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
Runtime verification and enforcement (monitors)

Runtime verification and enforcement:

- A monitor observes the execution of a system (e.g., trace, log, messages).
- No system model.
- A correctness property $\varphi$. 

Runtime verification

Does the run satisfy the property?

Input: stream of events.

Output: stream of verdicts.

Runtime enforcement

The run should satisfy the property.

Input: stream of events.

Output: stream of events (should satisfy the property).
Runtime verification and enforcement (monitors)

Runtime verification and enforcement:

- A monitor observes the execution of a system (e.g., trace, log, messages).
- No system model.
- A correctness property $\varphi$.

**Runtime verification**

- Does the run satisfy the property?
- Input: stream of events.
- Output: stream of **verdicts**.

$\sigma \models \varphi$?

$$\sigma \in \Sigma^\infty$$
Runtime verification and enforcement (monitors)

Runtime verification and enforcement:
- A monitor observes the execution of a system (e.g., trace, log, messages).
- No system model.
- A correctness property $\varphi$.

**Runtime verification**

- Does the run satisfy the property?
- Input: stream of events.
- Output: stream of **verdicts**.

**Runtime enforcement**

- The run should satisfy the property.
- Input: stream of events.
- Output: stream of **events** (should satisfy the property).
Enforcement monitoring - untimed case

- Dedicated to a property $\varphi$.
- Possibly augmented with a memorization mechanism.

**Enforcement mechanism (EM)**

An EM modifies the current execution sequence (sometimes like a “filter”).

- **reads** an input sequence $\sigma \in \Sigma^*$.
- **outputs** a new sequence $o \in \Sigma^*$.
- **endowed with** a set of enforcement primitives:
  - operate on the memorization mechanism,
  - delete or insert events using the memory content and the current input.

An EM behaves as a function $E : \Sigma^* \rightarrow \Sigma^*$. 
Motivations for *timed* enforcement

Allow specifying desired behavior of a system more precisely (time constraints between events).

- 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.

Many application domains:
- Real-time embedded systems, monitor hardware failures, communication protocols, web services and many more.

Examples of monitor usage:
- Firewall to prevent DOS attack ensuring minimal delay between input events;
- Checking pre-conditions of a service in web applications.
Motivations for \textit{timed} enforcement

Specifying the timing behavior

Allow specifying desired behavior of a system more precisely (time constraints between events).
Motivations for *timed* enforcement

Specifying the timing behavior

Allow specifying desired behavior of a system more precisely (time constraints between events).

- After action “a”, action “b” should occur
Motivations for *timed* enforcement

Specifying the timing behavior

Allow specifying desired behavior of a system more precisely (time constraints between events).

- After action “a”, action “b” should occur with a delay of at least 5 time units between them.
Motivations for *timed* enforcement

Specifying the timing behavior

Allow specifying desired behavior of a system more precisely (time constraints between events).

- 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.
Motivations for *timed* enforcement

**Specifying the timing behavior**

Allow specifying desired behavior of a system more precisely (time constraints between events).

- 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.

**Many application domains**

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

Runtime Enforcement of **Untimed** properties

- Enforceable security policies – Fred B. Schneider et al.
- Enforcement Monitoring wrt. the Safety-Progress Classification of Properties – Yliès Falcone et al.
- Runtime enforcement of non-safety policies – Jay Ligatti et al.
Related work on monitoring

Runtime Enforcement of Untimed properties

- Enforceable security policies – Fred B. Schneider et al.
- Enforcement Monitoring wrt. the Safety-Progress Classification of Properties – Yliès Falcone et al.
- Runtime enforcement of non-safety policies – Jay Ligatti et al.

Runtime Verification of Timed properties

Efforts mainly to verify timed properties at runtime:

- Runtime verification of TLTL – Andreas Bauer et al.
- The Analog Monitoring Tool.(monitoring specifications over continuous signals) – Dejan Nickovic et al.
- Safe runtime verification of real-time properties – Christian Colombo et al.
Problem tackled and Contributions

ϕ is a timed property

\[ o \leq \sigma \]

\[ o \models \varphi! \]

A formal framework for runtime enforcement of timed properties
Problem tackled and Contributions

ϕ is a timed property

A formal framework for runtime enforcement of timed properties

- Any regular timed property ϕ as input.
Problem tackled and Contributions

$\varphi$ is a timed property

A formal framework for runtime enforcement of timed properties

- Any regular timed property $\varphi$ as input.
- Enforcement mechanism adds additional delays between input actions in order to satisfy the property. – works as a “delay”
A formal framework for runtime enforcement of timed properties

- Any regular timed property $\varphi$ as input.
- Enforcement mechanism adds additional delays between input actions in order to satisfy the property. – works as a “delayer”
- A general definition of mechanisms for regular properties.
Problem tackled and Contributions

φ is a timed property

\[
\begin{align*}
\text{timed events} & \rightarrow \text{Enforcement Monitor} \\
o \preceq \sigma & \rightarrow \text{timed Memory} \\
o \models \varphi & \rightarrow \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*
\end{align*}
\]

A formal framework for runtime enforcement of timed properties

- Any regular timed property \( \varphi \) as input.
- Enforcement mechanism adds additional delays between input actions in order to satisfy the property. – works as a “delayer”
- A general definition of mechanisms for regular properties.
- Optimizations for safety and co-safety properties.
Problem tackled and Contributions

ϕ is a timed property

\[ o \preceq \sigma \]
\[ o \models \varphi \]

