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)

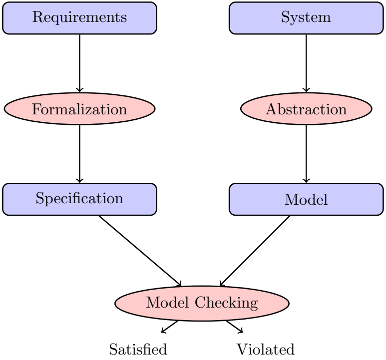
Traditional design and verification

- 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:

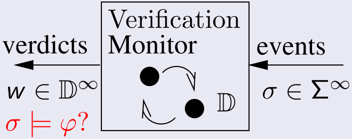
- Monitor execution of a system (e.g., trace, log, messages).
- No system model.
- A formal requirement: Property φ.

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 φ.

Runtime verification



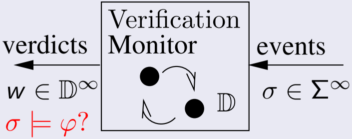
- Input: stream of events.
- Output: stream of **verdicts**.
- Does the run satisfy the property?

Runtime verification and enforcement (monitors)

Runtime verification and enforcement:

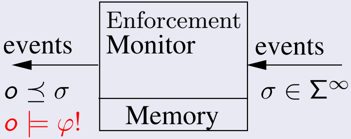
- Monitor execution of a system (e.g., trace, log, messages).
- No system model.
- A formal requirement: Property φ.

Runtime verification



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

Runtime enforcement



- 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

- Enforceable security policies – F. B. Schneider et al-2000.
- Runtime enforcement of non-safety policies – J. Ligatti et al-2009.
- Enforcement Monitoring wrt. the Safety-Progress Classification of Properties – Y. Falcone et al-2010.

Related work on monitoring

Runtime enforcement of untimed properties

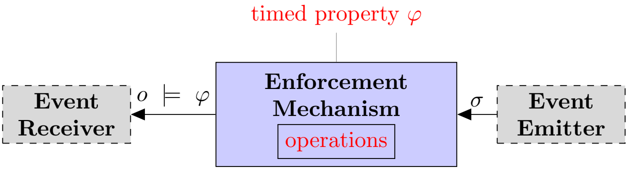
- Enforceable security policies – F. B. Schneider et al-2000.
- Runtime enforcement of non-safety policies – J. Ligatti et al-2009.
- Enforcement Monitoring wrt. the Safety-Progress Classification of Properties – Y. Falcone et al-2010.

Runtime verification of timed properties

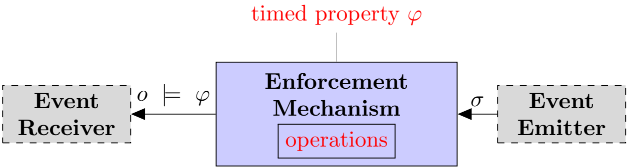
Efforts mainly to *verify* timed properties at runtime:

- Runtime verification of TLTL – A. Bauer et al-2011.
- Algorithms for monitoring real-time properties – D. Basin et al-2011.
- The Analog Monitoring Tool.(monitoring specifications over continuous signals) – D. Nickovic et al-2010.
- Safe runtime verification of real-time properties – C. Colombo et al.

Context and objectives



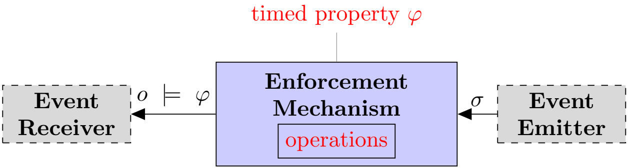
Context and objectives



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 σ in to output o such that $o \models \varphi$.

Context and objectives



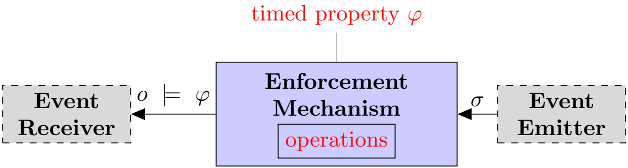
Objective

- 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 σ in to output o such that $o \models \varphi$.

