable by the released stream followed by $\sigma_{ms}$ and by $w$ which minimally delays $\sigma_{mc} \cdot (t, a)$. Sequence $w$ is appended to $\sigma_{ms}$.

2. **store-sup-φ**

   When update returns bad, $\sigma_{mc} \cdot (t, a)$ cannot be extended to be corrected. Event $(t, a)$ is suppressed, and the configuration remains unchanged.
## Enforcement monitor: operations

1. **store-\(\varphi\)**

   when update returns ok, \(\varphi\) is satisfiable by the released stream followed by \(\sigma_{ms}\) and by \(w\) which minimally delays \(\sigma_{mc} \cdot (t, a)\). Sequence \(w\) is appended to \(\sigma_{ms}\).

2. **store-sup-\(\overline{\varphi}\)**

   when update returns bad, \(\sigma_{mc} \cdot (t, a)\) cannot be extended to be corrected. Event \((t, a)\) is suppressed, and the configuration remains unchanged.

3. **store-\(\overline{\varphi}\)**

   when update returns c-bad, \(\sigma_{mc} \cdot (t, a)\) cannot be corrected yet. Event \((t, a)\) is appended to \(\sigma_{mc}\).
Enforcement monitor: operations

1. **store-ϕ**
   when update returns ok, ϕ is satisfiable by the released stream followed by σ_{ms} and by w which minimally delays σ_{mc} · (t, a). Sequence w is appended to σ_{ms}.

2. **store-sup-ϕ**
   when update returns bad, σ_{mc} · (t, a) cannot be extended to be corrected. Event (t, a) is suppressed, and the configuration remains unchanged.

3. **store-ϕ**
   when update returns c-bad, σ_{mc} · (t, a) cannot be corrected yet. Event (t, a) is appended to σ_{mc}.

4. **dump**
   at t when σ_{ms} = (t, a) · σ'_{ms}, event (t, a) is released.

5. **idle**
   when no other rule applies, time elapses of δ.
**Implementation relation** between Enforcement Monitor and Enforcement Function

Given some property $\varphi$, at any time $t$, the input/output behavior of the synthesized enforcement monitor is the same as the one of the corresponding enforcement function at time $t$, i.e., $\text{obs}(E_\varphi(\sigma), t)$.

**Corollary**

Enforcement Monitors respect soundness, transparency, and optimality.
Enforcement Monitor: example

\[ t = 0 \] - Operation: none

\[ \Sigma \setminus \{req, gr\} \]
\[ x := 0 \]

\[ l_0 \]

\[ \Sigma \setminus \{req, gr\} \]
\[ gr, 15 \leq x \leq 20; \]
\[ x := 0 \]

\[ l_1 \]

\[ \Sigma \setminus \{gr\}; \]
\[ g, x < 15 \lor x > 20 \]

\[ l_2 \]

\[ \Sigma \]

\[ (3, r) \cdot (13, g) \cdot (16, g) \cdot (21, r) \]

\[ \begin{array}{c|c|c}
q & \sigma_{ms} & \sigma_{mc} \\
(l_0, 0) & \epsilon & \epsilon \\
\end{array} \]
Enforcement Monitor: example

$t = 3$ - Operation: idle(3)
Enforcement Monitor: example

$$t = 3$$  
- Operation: store-$\varphi$

$$\begin{array}{c|c|c}
q & \sigma_{ms} & \sigma_{mc} \\
(l_0, 0) & \epsilon & (3, r)
\end{array}$$
Enforcement Monitor: example