A formal framework for runtime enforcement of timed properties

- Any regular timed property \( \varphi \) as input.
- Enforcement mechanism adds additional delays between input actions in order to satisfy the property. – works as a “delayer”
- A general definition of mechanisms for regular properties.
- Optimizations for safety and co-safety properties.
- Enforcement mechanisms at several levels of abstraction (facilitating the design and implementation of such mechanisms).
Problem tackled and Contributions

\[ \varphi \text{ is a timed property} \]

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

A formal framework for runtime enforcement of timed properties

- Any regular timed property \( \varphi \) as input.
- Enforcement mechanism adds additional delays between input actions in order to satisfy the property. – \textit{works as a “delayer”}
- A general definition of mechanisms for regular properties.
- Optimizations for safety and co-safety properties.
- Enforcement mechanisms at \textit{several levels of abstraction} (facilitating the design and implementation of such mechanisms).
- Exhibiting a notion of \textit{non-enforceable properties}. 
Outline - Run-time Enforcement of Regular Timed Properties

1. Introduction
2. Specifying Timed Properties
3. Runtime Enforcement of Regular Timed Properties
4. Conclusions and Future Work
Specifying timed properties

- Input/output sequences are timed words:
  \[ \sigma = (\delta_1, a_1) \cdot (\delta_2, a_2) \cdot \ldots \cdot (\delta_n, a_n), \delta_i \in \mathbb{R}_{\geq 0}, a_i \in \Sigma. \]

- Property:
  - defined by a regular timed language \( \varphi \subseteq (\mathbb{R}_{\geq 0} \times \Sigma)^* \),
  - specified by a TA \( \mathcal{A}_\varphi \).
Specifying timed properties

- Input/output sequences are **timed words**:
  \[ \sigma = (\delta_1, a_1) \cdot (\delta_2, a_2) \cdots (\delta_n, a_n), \delta_i \in \mathbb{R}_{\geq 0}, a_i \in \Sigma. \]

- Property:
  - defined by a regular timed language \( \varphi \subseteq (\mathbb{R}_{\geq 0} \times \Sigma)^* \),
  - specified by a TA \( A_\varphi \).

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

- Input/output sequences are **timed words**:
  \[ \sigma = (\delta_1, a_1) \cdot (\delta_2, a_2) \cdots (\delta_n, a_n), \delta_i \in \mathbb{R}_{\geq 0}, a_i \in \Sigma. \]

- Property:
  - defined by a regular timed language \( \varphi \subseteq (\mathbb{R}_{\geq 0} \times \Sigma)^* \),
  - specified by a TA \( A_\varphi \).

---

Safety, co-safety and response properties specified by TAs

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

\[ \Sigma \setminus \{ req \} \quad \Sigma \setminus \{ req \} \quad \Sigma \]

\[ l_0 \xrightarrow{req, x := 0} l_1 \xrightarrow{req, x < 5} l_2 \]

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

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

- **Input/output sequences are** timed words:
  \[\sigma = (\delta_1, a_1) \cdot (\delta_2, a_2) \cdots (\delta_n, a_n), \delta_i \in \mathbb{R}_{\geq 0}, a_i \in \Sigma.\]
- **Property:**
  - defined by a regular timed language \(\varphi \subseteq (\mathbb{R}_{\geq 0} \times \Sigma)^*\),
  - specified by a TA \(A_{\varphi}\).

Safety, co-safety and response properties specified by TAs

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

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

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

- **Input/output sequences are timed words:**
  \[ \sigma = (\delta_1, a_1) \cdot (\delta_2, a_2) \cdots (\delta_n, a_n), \delta_i \in \mathbb{R}_{\geq 0}, a_i \in \Sigma. \]

- **Property:**
  - defined by a regular timed language \( \varphi \subseteq (\mathbb{R}_{\geq 0} \times \Sigma)^* \),
  - specified by a TA \( A_\varphi \).

### 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

\[
\begin{align*}
\Sigma \setminus \{\text{req, gr}\} & \quad \text{req,} \\
x & := 0
\end{align*}
\]

\[
\begin{align*}
l_0 & \quad \Sigma \setminus \{\text{req, gr}\} \\
\text{gr, } 15 \leq x \leq 20; \\
x & := 0
\end{align*}
\]

\[
\begin{align*}
l_1 & \quad \Sigma \setminus \{\text{gr}\}; \\
g, x < 15 \lor x > 20
\end{align*}
\]

\[
\begin{align*}
l_2 & \quad \Sigma
\end{align*}
\]

\[
\begin{align*}
\Sigma & = \{\text{req, gr}\} \\
(3, \text{req}) \cdot (15, \text{gr}) \cdot (5, \text{req}) \cdot (19, \text{gr})
\end{align*}
\]

\[
\epsilon \models \varphi.
\]
Example: response property

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

- \( r \), \( x := 0 \)
- \( g \), \( 15 \leq x \leq 20 \)
- \( x := 0 \)
- \( g \), \( x < 15 \lor x > 20 \)

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

- \( (3, \text{req}) \cdot (15, \text{gr}) \cdot (5, \text{req}) \cdot (19, \text{gr}) \)

\[ \epsilon \models \varphi. \]

\( (3, \text{req}) \not\models \varphi. \)
Example: response property

\[ \sum \setminus \{\text{req}, \text{gr}\} \]

\[ \text{req}, \quad x := 0 \]

\[ \sum \setminus \{\text{gr}\}; \]

\[ g, x < 15 \lor x > 20 \]

\[ \sum \]

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

\[ (3, \text{req}) \cdot (15, \text{gr}) \cdot (5, \text{req}) \cdot (19, \text{gr}) \]

\[ \epsilon \models \varphi. \]

\[ (3, \text{req}) \not\models \varphi. \]

\[ (3, \text{req}) \cdot (15, \text{gr}) \models \varphi. \]
Example: response property

