Runtime Enforcement of Timed Properties

Srinivas Pinisetty

INRIA Rennes, Team: Sumo, Project: ANR Vacsim

23 January 2015, INRIA, Rennes

Jury

Martin Leucker, Université de Lübeck (Reviewer)
Didier Lime, École Centrale de Nantes (Reviewer)
Frédéric Herbreteau, ENSEIRB Bordeaux (Examiner)
Sophie Pinchinat, Université de Rennes 1 (Examiner)
Thierry Jéron, INRIA Rennes (Advisor)
Hervé Marchand, INRIA Rennes (Co-advisor)
Yliès Falcone, Université de Grenoble 1 (Co-advisor)
Traditional design and verification

- Cannot guarantee absence of errors.
- Late detection of errors (some times after deployment).
- Inadequate for safety-critical systems.

Some software bugs

- Ariane 5, Therac-25, Toyota’s ETCS.
**Formal verification techniques**

- **Spec:** Set of requirements/properties (LTL, CTL, ...).
- Abstraction of system/program (automata, ...).
- **Static:** Model checking, static analysis.
- **Dynamic:** Testing, **runtime verification/enforcement**.
Runtime verification and enforcement (monitors)

Runtime verification and enforcement:

- Monitor execution of a system (e.g., trace, log, messages).
- No system model.
- A formal requirement: Property $\varphi$. 
Runtime verification and enforcement (monitors)

Runtime verification and enforcement:

- Monitor execution of a system (e.g., trace, log, messages).
- No system model.
- A formal requirement: Property \( \varphi \).

### Runtime verification

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

\[
\sigma \models \varphi? \\
\sigma \in \Sigma^\infty \\
\omega \in \mathbb{D}^\infty
\]
Runtime verification and enforcement (monitors)

Runtime verification and enforcement:

- Monitor execution of a system (e.g., trace, log, messages).
- No system model.
- A formal requirement: Property $\varphi$.

**Runtime verification**

- Monitor verdicts events $\sigma \in \Sigma^\infty$
- Verdict: $\sigma \models \varphi$?

**Input:** stream of events.

**Output:** stream of **verdicts**.

**Does the run satisfy the property?**

**Runtime enforcement**

- Monitor events $\sigma \leq \sigma$
- Events: $\sigma \models \varphi$!

**Input:** stream of events.

**Output:** stream of events (should satisfy the property).

**Allowed to block/modify input.**
Motivations for *timed* enforcement

Specifying 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
Motivations for *timed* enforcement

Specifying 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.
Motivations for *timed* enforcement

Specifying 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.
Context and objectives

Objective

Given a timed property, synthesize enforcement mechanism operating at runtime.

Enforcer between event emitter and event receiver.

Reliable channels, safe communication.

Mechanism to transform input $\sigma$ into output $o$ such that $o \models \varphi$.

Three directions

Expressiveness of the supported specification formalism.

Power of the enforcement mechanism.

Implementability.
Objective

- Given a timed property, synthesize enforcement mechanism operating at runtime.
- Enforcer between event emitter and event receiver.
- Reliable channels, safe communication.
- Mechanism to transform input \( \sigma \) in to output \( o \) such that \( o \models \varphi \).
### Context and objectives

#### Objective
- Given a timed property, synthesize **enforcement mechanism** operating at **runtime**.
- Enforcer between event emitter and event receiver.
- Reliable channels, safe communication.
- Mechanism to transform input $\sigma$ into output $o$ such that $o \models \varphi$.

#### Three directions
- Expressiveness of the supported specification formalism.
Objective

- Given a timed property, synthesize enforcement mechanism operating at runtime.
- Enforcer between event emitter and event receiver.
- Reliable channels, safe communication.
- Mechanism to transform input $\sigma$ into output $o$ such that $o \models \varphi$.

Three directions

- Expressiveness of the supported specification formalism.
- Power of the enforcement mechanism.
Objective

- Given a timed property, synthesize enforcement mechanism operating at runtime.
- Enforcer between event emitter and event receiver.
- Reliable channels, safe communication.
- Mechanism to transform input $\sigma$ in to output $o$ such that $o \models \varphi$.

Three directions

- Expressiveness of the supported specification formalism.
- Power of the enforcement mechanism.
- Implementability.
Contributions

A formal framework for runtime enforcement of timed properties
Contributions

A formal framework for runtime enforcement of timed properties

- General definitions for all regular timed properties.
Contributions

A formal framework for runtime enforcement of timed properties

- General definitions for all regular timed properties.
- Work as *delayers*: either increase input dates or suppress events in order to satisfy the property.
Contributions

A formal framework for runtime enforcement of timed properties

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

1. Specifying Timed Properties

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

3. Implementation

4. RE of Parametric Timed Properties
   - PTAVs
   - Runtime Enforcement of PTAVs
   - Application Domains

5. Conclusions and Future Work
1. Specifying Timed Properties

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

3. Implementation

4. RE of Parametric Timed Properties
   - PTAVs
   - Runtime Enforcement of PTAVs
   - Application Domains

5. Conclusions and Future Work
Input/output stream of events

- Input/output streams of **timed events** are modelled as **timed words**.
- Given an alphabet of actions $\Sigma$,
  - Timed word: $\sigma = (t_1, a_1) \cdot (t_2, a_2) \cdots (t_n, a_n)$ **dates** are increasing.
  - $\text{tw}(\Sigma)$: set of timed words over $\Sigma$. 

\[ (2.43, \text{acq}) \cdot (5, \text{op1}) \cdot (6.4, \text{rel}) \] 
\[ (2.43, \text{acq}) \cdot (3, \text{op1}) \cdot (5.6, \text{rel}) \]
Specifying timed properties

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

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

**Examples: Properties specified by TAs**

**Regular**: any property.

“Requests and grants should alternate in this order with a delay between 15 and 20 t.u between request and grant.”
Specifying timed properties

- Enforced timed property: any regular timed language $\varphi \subseteq tw(\Sigma)$, accepted by a timed automaton $A_\varphi$.