$t = 13$ - Operation: idle(10)

\[
\begin{array}{ccc}
\sigma_{ms} & \sigma_{mc} \\
(l_0, 0) & (3, r) \\
\end{array}
\]
Enforcement Monitor: example

\[ t = 13 \quad - \quad \text{Operation: } \text{store-} \varphi \]

\[
\begin{array}{c}
\Sigma \setminus \{\text{req, gr}\} \\
\quad \text{req, } \quad x := 0
\end{array}
\]

\[
\begin{array}{c}
l_0 \quad \rightarrow \\
\quad \rightarrow \\
\quad \rightarrow
\end{array}
\]

\[
\begin{array}{c}
l_1 \quad \rightarrow \\
\quad \rightarrow \\
\quad \rightarrow
\end{array}
\]

\[
\begin{array}{c}
l_2 \quad \rightarrow \\
\quad \rightarrow \\
\quad \rightarrow
\end{array}
\]

\[ \epsilon \quad \leftarrow \quad (16, g) \cdot (21, r) \]

\[
\begin{array}{c|c|c|c}
q & \sigma_{ms} & \sigma_{mc} \\
\hline
(l_0, 0) & (13, r) \cdot (28, g) & \epsilon
\end{array}
\]
Enforcement Monitor: example

$t = 13$ - Operation: dump

$$\Sigma \setminus \{req, gr\} \quad req, \quad x := 0$$

$$\Sigma \setminus \{req, gr\}$$

$$gr, 15 \leq x \leq 20; \quad x := 0$$

$$gr$$

$$\Sigma \setminus \{gr\}; \quad g, x < 15 \lor x > 20$$

$$(13, r) \quad \leftarrow$$

$$(16, g) \cdot (21, r) \quad \leftarrow$$

<table>
<thead>
<tr>
<th>$q$</th>
<th>$\sigma_{ms}$</th>
<th>$\sigma_{mc}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(l_0, 0)$</td>
<td>$(28, g)$</td>
<td>$\epsilon$</td>
</tr>
</tbody>
</table>
Enforcement Monitor: example

\[ t = 16 \quad - \quad \text{Operation: idle}(3) \]

\[
\begin{align*}
\Sigma \setminus \{\text{req, gr}\} &\quad \text{req,} \\
&\quad x := 0 \\
\rightarrow &\quad l_0 \\
\Sigma \setminus \{\text{req, gr}\} &\quad \text{gr,} \\
&\quad 15 \leq x \leq 20; \\
&\quad x := 0 \\
\rightarrow &\quad l_1 \\
\Sigma \setminus \{\text{gr}\}; &\quad g, x < 15 \lor x > 20 \\
\rightarrow &\quad l_2 \\
\Sigma &\quad (13, r) \quad \leftarrow
\end{align*}
\]

\[
\begin{array}{c|c|c|c}
 q & \sigma_{ms} & \sigma_{mc} \\
(16, g) \cdot (21, r) & (28, g) & \epsilon
\end{array}
\]
Enforcement Monitor: example

\[ t = 16 \]

Operation: \( \text{store-sup-}\overline{\varphi} \)

\begin{array}{c|c|c|c}
 q & \sigma_{ms} & \sigma_{mc} \\
(l_0, 0) & (28, g) & \epsilon \\
\end{array}
Enforcement Monitor: example

$\begin{align*}
\begin{array}{c}
t = 21 \quad \text{Operation:} \quad \text{idle}(5)
\end{array}
\end{align*}$

\begin{align*}
\begin{array}{c}
\Sigma \setminus \{\text{req, gr}\} \quad \text{req,} \\
x := 0
\end{array}
\end{align*}

\begin{align*}
\begin{array}{c}
l_0 \quad \text{gr,} \\
15 \leq x \leq 20; \\
x := 0
\end{array}
\end{align*}

\begin{align*}
\begin{array}{c}
l_1 \quad \Sigma \setminus \{\text{req, gr}\}
\end{array}
\end{align*}

\begin{align*}
\begin{array}{c}
l_2 \quad \text{gr,} \\
x < 15 \lor x > 20
\end{array}
\end{align*}

\begin{align*}
\begin{array}{c}
\Sigma
\end{array}
\end{align*}

\begin{align*}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
13, r
\end{array}
\end{array}
\end{array}
\end{array}
\end{align*}

\begin{align*}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
21, r
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{align*}

\begin{align*}
\begin{array}{c|c|c}
q & \sigma_{ms} & \sigma_{mc} \\
(l_0, 0) & (28, g) & \epsilon
\end{array}
\end{align*}
Enforcement Monitor: example

\[ t = 21 \]