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

\[ \xrightarrow{\text{req, } x := 0} \]

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

\[ \xrightarrow{\text{gr, } 15 \leq x \leq 20; \ x := 0} \]

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

\[ \Sigma \]

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

\[ (3, \text{req}) \cdot (15, \text{gr}) \cdot (5, \text{req}) \cdot (19, \text{gr}) \]

\[ \epsilon \models \varphi. \]

\[ (3, \text{req}) \not\models \varphi. \]

\[ (3, \text{req}) \cdot (15, \text{gr}) \models \varphi. \]

\[ (3, \text{req}) \cdot (15, \text{gr}) \cdot (5, \text{req}) \not\models \varphi. \]
Example: response property

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

\[ (3, \text{req}) \cdot (15, \text{gr}) \cdot (5, \text{req}) \cdot (19, \text{gr}) \]

Remark: response properties are neither prefix nor extension closed.
Major Challenges

Major challenges when (possibly) correcting an input sequence:

- **safety properties**: after each event, the decision is made (whether it can be corrected or not).

\[
\begin{align*}
\Sigma \{\text{req}\} & \quad \rightarrow \quad \Sigma \{\text{req}\} \\
\sigma_0 \quad \rightarrow \quad \sigma_1 \quad \leftarrow \quad \sigma_2 \\
\text{req, } x := 0 & \quad \rightarrow \quad \text{req, } x < 5 \\
\text{req, } x \geq 5, \quad x := 0 & \quad \leftarrow \quad \text{req, } x < 5
\end{align*}
\]
Major Challenges

Major challenges when (possibly) correcting an input sequence:

- safety properties: after each event, the decision is made (whether it can be corrected or not).
- co-safety properties: after each event, we check starting from the first event, whether the entire sequence can be corrected.
Major Challenges

Major challenges when (possibly) correcting an input sequence:

- **safety properties**: after each event, the decision is made (whether it can be corrected or not).

- **co-safety properties**: after each event, we check starting from the first event, whether the entire sequence can be corrected.

- **response properties**:
  - we cannot decide for each event soon after it is observed.
  - we do not check/correct from the first event since we want to correct and output chunk of sequences as soon as possible.
Outline - Runt. Enforcement of Regular Timed Properties

1. Introduction

2. Specifying Timed Properties

3. 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
   - A note on Non-enforceable Properties

4. Conclusions and Future Work
Summary of the approach

Given some timed property $\varphi$:

$$\sigma \subseteq \varphi!$$

What can an enforcement mechanism do?

- CANNOT insert nor delete events.
- CANNOT change the order of events.
- CAN increase the delay between actions.

$\rightarrow$ the enforcement monitor is a “delayer.”
Requirements for any enforcement mechanism for $\varphi$

Functional definition (satisfies the requirements):
- description of the input/output behavior;
- composition of 3 functions: process input, computing the delayed timed word, and process output,

Enforcement monitor:
- description of the operational behavior;
- a rule-based transition system with enforcement operations,

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

- **Requirements** for any enforcement mechanism for $\varphi$
- **Functional definition** (satisfies the requirements):
  - description of the input/output behavior;
  - composition of 3 functions: process input, computing the delayed timed word, and process output,
- **Enforcement monitor**:
  - description of the operational behavior;
  - a rule-based transition system with enforcement operations,
- **Implementation**: translation of the EM semantic rules into algorithms.
Summary of the approach

- **Requirements** for any enforcement mechanism for $\varphi$
- **Functional definition** (satisfies the requirements):
  - description of the input/output behavior;
  - composition of 3 functions: process input, computing the delayed timed word, and process output,
- **Enforcement monitor**:
  - description of the operational behavior;
  - a rule-based transition system with enforcement operations,
- **Implementation**: translation of the EM semantic rules into algorithms.
**Summary of the approach**

- **Requirements** for any enforcement mechanism for $\varphi$
- **Functional definition** (satisfies the requirements):
  - description of the input/output behavior;
  - composition of 3 functions: process input, computing the delayed timed word, and process output,
- **Enforcement monitor**:
  - description of the operational behavior;
  - a rule-based transition system with enforcement operations,
- **Implementation**: translation of the EM semantic rules into algorithms.
Outline - Runtime Enforcement of Regular Timed Properties

1 Introduction

2 Specifying Timed Properties

3 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
   - A note on Non-enforceable Properties

4 Conclusions and Future Work
Specified on an enforcement function for $\varphi$

$$E_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \rightarrow (\mathbb{R}_{\geq 0} \times \Sigma)^*.$$
Requirements on an Enforcement Mechanism (1)

The input and output of the mechanism are timed words

\[ E_\varphi(\sigma, t) \]

\( \sigma \)

\( \rightarrow \)

timed words
Soundness: the output is correct

\[ \text{Snd} \quad E\varphi(\sigma, t) \neq \epsilon \implies \exists t' \geq t : E\varphi(\sigma, t') \models \varphi. \]
Requirements on an Enforcement Mechanism (1)

**Transparency:** events are preserved and delayed

\[ \text{Tr } E_\varphi(\sigma, t) \preceq_d \text{obs}(\sigma, t), \text{ where } \sigma' \preceq_d \sigma \text{ means} \]
Optimality: output is produced ASAP ... but not too soon
Requirements on an Enforcement Mechanism (2)

**Optimality:** output is produced ASAP ... but not too soon

\[ E_\varphi(\sigma, t) \]