Examples: Properties specified by TAs

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

```
Σ \ {req}  Σ \ {req}  Σ
l_0  req, x := 0  l_1  req, x < 5  l_2
Σ
req, x ≥ 5, x := 0
```

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

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

Examples: Properties specified by TAs

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

“A request, and then a grant should arrive between 10 and 15 t.u.”
Timed automata semantics

Semantics of a timed automaton

- **Timed transition system.**
- **States of the form** $q = (l, \nu)$
  (location, valuation of clocks).

- $(l_0, x = 0)$
Timed automata semantics

Semantics of a timed automaton

- Timed transition system.
- States of the form $q = (l, \nu)$ (location, valuation of clocks).

$(l_0, x = 0) \rightarrow (l_1, x = 0)$
Timed automata semantics

Semantics of a timed automaton

- Timed transition system.
- States of the form $q = (l, \nu)$ (location, valuation of clocks).

$$\Sigma \setminus \{\text{req, gr}\} \xrightarrow{\text{req, } x := 0} l_0$$

$$\Sigma \setminus \{\text{req, gr}\} \xrightarrow{\text{gr, } 15 \leq x \leq 20} l_1$$

$$\Sigma \setminus \{\text{req, gr}\} \xrightarrow{\text{gr, } x < 15 \lor x > 20} l_2$$

$$(3, \text{req}) \xrightarrow{\Sigma} (l_1, x = 0)$$

$$(18, \text{gr}) \xrightarrow{\Sigma} (l_0, x = 15)$$
Semantics of a timed automaton

- Timed transition system.
- States of the form $q = (l, \nu)$ (location, valuation of clocks).

$(l_0, x = 0) \xrightarrow{\Sigma} (l_1, x = 0) \xrightarrow{(3, \text{req})} (l_1, x = 0) \xrightarrow{(18, \text{gr})} (l_0, x = 15) \xrightarrow{(23, \text{req})} (l_0, x = 19)$
Timed automata semantics

Semantics of a timed automaton

- **Timed transition system.**
- **States of the form** $q = (l, \nu)$ (location, valuation of clocks).

\[
\begin{align*}
\Sigma \setminus \{\text{req, gr}\} &\xrightarrow{\text{req, } x := 0} l_0 \\
\Sigma \setminus \{\text{req, gr}\} &\xrightarrow{\text{gr, } 15 \leq x \leq 20} l_1 \\
\Sigma \setminus \{\text{req, gr}\} &\xrightarrow{\text{gr, } x < 15 \lor x > 20} l_2 \\
\Sigma &\xrightarrow{\sum} l_0,
\end{align*}
\]

- $(l_0, x = 0) \rightarrow (l_1, x = 0)$
- $(l_1, x = 0) \rightarrow (l_0, x = 15)$
- $(l_0, x = 0) \rightarrow (l_1, x = 19)$
- $(l_1, x = 0) \rightarrow (l_0, x = 19)$
- $(3, \text{req}) \cdot (18, \text{gr}) \cdot (23, \text{req}) \cdot (42, \text{gr})$
Specifying Timed Properties

RE of Regular Timed Properties

Requirements on an Enforcement Mechanism
Functional Definition of an Enforcement Mechanism
Operational Description of an Enforcement Mechanism

Implementation

RE of Parametric Timed Properties

PTAVs
Runtime Enforcement of PTAVs
Application Domains

Conclusions and Future Work
Enforcement mechanism behavior: example

- input: \((2, \text{req}) \cdot (3, \text{req}) \cdot (6, \text{gr})\)

- Diagram showing the enforcement mechanism with transitions and states:
  - \(l_0\) with transition: \(\text{req}, x := 0\)
  - \(l_1\) with transitions: \(\text{op}, x \geq 2, x := 0\) and \(\text{gr}, x \geq 2\)
  - \(l_2\) with transition: \(\text{req}; \Sigma \backslash \{\text{req}\}, x < 2\)

- Time axis and action axis with input events labeled.
Enforcement mechanism behavior: example

- **Input:** \((2, req) \cdot (3, req) \cdot (6, gr)\)
- **Output:** \((6, req) \cdot (8, gr)\)
Enforcement mechanism behavior

What can an enforcement mechanism do?

- CAN increase the dates of events to satisfy \( \varphi \).
- CAN suppress events if no future can satisfy \( \varphi \).
Summary of the approach

\[ \varphi \]

Enforcement Mechanism

\[ o \in tw(\Sigma) \]

\[ \sigma \in tw(\Sigma) \]
Summary of the approach

\[ \varphi \]

\[ o \in tw(\Sigma) \]

Enforcement Mechanism

\[ \sigma \in tw(\Sigma) \]
Summary of the approach

- **Requirements**: Physical, soundness and transparency constraints.
Summary of the approach

- **Requirements**: Physical, soundness and transparency constraints.
- **Functional definition**:  
  - description of the global input/output behavior,
  - satisfying requirements;
Summary of the approach

\[ \phi \]

**Requirements:** Physical, soundness and transparency constraints.

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

**Enforcement monitor:**
- refining the functional definition,
- *timed operational behavior* as a rule-based transition system,
- runtime (online) behavior of the enforcement mechanism;
Summary of the approach

- **Requirements**: Physical, soundness and transparency constraints.