- Operation: \[ \text{store-} \varphi \]

\[
\begin{align*}
\Sigma \setminus \{\text{req, gr}\} & \xrightarrow{\text{req}, x := 0} \Sigma \setminus \{\text{req, gr}\} \\
\Sigma \setminus \{\text{req, gr}\} & \xrightarrow{\text{gr}, 15 \leq x \leq 20; x := 0} \Sigma \setminus \{\text{gr}\}; g, x < 15 \lor x > 20 \\
\Sigma & \xrightarrow{\Sigma} \Sigma
\end{align*}
\]

\[
\begin{array}{|c|c|c|}
\hline
q & \sigma_{ms} & \sigma_{mc} \\
\hline
(l_0, 0) & (28, g) & (21, r) \\
\hline
\end{array}
\]
Enforcement Monitor: example

\[
t = 28 - \text{Operation: \ idle(7)}
\]

\[
\begin{array}{c|c|c|c}
q & \sigma_{ms} & \sigma_{mc} \\
(l_0, 0) & (28, g) & (21, r)
\end{array}
\]
Enforcement Monitor: example

$t = 28$

Operation: $dump$

$\Sigma \setminus \{ req, gr \}$

$l_0$ to $l_1$

$gr, 15 \leq x \leq 20$
$x := 0$

$l_1$ to $l_2$

$\Sigma \setminus \{ gr \}$

$l_2$ to $\Sigma$

$(13, r) \cdot (28, g)$

$q | \sigma_{ms} | \sigma_{mc}$

$(l_0, 0) | \epsilon | (21, r)$
Implementation

Algorithm: StoreProcess

$(t, q) \leftarrow (0, q_0)$
$(\sigma_{ms}, \sigma_{mc}) \leftarrow (\epsilon, \epsilon)$

while $tt$ do
    $(t, a) \leftarrow \text{await (event)}$
    $(q', \sigma_{mc}', \text{isPath}) \leftarrow \text{update}(q, \sigma_{mc}, (t, a))$
    if $\text{isPath} = \text{ok}$ then
        $\sigma_{ms} \leftarrow \sigma_{ms} \cdot \sigma_{mc}'$
        $\sigma_{mc} \leftarrow \epsilon$
        $q \leftarrow q'$
    else
        $\sigma_{mc} \leftarrow \sigma_{mc}'$
    end if
end while

Algorithm: DumpProcess

$d \leftarrow 0$

while $tt$ do
    $\text{await (} \sigma_{ms} \neq \epsilon)$
    $(t, a) \leftarrow \text{dequeue (} \sigma_{ms})$
    $\text{wait (} t - d)$
    $\text{dump (} a)$
end while
1 Specifying Timed Properties

2 Runtime Enforcement of Regular Timed Properties
   - Requirements on an Enforcement Mechanism
   - Functional Definition of an Enforcement Mechanism
   - Operational Description of an Enforcement Mechanism
   - Algorithmic Description of an Enforcement Mechanism

3 Conclusions and Future Work
Conclusions and Future Work

Enforcement monitoring for systems with timing requirements.

- For all regular timed property modeled as a timed automaton.
- Enforcement mechanisms described at several levels of abstraction (enforcement function, enforcement monitor and algorithms).

Future Work

- What are enforceable properties?
- More expressive formalisms such as context-free timed languages.
- Predictive enforcement.
- Enforcement with partial control.
- Implementing efficient enforcement monitors (in application scenarios).
Some remarks / questions

- is it possible to pre-compute some “up” operation on automata?
- are $\kappa$, $\kbar$ computable?
- what about approximations?
- lookahead?
- in $B_w$, can we change $\preceq_{dp}$ into $\preceq_{sdp}$ (i.e., anticipate that we can suppress some event to satisfy $\varphi$, but be sure that it will be effectively suppressed ($\kbar = \emptyset$ when this event arrives))? otherwise we run the risk to loose soundness. NB: the objective of trying to minimize suppression of events, is antagonist with the objective of trying to anticipate.