![Diagram showing timed words and enforcement mechanism](image-url)
Optimality: output is produced ASAP ... but not too soon
Requirements on an Enforcement Mechanism (2)

Optimality: output is produced ASAP ... but not too soon

- ideal sequence
- time needed to read the chunk
- timed words
- chunk in memory

Requirements on an Enforcement Mechanism (2)

**Optimality:** output is produced ASAP ... but not too soon

The diagram illustrates the relationship between the ideal sequence, actual delay, ideal delay, time needed to read the chunk, and actual delay. The function $E_{\phi}(\sigma, t)$ represents the runtime enforcement of timed properties, indicating the time at which output is produced and ensuring it is not too soon.
Outline - Runtime Enforcement of Regular Timed Properties

1. Introduction

2. Specifying Timed Properties

3. 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
   - A note on Non-enforceable Properties

4. Conclusions and Future Work
Functional definition (1)

The functional definition describes the mechanism as a function

\[ E_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \rightarrow (\mathbb{R}_{\geq 0} \times \Sigma)^*. \]
Functional definition (1)

The functional definition describes the mechanism as a function

\[ E_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \rightarrow (\mathbb{R}_{\geq 0} \times \Sigma)^*. \]
The functional definition describes the mechanism as a function

$$E_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \rightarrow (\mathbb{R}_{\geq 0} \times \Sigma)^*.$$
**Functional definition (2)**

$$E_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \rightarrow (\mathbb{R}_{\geq 0} \times \Sigma)^*$$

$$E_\varphi(\sigma, t) = \text{obs}\left(\prod_1(\text{store}_\varphi(\text{obs}(\sigma, t))), t\right).$$
Functional definition (2)

$$E_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \rightarrow (\mathbb{R}_{\geq 0} \times \Sigma)^*$$

$$E_\varphi(\sigma, t) = \text{obs}(\Pi_1(\text{store}_\varphi(\text{obs}(\sigma, t))), t).$$

$$\text{store}_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \rightarrow (\mathbb{R}_{\geq 0} \times \Sigma)^* \times (\mathbb{R}_{\geq 0} \times \Sigma)^*$$

\text{store}_\varphi(\sigma) is a pair:

1. delayed correct prefix of \( \sigma \),
2. suffix of \( \sigma \) for which delays still have to be computed.
Functional definition (2)

\[ E_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \rightarrow (\mathbb{R}_{\geq 0} \times \Sigma)^* \]

\[ E_\varphi(\sigma, t) = \text{obs}\left( \Pi_1(\text{store}_\varphi(\text{obs}(\sigma, t))), t \right). \]

\[ \text{store}_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \rightarrow (\mathbb{R}_{\geq 0} \times \Sigma)^* \times (\mathbb{R}_{\geq 0} \times \Sigma)^* \]

\[ \text{store}_\varphi(\sigma) \text{ is a pair:} \]

1. delayed correct prefix of \( \sigma \),
2. suffix of \( \sigma \) for which delays still have to be computed.

\[ \text{store}_\varphi(\epsilon) = (\epsilon, \epsilon) \]
## Functional definition (2)

\[
E_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \rightarrow (\mathbb{R}_{\geq 0} \times \Sigma)^*
\]

\[
E_\varphi(\sigma, t) = \text{obs}\left(\Pi_1(\text{store}_\varphi(\text{obs}(\sigma, t))), t\right).
\]

\[
\text{store}_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \rightarrow (\mathbb{R}_{\geq 0} \times \Sigma)^* \times (\mathbb{R}_{\geq 0} \times \Sigma)^*
\]

\text{store}_\varphi(\sigma) \text{ is a pair:}

1. delayed correct prefix of \(\sigma\),
2. suffix of \(\sigma\) for which delays still have to be computed.

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

Suppose \((\sigma_s, \sigma_c) = \text{store}_\varphi(\sigma)\)
E_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \rightarrow (\mathbb{R}_{\geq 0} \times \Sigma)^*

E_\varphi(\sigma, t) = \text{obs}\left(\Pi_1(\text{store}_\varphi(\text{obs}(\sigma, t))), t\right).

\text{store}_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \rightarrow (\mathbb{R}_{\geq 0} \times \Sigma)^* \times (\mathbb{R}_{\geq 0} \times \Sigma)^*

\text{store}_\varphi(\sigma) \text{ is a pair:}

1. delayed correct prefix of \sigma,
2. suffix of \sigma for which delays still have to be computed.

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

Suppose \((\sigma_s, \sigma_c) = \text{store}_\varphi(\sigma)\)

\text{store}_\varphi(\sigma \cdot (\delta, a)) = \begin{cases} 
(\sigma_s \cdot \min_{\leq_{\text{lex}}} K, \epsilon) & \text{if } K \neq \emptyset \\
(\sigma_s, \sigma_c \cdot (\delta, a)) & \text{otherwise}
\end{cases}

\text{where } K \text{ is the set of possible corrected factors of } \sigma \text{ between positions } |\sigma_s| \text{ and } |\sigma_c| + 1 \text{ with a delay for the first event greater than } \text{time}(\sigma_c \cdot (\delta, a)).