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

- **Enforcement monitor**:
  - refining the functional definition,
  - timed operational behavior as a rule-based transition system,
  - runtime (online) behavior of the enforcement mechanism;

- **Implementation**: translation of the EM semantic rules into algorithms.
Subsequence: \( \sigma' \triangleleft \sigma \) if \( \sigma' \) obtained from \( \sigma \) by suppressions.

\[
(2, a) \cdot (3.5, a) \triangleleft (2, a) \cdot (3, b) \cdot (3.5, a)
\]
Preliminaries

**Subsequence:** $\sigma' \triangleleft \sigma$ if $\sigma'$ obtained from $\sigma$ by suppressions.

$(2, a) \cdot (3.5, a) \triangleleft (2, a) \cdot (3, b) \cdot (3.5, a)$
Preliminaries

**Subsequence:** $\sigma' \triangleleft \sigma$ if $\sigma'$ obtained from $\sigma$ by suppressions. 
$$(2, a) \cdot (3.5, a) \triangleleft (2, a) \cdot (3, b) \cdot (3.5, a)$$

**Delaying order** $\succsim_d$: $\sigma' \succsim_d \sigma$ if they have same untimed projections but dates in $\sigma'$ exceed corresponding dates in $\sigma$. 
$$(2.5, a) \cdot (3, b) \cdot (4.5, c) \succsim_d (2, a) \cdot (3, b) \cdot (3.5, a).$$
Preliminaries

**Subsequence:** \( \sigma' \triangleleft \sigma \) if \( \sigma' \) obtained from \( \sigma \) by suppressions.

\[(2, a) \cdot (3.5, a) \triangleleft (2, a) \cdot (3, b) \cdot (3.5, a)\]

**Delaying order** \( \succsim_d \): \( \sigma' \succsim_d \sigma \) if they have same untimed projections but dates in \( \sigma' \) exceed corresponding dates in \( \sigma \).

\[(2.5, a) \cdot (3, b) \cdot (4.5, c) \succsim_d (2, a) \cdot (3, b) \cdot (3.5, a)\]
Delaying subsequence order $\triangleleft_d$:

\[
\sigma' \triangleleft_d \sigma \overset{\text{def}}{=} \exists \sigma'' \in \text{tw}(\Sigma) : \sigma'' \triangleright \sigma \land \sigma' \succeq_d \sigma''
\]

\[(2.5, a) \cdot (3.5, b) \cdot (3.5, a) \triangleleft_d (2, a) \cdot (3, b) \cdot (3.5, a)\]

i.e., $\sigma'$ obtained from $\sigma$ by first suppressing some actions, and then increasing the dates of the actions to be kept.
Requirements on an enforcement mechanism

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

**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.
Requirements on an enforcement mechanism

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

**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 either satisfies property \( \varphi \), or is empty.
Requirements on an enforcement mechanism

**Enforcement function**

\[ E_\varphi(\sigma) \]

- **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 either satisfies property \( \varphi \), or is empty.

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

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

**Enforcement function definition:**

\[ E_\varphi : \text{tw}(\Sigma) \rightarrow \text{tw}(\Sigma) \]
Requirements on an enforcement mechanism

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

**Physical constraint:** \( \forall \sigma, \sigma' \in \text{tw}(\Sigma) : \sigma \preceq \sigma' \implies E_\varphi(\sigma) \preceq E_\varphi(\sigma') \).  
(\( \preceq \) 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 either satisfies property \( \varphi \), or is empty.

**Transparency:** \( \forall \sigma \in \text{tw}(\Sigma) : E_\varphi(\sigma) \triangleleft_d \sigma. \)  
i.e., the output is a delaying subsequence of the input.

**Additional requirements**

- Streaming behavior, deciding to output as soon as possible.
- Optimal dates.
- Optimal suppression.
Functional definition: principle

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

\[ E_\varphi(\sigma) = \Pi_1(\text{store}_\varphi(\sigma)) \]

\[ \text{store}_\varphi : \text{tw}(\Sigma) \rightarrow \text{tw}(\Sigma) \times \text{tw}(\Sigma) \]

- \( \text{store}_\varphi(\sigma) = (\sigma_s, \sigma_c) \) describes how the input stream is transformed:
  - \( \sigma_s \): computed output (to be released);
  - \( \sigma_c \): a sub-seq. of the suffix of \( \sigma \) for which output dates cannot be computed (e.g., co-safety, response).
Functional definition: principle

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

\[ E_\varphi(\sigma) = \Pi_1(\text{store}_\varphi(\sigma)) \]

\[ \text{store}_\varphi : \text{tw}(\Sigma) \to \text{tw}(\Sigma) \times \text{tw}(\Sigma) \]

- \( \text{store}_\varphi(\sigma) = (\sigma_s, \sigma_c) \) describes how the input stream is transformed:
  - \( \sigma_s \): computed output (to be released);
  - \( \sigma_c \): a sub-seq. of the suffix of \( \sigma \) for which output dates cannot be computed (e.g., co-safety, response).

- \( \text{store}_\varphi(\sigma) \) is inductively defined, has 3 cases upon reading a new event \((t, a)\).
  - \( \sigma_c \cdot (t, a) \) can be corrected.
  - \( \sigma_c \cdot (t, a) \) can never be corrected.
  - \( \sigma_c \cdot (t, a) \) can be corrected in future.
Functional definition: example

\[
\begin{align*}
\sigma &= \epsilon \\
\text{store}_{\varphi}(\sigma) &= (\sigma_s, \sigma_c) = (\epsilon, \epsilon) \\
E_{\varphi}(\sigma) &= \prod_1 (\text{store}_{\varphi}(\sigma)) = \sigma_s
\end{align*}
\]
Functional definition: example

\[\begin{align*}
\sigma &= \epsilon \\
\text{store}_\varphi(\sigma) &= (\sigma_s, \sigma_c) = (\epsilon, \epsilon) \\
E_{\varphi}(\sigma) &= \Pi_1(\text{store}_\varphi(\sigma)) = \sigma_s
\end{align*}\]

\[\begin{align*}
\sigma &= (2, \text{req}) \\
\sigma' &= \epsilon \cdot (2, \text{req}) \\
\text{store}_\varphi(\sigma) &= (\sigma_s, \sigma_c) = (\epsilon, (2, \text{req}))
\end{align*}\]