(cf. details in the paper)
Functional definition: Example

\[ \Sigma = \{r, g\}. \]
\[ \sigma = (3, req) \cdot (10, gr) \cdot (3, req) \cdot (5, req). \]
Functional definition: Example

\[ \Sigma = \{ r, g \} \]
\[ \sigma = (3, req) \cdot (10, gr) \cdot (3, req) \cdot (5, req) \]

\[ t \in [0, 3[ \]

\[ \text{obs}(\sigma, t) = \epsilon \]
\[ \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon) \]
\[ E_\varphi(\sigma, t) = \text{obs}(\epsilon, t) \]
Functional definition: Example

\[ \Sigma = \{r, g\} . \]
\[ \sigma = (3, req) \cdot (10, gr) \cdot (3, req) \cdot (5, req) . \]

\[
\begin{align*}
\text{t} & \in [0, 3] \\
\text{obs}(\sigma, t) & = \epsilon \\
\text{store}_\varphi(\text{obs}(\sigma, t)) & = (\epsilon, \epsilon) \\
E_\varphi(\sigma, t) & = \text{obs}(\epsilon, t)
\end{align*}
\]

\[
\begin{align*}
\text{t} & \in [3, 13] \\
\text{obs}(\sigma, t) & = (3, req) \\
\text{store}_\varphi(\text{obs}(\sigma, t)) & = (\epsilon, (3, req)) \\
E_\varphi(\sigma, t) & = \text{obs}(\epsilon, t)
\end{align*}
\]
Functional definition: Example

\[ \Sigma = \{ r, g \} \]
\[ \sigma = (3, \text{req}) \cdot (10, \text{gr}) \cdot (3, \text{req}) \cdot (5, \text{req}). \]

<table>
<thead>
<tr>
<th>Interval</th>
<th>Condition</th>
<th>( \text{obs}(\sigma, t) )</th>
<th>( \text{store}_\varphi(\text{obs}(\sigma, t)) )</th>
<th>( E_\varphi(\sigma, t) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t \in [0, 3] )</td>
<td></td>
<td>( \epsilon )</td>
<td>( \epsilon, \epsilon )</td>
<td>( \text{obs}(\epsilon, t) )</td>
</tr>
<tr>
<td>( t \in [3, 13] )</td>
<td></td>
<td>( (3, \text{req}) )</td>
<td>( \epsilon, (3, \text{req}) )</td>
<td>( E_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
</tr>
<tr>
<td>( t \in [13, 16] )</td>
<td></td>
<td>( (3, \text{req}) \cdot (10, \text{gr}) )</td>
<td>( (13, \text{req}) \cdot (15, \text{gr}), \epsilon )</td>
<td>( E_\varphi(\sigma, t) = \text{obs}((13, \text{req}) \cdot (15, \text{gr}), t) )</td>
</tr>
</tbody>
</table>
Functional definition: Example

\[ \Sigma = \{ r, g \} \]
\[ \sigma = (3, req) \cdot (10, gr) \cdot (3, req) \cdot (5, req) \].

<table>
<thead>
<tr>
<th>Interval</th>
<th>Observation</th>
<th>Store ( \varphi )</th>
<th>( E_\varphi )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t \in [0, 3] )</td>
<td>( \text{obs}(\sigma, t) = \epsilon )</td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon) )</td>
<td>( E_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
</tr>
<tr>
<td>( t \in [3, 13] )</td>
<td>( \text{obs}(\sigma, t) = (3, req) )</td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req)) )</td>
<td>( E_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
</tr>
<tr>
<td>( t \in [13, 16] )</td>
<td>( \text{obs}(\sigma, t) = (3, req) \cdot (10, gr) )</td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = ((13, req) \cdot (15, gr), \epsilon) )</td>
<td>( E_\varphi(\sigma, t) = \text{obs}((13, req) \cdot (15, gr), t) )</td>
</tr>
<tr>
<td>( t \in [16, 21] )</td>
<td>( \text{obs}(\sigma, t) = (3, req) \cdot (10, gr) \cdot (3, req) )</td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = ((13, req) \cdot (15, gr), (3, req)) )</td>
<td>( E_\varphi(\sigma, t) = \text{obs}((13, req) \cdot (15, gr), t) )</td>
</tr>
</tbody>
</table>

\[ \Sigma \setminus \{ req, gr \} \]
\[ \text{req}, x := 0 \]
\[ \Sigma \setminus \{ req, gr \} \]
\[ \Sigma \setminus \{ gr \} \]
\[ g, x < 15 \lor x > 20 \]
\[ \Sigma \setminus \{ gr \} \]
\[ g, 15 \leq x \leq 20; \]
\[ x := 0 \]
\[ \Sigma \setminus \{ gr \} \]
\[ g, x < 15 \lor x > 20 \]
\[ \Sigma \setminus \{ gr \} \]
\[ g, x < 15 \lor x > 20 \]
\[ \Sigma \setminus \{ gr \} \]
\[ g, x < 15 \lor x > 20 \]
Functional definition: Example

\[ \Sigma = \{r, g\}. \]
\[ \sigma = (3, req) \cdot (10, gr) \cdot (3, req) \cdot (5, req). \]

<table>
<thead>
<tr>
<th>( t \in [0, 3] )</th>
<th>( \text{obs}(\sigma, t) = \epsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon) )</td>
</tr>
<tr>
<td></td>
<td>( E_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( t \in [3, 13] )</th>
<th>( \text{obs}(\sigma, t) = (3, req) )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req)) )</td>
</tr>
<tr>
<td></td>
<td>( E_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( t \in [13, 16] )</th>
<th>( \text{obs}(\sigma, t) = (3, req) \cdot (10, gr) )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = ((13, req) \cdot (15, gr), \epsilon) )</td>
</tr>
<tr>
<td></td>
<td>( E_\varphi(\sigma, t) = \text{obs}((13, req) \cdot (15, gr), \epsilon) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( t \in [16, 21] )</th>
<th>( \text{obs}(\sigma, t) = (3, req) \cdot (10, gr) \cdot (3, req) )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = ((13, req) \cdot (15, gr), (3, req)) )</td>
</tr>
<tr>
<td></td>
<td>( E_\varphi(\sigma, t) = \text{obs}((13, req) \cdot (15, gr), (3, req)) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( t \in [21, \infty] )</th>
<th>( \text{obs}(\sigma, t) = (3, req) \cdot (10, gr) \cdot (3, req) \cdot (5, req) )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = ((13, req) \cdot (15, gr), (3, req) \cdot (5, req)) )</td>
</tr>
<tr>
<td></td>
<td>( E_\varphi(\sigma, t) = \text{obs}((13, req) \cdot (15, gr), (3, req) \cdot (5, req)) )</td>
</tr>
</tbody>
</table>
The enforcement function satisfies the requirements

**Proposition: Enforcement function vs requirements**

The proposed definition of enforcement function satisfies the **soundness**, **transparency**, and **optimality** requirements.
Outline - Runtime Enforcement of Regular Timed Properties

1 Introduction

2 Specifying Timed Properties

3 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
   • A note on Non-enforceable Properties

4 Conclusions and Future Work
A *rule-based transition system*:

- configurations keep track of
  - the prefix of $\sigma$ that has been corrected but yet to be output ("good memory")
  - the suffix of $\sigma$ that cannot be corrected ("bad memory")
  - a clock reset at the moment of the last *input* event ("store clock")
  - a clock reset at the moment of the last *output* event ("dump clock")
  - a state in the semantics of the TA
A rule-based transition system:

- configurations keep track of
  - the prefix of \( \sigma \) that has been corrected but yet to be output ("good memory")
  - the suffix of \( \sigma \) that cannot be corrected ("bad memory")
  - a clock reset at the moment of the last input event ("store clock")
  - a clock reset at the moment of the last output event ("dump clock")
  - a state in the semantics of the TA

- an initial configuration
A rule-based transition system:
- configurations keep track of
  - the prefix of $\sigma$ that has been corrected but yet to be output ("good memory")
  - the suffix of $\sigma$ that cannot be corrected ("bad memory")
  - a clock reset at the moment of the last input event ("store clock")
  - a clock reset at the moment of the last output event ("dump clock")
  - a state in the semantics of the TA
- an initial configuration
- rule-based transitions executing enforcement operations (cf. next slide)
A *rule-based transition system*:

- configurations keep track of
  - the prefix of $\sigma$ that has been corrected but yet to be output ("good memory")
  - the suffix of $\sigma$ that cannot be corrected ("bad memory")
  - a clock reset at the moment of the last *input* event ("store clock")
  - a clock reset at the moment of the last *output* event ("dump clock")
  - a state in the semantics of the TA

- an initial configuration

- rule-based transitions executing enforcement operations (cf. next slide)

Remark: for safety and co-safety, some memories and clocks can be discarded.
Enforcement monitor: operations

1. **store-\(\overline{\varphi}\)**
   - when a new event is received and the new event *cannot make the property satisfied* by delaying.
   - updates “bad” memory and store clock
Enforcement monitor: operations

1. **store-\(\overline{\varphi}\)**
   - when a new event is received and the new event *cannot make the property satisfied* by delaying.
   - updates “bad” memory and store clock

2. **store-\(\varphi\)**
   - when a new event is received and the new event *can make the property satisfied* by delaying
   - updates “good” memory and dump clock
Enforcement monitor: operations

1. **store-ϕ**
   - when a new event is received and the new event *cannot make the property satisfied* by delaying.
   - updates “bad” memory and store clock

2. **store-φ**
   - when a new event is received and the new event *can make the property satisfied* by delaying
   - updates “good” memory and dump clock

3. **dump**
   - when an event in the good memory can be released
   - updates “good” memory and dump clock
## Enforcement monitor: operations

1. **store-$$\varphi$$**
   - when a new event is received and the new event *cannot make the property satisfied* by delaying.
   - updates “bad” memory and store clock

2. **store-$$\overline{\varphi}$$**
   - when a new event is received and the new event *can make the property satisfied* by delaying
   - updates “good” memory and dump clock

3. **dump**
   - when an event in the good memory can be released
   - updates “good” memory and dump clock

4. **idle**
   - when no other rule can
   - updates dump and store clocks
Enforcement Monitor: correctness

**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 one of the corresponding enforcement function.

**Corollary**

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

\( t = 0 \) - Executed operation: none

- Good Memory: \( \epsilon \)
- Bad Memory: \( \epsilon \)
- State: \((l_0, 0)\)
t = 3 - Executed operation: idle(3)

Good Memory: $\epsilon$
Bad Memory: $\epsilon$
State: $(l_0, 3)$
Enforcement Monitor: example

$t = 3$ - Executed operation: \textit{store-}$$\bar{\varphi}$$

- Good Memory: $\epsilon$
- Bad Memory: $(3, r)$
- State: $(l_0, 0)$
Enforcement Monitor: example

$t = 13$ - Executed operation: idle(10)

- Good Memory: $\epsilon$
- Bad Memory: $(3, r)$
- State: $(l_0, 0)$
Enforcement Monitor: example

- $t = 13$
- Executed operation: $store-\varphi$

Good Memory:
(13, r) \cdot (15, g)

Bad Memory:
$\epsilon$

State:
$l_0, 0$
**Enforcement Monitor: example**

- **$t = 13$**

- **Executed operation:** *dump*

- **Good Memory:** $(15, g)$

- **Bad Memory:** $\epsilon$

- **State:** $(l_0, 15)$
Enforcement Monitor: example

\[ t = 16 \]