Do not output, but store!!
Functional definition: example

\[
\sigma = \epsilon \\
\text{store}_\varphi(\sigma) = (\sigma_s, \sigma_c) = (\epsilon, \epsilon) \\
E_\varphi(\sigma) = \Pi_1(\text{store}_\varphi(\sigma)) = \sigma_s
\]

\[
\sigma = (2, \text{req}) \\
\sigma_c' = \epsilon \cdot (2, \text{req}) \\
\text{store}_\varphi(\sigma) = (\sigma_s, \sigma_c) = (\epsilon, (2, \text{req})) \\
\text{Do not output, but store!!}
\]

\[
\sigma = (2, \text{req}) \cdot (3, \text{req}) \\
\sigma_c' = (2, \text{req}) \cdot (3, \text{req}) \\
\text{store}_\varphi(\sigma) = (\sigma_s, \sigma_c) = (\epsilon, (2, \text{req})) \\
\text{Suppress!!}
\]
Functional definition: example

\[\sigma = \epsilon\]
\[\text{store}_\varphi(\sigma) = (\sigma_s, \sigma_c) = (\epsilon, \epsilon)\]
\[E_\varphi(\sigma) = \prod_1 (\text{store}_\varphi(\sigma)) = \sigma_s\]

\[\sigma = (2, \text{req})\]
\[\sigma'_c = \epsilon \cdot (2, \text{req})\]
\[\text{store}_\varphi(\sigma) = (\sigma_s, \sigma_c) = (\epsilon, (2, \text{req}))\]
\[\text{Do not output, but store!!}\]

\[\sigma = (2, \text{req}) \cdot (3, \text{req})\]
\[\sigma'_c = (2, \text{req}) \cdot (3, \text{req})\]
\[\text{store}_\varphi(\sigma) = (\sigma_s, \sigma_c) = (\epsilon, (2, \text{req}))\]
\[\text{Suppress!!}\]

\[\sigma = (2, \text{req}) \cdot (3, \text{req}) \cdot (6, \text{gr})\]
\[\sigma'_c = (2, \text{req}) \cdot (6, \text{gr})\]
\[\text{store}_\varphi(\sigma) = (\sigma_s, \sigma_c) = (\epsilon \cdot (6, \text{req}) \cdot (8, \text{gr}), \epsilon)\]
\[\text{Delay, add to output!!}\]
Functional definition: formal definition

$$E_\varphi : \text{tw}(\Sigma) \rightarrow \text{tw}(\Sigma)$$

$$E_\varphi (\sigma) = \Pi_1 (\text{store}_\varphi (\sigma))$$

$$\text{store}_\varphi : \text{tw}(\Sigma) \rightarrow \text{tw}(\Sigma) \times \text{tw}(\Sigma)$$

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

if $$\text{store}_\varphi (\sigma) = (\sigma_s, \sigma_c),$$
then
t, a \in \mathbb{R}_{\geq 0} \times \Sigma$$ (new input),
$$\sigma'_c = \sigma_c \cdot (t, a)$$

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

$$\kappa_\varphi (\sigma_s, \sigma'_c) = \text{CanD}(\sigma'_c) \cap \sigma^{-1}_s \cdot \varphi,$$

$$\kappa_{\text{pref}}(\varphi)(\sigma_s, \sigma'_c) = \text{CanD}(\sigma'_c) \cap \sigma^{-1}_s \cdot \text{pref}(\varphi)$$

- $$\text{CanD}(\sigma'_c) = \{ w \in \text{tw}(\Sigma) \mid w \succeq_d \sigma'_c \land \text{start}(w) \geq \text{end}(\sigma'_c) \}$$
  i.e., sequences delaying $$\sigma'_c$$ and starting after $$t$$.
- $$\sigma^{-1}_s \cdot \varphi = \{ w \in \text{tw}(\Sigma) \mid \sigma_s \cdot w \models \varphi \}.$$ 
- $$\sigma^{-1}_s \cdot \text{pref}(\varphi) = \{ w \in \text{tw}(\Sigma) \mid \exists w' \in \text{tw}(\Sigma) : \sigma_s \cdot w \cdot w' \models \varphi \}.$$
The enforcement function satisfies the requirements

**Proposition: Enforcement function vs requirements**

The proposed definition of enforcement function satisfies the **physical**, **soundness**, **transparency** constraints.
Enforcement monitor

\[ EM(\sigma, t) = obs(E\varphi(\sigma), t) \]

\( \varphi (TA A) \)

Enforcement Monitor

- \( \sigma \)
- \( t \)
- \( EM(\sigma, t) \)
- Correspondence with \( E\varphi \):
  - Maximal prefix of \( \sigma \) observed at \( t \),
  - \( E\varphi(\text{obs}(\sigma, t)) = \text{obs}(E\varphi(\sigma), t) \cdot \sigma_{ms} \),
  - \( \sigma_{c} = \sigma_{mc} \),
  - \( q \): state reached in \([A]\) upon \( \varphi \) (corresponds to \( \sigma_{s} \)).