- Executed operation: \( \text{idle}(3) \)

\[ \begin{align*}
\Sigma \setminus \{\text{req, gr}\} & \quad \text{req,} \\
\quad \quad \quad \quad \quad \quad \quad x := 0 & \quad \text{Σ} \setminus \{\text{req, gr}\} \\
\rightarrow l_0 & \quad \quad \rightarrow l_1 \quad \Sigma \setminus \{\text{req, gr}\} \\
gr, 15 \leq x \leq 20; & \quad \text{Σ \setminus \{gr\};} \\
\quad \quad \quad \quad \quad \quad \quad x := 0 & \quad \quad \quad \quad g, x < 15 \lor x > 20 \\
g & \quad \rightarrow l_2 \quad \Sigma \setminus \{gr\}; \\
(13, r) & \quad \leftarrow (3, r) \cdot (5, r)
\end{align*} \]

- Good Memory: \((15, g)\)
- Bad Memory: \(\epsilon\)
- State: \((l_0, 15)\)
Enforcement Monitor: example

- Executed operation: \( store-\overline{\varphi} \)

- Time: \( t = 16 \)

- State: \( (l_0, 15) \)

- Good Memory: \( (15, g) \)

- Bad Memory: \( (3, r) \)
Enforcement Monitor: example

$t = 21$

- Executed operation: idle(5)

- Pinisetty, Falcone, Jéron, Marchand (INRIA, UJF)
Enforcement Monitor: example

$t = 21$

- Executed operation: $store-\overline{\varphi}$

Good Memory: $(15, g)$

Bad Memory: $(3, r) \cdot (5, r)$

State: $(l_0, 15)$
Enforcement Monitor: example

$t = 28$

Executed operation: idle(7)

Good Memory: (15, g)

Bad Memory: (3, r) ∙ (5, r)

State: (l₀, 15)
Enforcement Monitor: example

$t = 28$

- Executed operation: $dump$

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

- Good Memory: $\epsilon$
- Bad Memory:
  - $(3, r) \cdot (5, r)$
- State: $(l_0, 15)$
Outline - Runtime Enforcement of Regular Timed Properties

1. Introduction

2. Specifying Timed Properties

3. 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
   - A note on Non-enforceable Properties

4. Conclusions and Future Work
Implementation

Algorithm: StoreProcess

\[(l, \nu) \leftarrow (l_0, [X \leftarrow 0])\]
\[\sigma_s, \sigma_c \leftarrow \epsilon\]
\[m_t \leftarrow 0\]
while tt do
    \[(\delta, a) \leftarrow \text{await} \ (\text{event})\]
    \[\sigma_c \leftarrow \sigma_c \cdot (\delta, a)\]
    \[m_t \leftarrow m_t + \delta\]
    \[(\sigma'_c, \text{isPath}) \leftarrow \text{update}(l, \nu, \sigma_c, m_t)\]
    if isPath = tt then
        \[m_t \leftarrow m_t - \text{time}(\sigma'_c)\]
        \[\sigma_s \leftarrow \sigma_s \cdot \sigma'_c\]
        \[(l, \nu) \leftarrow \text{post}(l, \nu, \sigma'_c)\]
        \[\sigma_c \leftarrow \epsilon\]
    end if
end while

Algorithm: DumpProcess

\[d \leftarrow 0\]
while tt do
    \[\text{await} \ (\sigma_s \neq \epsilon)\]
    \[(\delta, a) \leftarrow \text{dequeue} \ (\sigma_s)\]
    wait \(\delta - d\)
    dump \(a\)
    \[d \leftarrow 0\]
end while
Outline - Runtime Enforcement of Regular Timed Properties

1. Introduction

2. Specifying Timed Properties

3. 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
   - A note on Non-enforceable Properties

4. Conclusions and Future Work
Non-enforceable response properties

\[ \Sigma = \{\text{gr}, \text{req}\} \]
\[ \sigma = (3, \text{req}) \cdot (4, \text{gr}) \cdot (2, \text{req}) \cdot (6, \text{gr}) \]
Non-enforceable response properties

\[ \Sigma = \{ gr, req \}. \]
\[ \sigma = (3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr). \]

\[ t \in [0, 3[ \]
\[ \text{obs}(\sigma, t) = \epsilon \]
\[ \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon) \]
\[ \text{E}_\varphi(\sigma, t) = \text{obs}(\epsilon, t) \]
Non-enforceable response properties

\[ \Sigma = \{ gr, req \} . \]
\[ \sigma = (3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr) . \]

<table>
<thead>
<tr>
<th>( t \in [0, 3[ )</th>
<th>( t \in [3, 7[ )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{obs}(\sigma, t) = \epsilon )</td>
<td>( \text{obs}(\sigma, t) = (3, req) )</td>
</tr>
<tr>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon) )</td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req)) )</td>
</tr>
<tr>
<td>( E_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
<td>( E_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
</tr>
</tbody>
</table>
Non-enforceable response properties

\[ \Sigma = \{gr, req\}. \]
\[ \sigma = (3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr). \]

\[
\begin{align*}
\text{obs}(\sigma, t) &= \epsilon \\
\text{store}_\varphi(\text{obs}(\sigma, t)) &= (\epsilon, \epsilon) \\
E_\varphi(\sigma, t) &= \text{obs}(\epsilon, t)
\end{align*}
\]

\[
\begin{align*}
t \in [0, 3] \\
\text{obs}(\sigma, t) &= (3, req) \\
\text{store}_\varphi(\text{obs}(\sigma, t)) &= (\epsilon, (3, req)) \\
E_\varphi(\sigma, t) &= \text{obs}(\epsilon, t)
\end{align*}
\]

\[
\begin{align*}
t \in [3, 7] \\
\text{obs}(\sigma, t) &= (3, req) \cdot (4, gr) \\
\text{store}_\varphi(\text{obs}(\sigma, t)) &= (\epsilon, (3, req) \cdot (4, gr)) \\
E_\varphi(\sigma, t) &= \text{obs}(\epsilon, t)
\end{align*}
\]

\[
\begin{align*}
t \in [7, 9] \\
\text{obs}(\sigma, t) &= (3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr) \\
\text{store}_\varphi(\text{obs}(\sigma, t)) &= (\epsilon, (3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr)) \\
E_\varphi(\sigma, t) &= \text{obs}(\epsilon, t)
\end{align*}
\]
Non-enforceable response properties

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

\[ \sigma = (3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr) . \]

- \( t \in [0, 3[ \):
  \[ \begin{align*}
  \text{obs}(\sigma, t) &= \epsilon \\
  \text{store}_\varphi(\text{obs}(\sigma, t)) &= (\epsilon, \epsilon) \\
  E_\varphi(\sigma, t) &= \text{obs}(\epsilon, t)
  \end{align*} \]

- \( t \in [3, 7[ \):
  \[ \begin{align*}
  \text{obs}(\sigma, t) &= (3, req) \\
  \text{store}_\varphi(\text{obs}(\sigma, t)) &= (\epsilon, (3, req)) \\
  E_\varphi(\sigma, t) &= \text{obs}(\epsilon, t)
  \end{align*} \]

- \( t \in [7, 9[ \):
  \[ \begin{align*}
  \text{obs}(\sigma, t) &= (3, req) \cdot (4, gr) \\
  \text{store}_\varphi(\text{obs}(\sigma, t)) &= (\epsilon, (3, req) \cdot (4, gr)) \\
  E_\varphi(\sigma, t) &= \text{obs}(\epsilon, t)
  \end{align*} \]

- \( t \in [9, 15[ \):
  \[ \begin{align*}
  \text{obs}(\sigma, t) &= (3, req) \cdot (4, gr) \cdot (2, req) \\
  \text{store}_\varphi(\text{obs}(\sigma, t)) &= (\epsilon, (3, req) \cdot (4, gr) \cdot (2, req)) \\
  E_\varphi(\sigma, t) &= \text{obs}(\epsilon, t)
  \end{align*} \]

\[ \Sigma \setminus \{ req, gr \} \]

\( x \leq 5 \)

\( \Sigma \)

\( l_0 \) \quad \text{gr, } x \leq 10; \quad x := 0 \quad \text{gr, } x > 10 \quad \Sigma \setminus \{ req, gr \} \)

\( l_1 \) \quad \text{req, } x < 5 \quad \Sigma \setminus \{ req, gr \} \)

\( l_2 \) \quad \text{req, } x \leq 5 \quad \text{gr} \)
Non-enforceable response properties

\[ \Sigma = \{ \text{gr}, \text{req} \} \]
\[ \sigma = (3, \text{req}) \cdot (4, \text{gr}) \cdot (2, \text{req}) \cdot (6, \text{gr}) \]

\[
\begin{align*}
t \in [0, 3] & \quad \text{obs}(\sigma, t) = \epsilon \\
& \quad \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon) \\
& \quad E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)
\end{align*}
\]

\[
\begin{align*}
t \in [3, 7] & \quad \text{obs}(\sigma, t) = (3, \text{req}) \\
& \quad \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, \text{req})) \\
& \quad E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)
\end{align*}
\]

\[
\begin{align*}
t \in [7, 9] & \quad \text{obs}(\sigma, t) = (3, \text{req}) \cdot (4, \text{gr}) \\
& \quad \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, \text{req}) \cdot (4, \text{gr})) \\
& \quad E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)
\end{align*}
\]

\[
\begin{align*}
t \in [9, 15] & \quad \text{obs}(\sigma, t) = (3, \text{req}) \cdot (4, \text{gr}) \cdot (2, \text{req}) \\
& \quad \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, \text{req}) \cdot (4, \text{gr}) \cdot (2, \text{req})) \\
& \quad E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)
\end{align*}
\]

\[
\begin{align*}
t \in [15, \infty] & \quad \text{obs}(\sigma, t) = (3, \text{req}) \cdot (4, \text{gr}) \cdot (2, \text{req}) \cdot (6, \text{gr}) \\
& \quad \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, \text{req}) \cdot (4, \text{gr}) \cdot (2, \text{req}) \cdot (6, \text{gr})) \\
& \quad E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)
\end{align*}
\]
<table>
<thead>
<tr>
<th>Outline - Run. Enforcement of Regular Timed Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
</tr>
<tr>
<td>2. Specifying Timed Properties</td>
</tr>
<tr>
<td>3. Runtime Enforcement of Regular Timed Properties</td>
</tr>
<tr>
<td>4. Conclusions and Future Work</td>
</tr>
</tbody>
</table>
Conclusions and Future Work

Enforcement monitoring for systems with timing requirements.

- Input any regular timed property modeled as a timed automaton.
- Enforcement mechanisms described at several levels of abstraction (enforcement function, enforcement monitor and algorithms).
- Exhibiting a notion of non-enforceable properties.

Future Work

- Delineate the set of enforceable response properties.
- More expressive formalisms such as context-free timed languages.
- Requirements with constraints on data and time – cf. WODES’14.
- Alternative enforcement primitives (reduce delays, suppress events).
- Implementing efficient enforcement monitors (in application scenarios).