- Enforcement operations (cf. next slide)
- Update function: \( \text{update}(q, \sigma_{mc}, (t, a)) \) (similar to \( \text{store} \varphi \), computing online using \([A]\))
Enforcement monitor

A rule-based transition system:

EM configuration: \((\sigma_{ms}, \sigma_{mc}, t, q)\)

- \(\sigma_{ms}\): corrected sequence (yet to be released),
- \(\sigma_{mc}\): input sequence read by the EM (yet to be corrected),
- \(t\): clock indicating the current time instant,
- \(q\): current state of \([A]\).
Enforcement monitor

A rule-based transition system:

**EM configuration:** \((\sigma_{ms}, \sigma_{mc}, t, q)\)

- \(\sigma_{ms}\): corrected sequence (yet to be released),
- \(\sigma_{mc}\): input sequence read by the EM (yet to be corrected),
- \(t\): clock indicating the current time instant,
- \(q\): current state of \(\mathcal{A}\).

**Correspondence with \(E_\varphi\)**

- \(\text{obs}(\sigma, t)\): Maximal prefix of \(\sigma\) observed at \(t\),
- \(E_\varphi(\text{obs}(\sigma, t)) = \text{obs}(E_\varphi(\text{obs}(\sigma, t)), t) \cdot \sigma_{ms}\),
- \(\sigma_c = \sigma_{mc}\),
- \(q\) state reached in \(\mathcal{A}\) upon \(E_\varphi(\text{obs}(\sigma, t))\) (corresponds to \(\sigma_s\)).
Enforcement monitor

A rule-based transition system:

**EM configuration:** \((\sigma_{ms}, \sigma_{mc}, t, q)\)

- \(\sigma_{ms}\): corrected sequence (yet to be released),
- \(\sigma_{mc}\): input sequence read by the EM (yet to be corrected),
- \(t\): clock indicating the current time instant,
- \(q\): current state of \([A]\).

**Correspondence with \(E_\varphi\)**

- \(\text{obs}(\sigma, t)\): Maximal prefix of \(\sigma\) observed at \(t\),
- \(E_\varphi(\text{obs}(\sigma, t)) = \text{obs}(E_\varphi(\text{obs}(\sigma, t)), t) \cdot \sigma_{ms}\),
- \(\sigma_c = \sigma_{mc}\),
- \(q\) state reached in \([A]\) upon \(E_\varphi(\text{obs}(\sigma, t))\) (corresponds to \(\sigma_s\)).

- enforcement operations (cf. next slide)
Enforcement monitor

Formula: $\varphi (TA A)$

Enforcement Monitor

$EM(\sigma, t) = obs(E\varphi(\sigma), t)$

A rule-based transition system:

**EM configuration:** $(\sigma_{ms}, \sigma_{mc}, t, q)$

- $\sigma_{ms}$: corrected sequence (yet to be released),
- $\sigma_{mc}$: input sequence read by the EM (yet to be corrected),
- $t$: clock indicating the current time instant,
- $q$: current state of $[A]$.

**Correspondence with $E\varphi$**

- $obs(\sigma, t)$: Maximal prefix of $\sigma$ observed at $t$,
- $E\varphi(obs(\sigma, t)) = obs(E\varphi(obs(\sigma, t)), t) \cdot \sigma_{ms}$,
- $\sigma_c = \sigma_{mc}$,
- $q$ state reached in $[A]$ upon $E\varphi(obs(\sigma, t))$ (corresponds to $\sigma_s$).

- enforcement operations (cf. next slide)
- update function: $update(q, \sigma_{mc}, (t, a))$ (similar to $store_\varphi$, computing online using $[A]$)

Srinivas Pinisetty
Runtime Enforcement of Timed Properties
23 January 2015, INRIA, Rennes

25 / 41
Enforcement monitor: Operations

1. **store-ϕ**

   when update returns *ok*, \((\sigma_{ms}, \sigma_{mc}, t, q)\)
   \((t, a)/\text{store-ϕ}(t, a)/\epsilon\)
   \(\sigma_{ms} \cdot w, \epsilon, t, q'\)

2. **store-sup-ϕ**

   when update returns *bad*, \((\sigma_{ms}, \sigma_{mc}, t, q)\)
   \((t, a)/\text{store-sup-ϕ}(t, a)/\epsilon\)
   \(\sigma_{ms}, \sigma_{mc}, t, q\)

3. **store-ϕ**

   when update returns *c-bad*, \((\sigma_{ms}, \sigma_{mc}, t, q)\)
   \((t, a)/\text{store-ϕ}(t, a)/\epsilon\)
   \(\sigma_{ms}, \sigma_{mc} \cdot (t, a), t, q\)
Enforcement monitor: Operations

1. **store-\(\varphi\)**
   
   when update returns *ok*, \((\sigma_{ms}, \sigma_{mc}, t, q)\)  
   
   \[
   (t,a) / \text{store-} \varphi(t,a) / \epsilon \mapsto \epsilon (\sigma_{ms} \cdot w, \epsilon, t, q')
   \]

2. **store-sup-\(\varphi\)**
   
   when update returns *bad*, \((\sigma_{ms}, \sigma_{mc}, t, q)\)  
   
   \[
   (t,a) / \text{store-sup-} \varphi(t,a) / \epsilon \mapsto \epsilon (\sigma_{ms}, \sigma_{mc}, t, q)
   \]

3. **store-\(\overline{\varphi}\)**
   
   when update returns *c-bad*, \((\sigma_{ms}, \sigma_{mc}, t, q)\)  
   
   \[
   (t,a) / \text{store-} \overline{\varphi}(t,a) / \epsilon \mapsto \epsilon (\sigma_{ms}, \sigma_{mc} \cdot (t, a), t, q)
   \]

4. **dump**
   
   at \(t\), \(((t, a) \cdot \sigma'_{ms}, \sigma_{mc}, t, q)\)  
   
   \[
   \epsilon / \text{dump}(t,a)/(t,a) \mapsto \epsilon (\sigma'_{ms}, \sigma_{mc}, t, q)
   \]
## Enforcement monitor: Operations

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

   when update returns *ok*, \((\sigma_{ms}, \sigma_{mc}, t, q)\) \(\xrightarrow{(t,a)/\text{store-}\varphi(t,a)/\epsilon} (\sigma_{ms} \cdot w, \epsilon, t, q')\)

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

   when update returns *bad*, \((\sigma_{ms}, \sigma_{mc}, t, q)\) \(\xrightarrow{(t,a)/\text{storesup-}\overline{\varphi(t,a)}/\epsilon} (\sigma_{ms}, \sigma_{mc}, t, q)\)

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

   when update returns *c-bad*, \((\sigma_{ms}, \sigma_{mc}, t, q)\) \(\xrightarrow{(t,a)/\text{store-}\overline{\varphi(t,a)}/\epsilon} (\sigma_{ms}, \sigma_{mc} \cdot (t, a), t, q)\)

4. **dump**

   at \(t\), \(((t, a) \cdot \sigma'_{ms}, \sigma_{mc}, t, q)\) \(\xrightarrow{\epsilon/dump(t,a)/(t,a)} (\sigma'_{ms}, \sigma_{mc}, t, q)\)

5. **idle**

   when no other rule applies, \((\sigma_{ms}, \sigma_{mc}, t, q)\) \(\xrightarrow{\epsilon/\text{idle(}\delta)/\epsilon} (\sigma_{ms}, \sigma_{mc}, t + \delta, q)\)
Enforcement monitor: example

Operation: none

\[ \epsilon \leftarrow t \quad q \quad \sigma_{ms} \quad \sigma_{mc} \]

\[ \epsilon \leftarrow (2, \text{req}) \cdot (3, \text{req}) \cdot (6, gr) \]
Enforcement monitor: example

Operation: idle(2)

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

Operation: \textit{store-}\overline{\varphi}

<table>
<thead>
<tr>
<th>t</th>
<th>q</th>
<th>\sigma_{ms}</th>
<th>\sigma_{mc}</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>\langle l_0, 0 \rangle</td>
<td>\epsilon</td>
<td>\langle 2, req \rangle</td>
</tr>
</tbody>
</table>

(3, req) \cdot (6, gr)
Enforcement monitor: example

Operation: idle(1)

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>$t$</th>
<th>$q$</th>
<th>$\sigma_{ms}$</th>
<th>$\sigma_{mc}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>$(l_0, 0)$</td>
<td>$\epsilon$</td>
<td>$(2, req)$</td>
</tr>
</tbody>
</table>
Enforcement monitor: example

\[\begin{align*}
\text{req}, x &:= 0 \\
\Sigma \setminus \{\text{req}\} \quad &\text{op}, x \geq 2, \\
&x := 0 \\
\Sigma \setminus \{\text{req}\}, x < 2
\end{align*}\]

\[\begin{align*}
gr, x &\geq 2 \\
\text{req}; \\
\Sigma \setminus \{\text{req}\}, x < 2
\end{align*}\]

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

\[\begin{array}{c|c|c|c}
\epsilon & (6, gr) \\
\hline
3 & (l_0, 0) & \epsilon & (2, \text{req})
\end{array}\]
Enforcement monitor: example

Operation: \texttt{idle(3)}

\[
\begin{array}{c|c|c|c}
\text{t} & \text{q} & \sigma_{ms} & \sigma_{mc} \\
6 & \langle l_0, 0 \rangle & \epsilon & (2, \text{req}) \\
\end{array}
\]
Enforcement monitor: example

Operation: \textit{store-}\varphi

\begin{tabular}{c|c|c|c}
\epsilon & t & q & \sigma_{ms} \\
\hline
6 & (l_0, 2) & (6, req) \cdot (8, gr) & \sigma_{mc} \\
\end{tabular}
Enforcement monitor: example

Operation: dump

<table>
<thead>
<tr>
<th></th>
<th>q</th>
<th>$\sigma_{ms}$</th>
<th>$\sigma_{mc}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>$(l_0, 2)$</td>
<td>$(8, gr)$</td>
<td>$\epsilon$</td>
</tr>
</tbody>
</table>

$(6, req)$
Enforcement monitor: example

Operation: idle(2)

\[
\begin{array}{c|c|c|c}
\text{t} & \text{q} & \sigma_{ms} & \sigma_{mc} \\
8 & (l_0, 2) & (8, gr) & \epsilon \\
\end{array}
\]
Enforcement monitor: example

Operation: dump

(6, req) · (8, gr) ←

<table>
<thead>
<tr>
<th>t</th>
<th>q</th>
<th>σ_{ms}</th>
<th>σ_{mc}</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>(l_0, 2)</td>
<td>ε</td>
<td>ε</td>
</tr>
</tbody>
</table>
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 the one of the corresponding enforcement function at time $t$, i.e., $\text{obs}(E_\varphi(\sigma), t)$.

**Corollary**

Enforcement Monitors respect physical, soundness, and transparency constraints.
1. Specifying Timed Properties

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

3. Implementation

4. RE of Parametric Timed Properties
   - PTAVs
   - Runtime Enforcement of PTAVs
   - Application Domains

5. Conclusions and Future Work
Enforcement algorithms

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

await \((\sigma_{ms} \neq \epsilon)\)

\[(t, a) \leftarrow \text{dequeue}(\sigma_{ms})\]

wait \((t - d)\)

dump \((a)\)

end while
TIPEX tool

- Trace Generator: actions, max delays, increment, # of traces
- Main Test Method: events, monitoring metrics
- Store: events, execution time of update
- EME
- Property (as TA)
- GTA
- TA Generator: pattern, complexity, constants
- Class Checker: class (safety, co-safety)
- Boolean Operations: timed automata

Run-time Enforcement of Timed Properties
Srinivas Pinisetty
23 January 2015, INRIA, Rennes
“There should be a delay of at least 5 time units between any two request actions.”

![Graph showing the relationship between the length of the input trace and the total execution time of update (sec).]
1. Specifying Timed Properties

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

3. Implementation

4. RE of Parametric Timed Properties
   - PTAVs
   - Runtime Enforcement of PTAVs
   - Application Domains

5. Conclusions and Future Work
Motivations for enforcement with *time* and data
Motivations for enforcement with *time* and data

Specifying constraints over time and data

Support richer requirements in our enforcement framework (time constraints between events, and allowing events to carry data).
Motivations for enforcement with \textit{time} and data

Specifying constraints over time and data

Support richer requirements in our enforcement framework (time constraints between events, and allowing events to carry data).

- \textit{After a request, there should be a response after a delay of 5 t.u.}
Motivations for enforcement with *time* and data

**Specifying constraints over time and data**

Support richer requirements in our enforcement framework (time constraints between events, and allowing events to carry data).

- *After a request, there should be a response after a delay of 5 t.u. if there are more than 10 request messages.*
Motivations for enforcement with *time* and data

Specifying constraints over time and data

Support richer requirements in our enforcement framework (time constraints between events, and allowing events to carry data).

- After a request, there should be a response *after a delay of 5 t.u.* if there are more than 10 request messages.

- For each client, after a request, there should be a response after a delay of 5 t.u. if the number of request messages is more than 10.
Motivations for enforcement with *time* and data

Specifying constraints over time and data

Support richer requirements in our enforcement framework (time constraints between events, and allowing events to carry data).

- *After a request, there should be a response after a delay of 5 t.u. if there are more than 10 request messages.*

- *For each client, after a request, there should be a response after a delay of 5 t.u. if the number of request messages is more than 10.*

Resource allocation in a client-server model

![Resource allocation diagram](image)
Parameterized Timed Automata with Variables (PTAV)

Syntax of PTAV

\[ A(p) = (p, V, C, \Theta, L, l_0, F, X, \Sigma_p, \Delta). \]

- \( p \) - A parameter (for example to handle multiple clients/instances).
- \( C \) - External variables (to model transfer of data from the monitored system along with events).
- \( V \) - Internal variables (used for internal computation).

A PTAV is denoted as \( A(p) \), and an instance of a PTAV for a value \( \pi \) of \( p \) is denoted as \( A(\pi) \).

PTAV Example: in the context of resource allocation

\[ \Sigma_{sId} \setminus \{ R1.alloc(sId, d) \} \]

\[ R1.alloc(sId, d), \]
\[ x \geq \text{reset}, \]
\[ counter := 1, x := 0, y := 0 \]

\[ \text{delay} = \text{int}\left(\frac{\text{counter}\times sId}{6}\right) \]

\[ R1.alloc(sId, d), \]
\[ x < \text{reset} \land y \geq \text{delay}, \]
\[ counter ++, y := 0 \]

\[ l_0 \]

\[ l_1 \]

\[ \Sigma_{sId} \setminus \{ R1.alloc(sId, d) \} \]
Enforcement of **parametric** timed properties

- **Input/output timed words:** \( \sigma = (t_1, a_1(\pi_1, \eta_1)) \cdots (t_n, a_n(\pi_n, \eta_n)) \).
Enforcement of parametric timed properties

- **Input/output timed words:** \( \sigma = (t_1, a_1(\pi_1, \eta_1)) \cdots (t_n, a_n(\pi_n, \eta_n)) \).

- **Property \( \varphi \)** specified by PTAV.
Enforcement of **parametric** timed properties

- **Input/output timed words:** $\sigma = (t_1, a_1(\pi_1, \eta_1)) \cdots (t_n, a_n(\pi_n, \eta_n))$.
- Property $\varphi$ specified by PTAV.
- An instance of EM per parameter value (takes as input only the events with same parameter value).
Enforcement of **parametric** timed properties

- **Input/output timed words**: $\sigma = (t_1, a_1(\pi_1, \eta_1)) \cdot \ldots \cdot (t_n, a_n(\pi_n, \eta_n))$.
- Property $\varphi$ specified by PTAV.
- An instance of EM per parameter value (takes as input only the events with same parameter value).
- Slicing input.
  - $\sigma = (0.5, a(1, \eta_1)) \cdot (0.8, a(2, \eta_2)) \cdot (1.0, a(1, \eta_3)) \cdot (1.4, a(2, \eta_4))$
  - $\sigma_{\downarrow 1} = (0.5, a(1, \eta_1)) \cdot (1.0, a(1, \eta_3))$
Enforcement of **parametric** timed properties

- Input/output timed words: \( \sigma = (t_1, a_1(\pi_1, \eta_1)) \cdots (t_n, a_n(\pi_n, \eta_n)) \).
- Property \( \varphi \) specified by PTAV.
- An instance of EM per parameter value (takes as input only the events with same parameter value).
- Slicing input.
  - \( \sigma = (0.5, a(1, \eta_1)) \cdot (0.8, a(2, \eta_2)) \cdot (1.0, a(1, \eta_3)) \cdot (1.4, a(2, \eta_4)) \)
  - \( \sigma \downarrow_1 = (0.5, a(1, \eta_1)) \cdot (1.0, a(1, \eta_3)) \)
- Update function is computable (straightforward adaptation).
Input/output timed words: $\sigma = (t_1, a_1(\pi_1, \eta_1)) \cdot \cdots \cdot (t_n, a_n(\pi_n, \eta_n))$.

Property $\varphi$ specified by PTAV.

An instance of EM per parameter value (takes as input only the events with same parameter value).

Slicing input.

- $\sigma = (0.5, a(1, \eta_1)) \cdot (0.8, a(2, \eta_2)) \cdot (1.0, a(1, \eta_3)) \cdot (1.4, a(2, \eta_4))$
- $\sigma \downarrow_1 = (0.5, a(1, \eta_1)) \cdot (1.0, a(1, \eta_3))$

Update function is computable (straightforward adaptation).

Output of each EM instance satisfies constraints.
Other application domains

Protecting mail servers

If the number of RCPT_TO messages from a client is greater than maxreq, then there should be a delay of at least \( \text{del \, t.u.} \) before responding an \( \text{OK\_250} \).
1 Specifying Timed Properties

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

3 Implementation

4 RE of Parametric Timed Properties
   - PTAVs
   - Runtime Enforcement of PTAVs
   - Application Domains

5 Conclusions and Future Work
Formally based runtime enforcement mechanisms for requirements with real-time constraints.

For all regular timed properties modeled as a timed automaton.

Enforcer can delay and suppress events.

Enforcement mechanisms described at several levels of abstraction (enforcement function, enforcement monitor and algorithms).

Prototype tool implementation (TIPEX).

Requirements with constraints both on time and data (PTAV).
Summary: Evolution of results

- Runtime Enforcement of Timed Properties [RV 2012].
- Runtime Enforcement of Regular Timed Properties [SAC-SVT 2014].
- Runtime Enforcement of Timed Properties Revisited [FMSD Journal].
- Runtime Enforcement of Regular Timed Properties by Suppressing and Delaying Events [SCP Journal].
- Runtime Enforcement of Parametric Timed Properties with Practical Applications [WODES 2014].
Summary: Evolution of results

- Runtime Enforcement of Timed Properties [RV 2012].
- Runtime Enforcement of Regular Timed Properties [SAC-SVT 2014].
- Runtime Enforcement of Timed Properties Revisited [FMSD Journal].
- Runtime Enforcement of Regular Timed Properties by Suppressing and Delaying Events [SCP Journal].
- Runtime Enforcement of Parametric Timed Properties with Practical Applications [WODES 2014].
Summary: Evolution of results

- Runtime Enforcement of Timed Properties [RV 2012].
- Runtime Enforcement of Regular Timed Properties [SAC-SVT 2014].
- Runtime Enforcement of Timed Properties Revisited [FMSD Journal].
- Runtime Enforcement of Regular Timed Properties by Suppressing and Delaying Events [SCP Journal].
- Runtime Enforcement of Parametric Timed Properties with Practical Applications [WODES 2014].
Summary: Evolution of results

- Runtime Enforcement of Timed Properties [RV 2012].
- Runtime Enforcement of Regular Timed Properties [SAC-SVT 2014].
- Runtime Enforcement of Timed Properties Revisited [FMSD Journal].
- Runtime Enforcement of Regular Timed Properties by Suppressing and Delaying Events [SCP Journal].
- Runtime Enforcement of Parametric Timed Properties with Practical Applications [WODES 2014].
Summary: Evolution of results

- Runtime Enforcement of Timed Properties [RV 2012].
- Runtime Enforcement of Regular Timed Properties [SAC-SVT 2014].
- Runtime Enforcement of Timed Properties Revisited [FMSD Journal].
- Runtime Enforcement of Regular Timed Properties by Suppressing and Delaying Events [SCP Journal].
- Runtime Enforcement of Parametric Timed Properties with Practical Applications [WODES 2014].
Theoretical extensions

- Predictive enforcement.
  - $E_{\psi/\varphi} : \psi \subseteq tw(\Sigma) \rightarrow \{\epsilon\} \cup \varphi \subseteq tw(\Sigma)$.
  - Can we do better, knowing possible future?
Perspectives

Theoretical extensions

- Predictive enforcement.
  - $E_{\varphi/\psi} : \psi \subseteq tw(\Sigma) \rightarrow \{\epsilon\} \cup \varphi \subseteq tw(\Sigma)$.
  - Can we do better, knowing possible future?

- Combining enforcement monitors (in series) enforcing multiple properties.
Perspectives

Theoretical extensions

- Predictive enforcement.
  - $E_{\varphi/\psi} : \psi \subseteq \text{tw}(\Sigma) \rightarrow \{\epsilon\} \cup \varphi \subseteq \text{tw}(\Sigma)$.
  - Can we do better, knowing possible future?

- Combining enforcement monitors (in series) enforcing multiple properties.

- Enforcement with partial control.
Theoretical extensions

- Predictive enforcement.
  - $E_{\varphi/\psi} : \psi \subseteq \text{tw}(\Sigma) \rightarrow \{\epsilon\} \cup \varphi \subseteq \text{tw}(\Sigma)$.
  - Can we do better, knowing possible future?

- Combining enforcement monitors (in series) enforcing multiple properties.
- Enforcement with partial control.
- What are enforceable properties?
Perspectives

Theoretical extensions

- Predictive enforcement.
  - \( E_{\varphi/\psi} : \psi \subseteq tw(\Sigma) \rightarrow \{\varepsilon\} \cup \varphi \subseteq tw(\Sigma) \).
  - Can we do better, knowing possible future?

- Combining enforcement monitors (in series) enforcing multiple properties.
- Enforcement with partial control.
- What are enforceable properties?

Applications

- Implementing efficient enforcement monitors (in application scenarios).
Theoretical extensions

- Predictive enforcement.
  \[ E_{\varphi/\psi} : \psi \subseteq \text{tw}(\Sigma) \rightarrow \{\epsilon\} \cup \varphi \subseteq \text{tw}(\Sigma). \]
- Can we do better, knowing possible future?

- Combining enforcement monitors (in series) enforcing multiple properties.
- Enforcement with partial control.
- What are *enforceable* properties?

Applications

- Implementing efficient enforcement monitors (in application scenarios).
- Enforcement monitoring techniques to guarantee behavior of off-the-shelf components.
- Enforcement monitor synthesis mechanism to realize some requirements automatically.