Three directions

- Expressiveness of the supported specification formalism.

Context and objectives



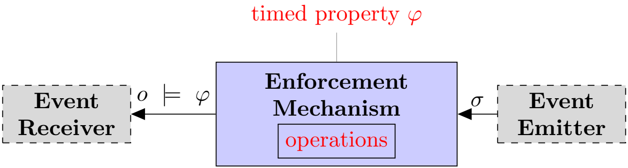
Objective

- 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 σ in to output o such that $o \models \varphi$.

Three directions

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

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 σ 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.

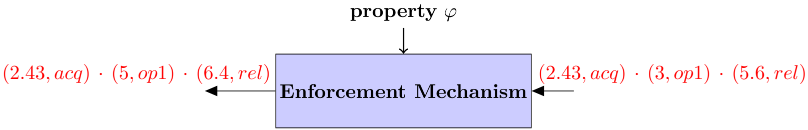
Contributions

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.

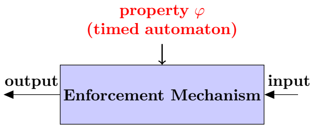
Outline

Input/output stream of events



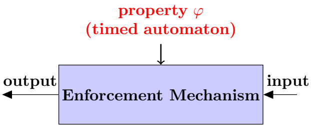
- Input/output streams of timed events are modelled as timed words.
- Given an alphabet of actions Σ,
  - Timed word: $\sigma = (t_1, a_1) \cdot (t_2, a_2) \cdots (t_n, a_n)$ dates are increasing.
  - tw(Σ): set of timed words over Σ.

Specifying timed properties



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

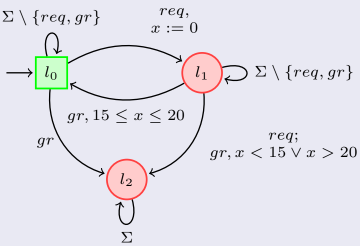
Specifying timed properties



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

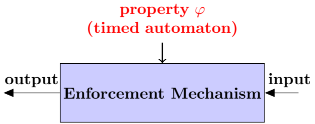
Examples: Properties specified by TAs

Regular: any property.



"Requests and grants should alternate in this order with a delay between 15 and 20 t.u between request and grant."

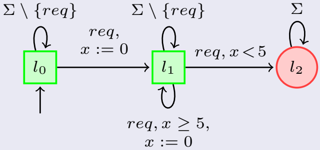
Specifying timed properties



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

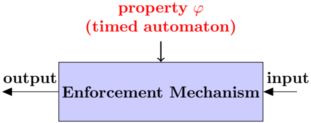
Examples: Properties specified by TAs

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



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

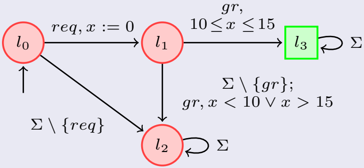
Specifying timed properties



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

Examples: Properties specified by TAs

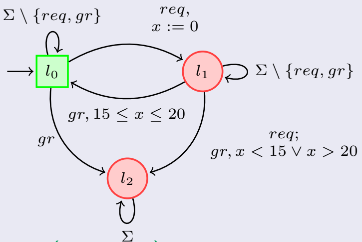
Co-safety: something good will eventually happen within a finite amount of time (extension closed).



"A request, and then a grant should arrive between 10 and 15 t.u."

Timed automata semantics

Semantics of a timed automaton

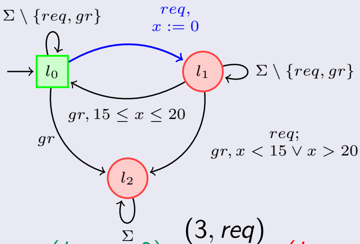


- $(l_0, x = 0)$

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

Timed automata semantics

Semantics of a timed automaton

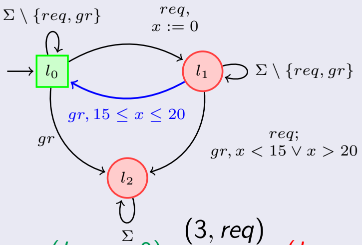


- Timed transition system.
- States of the form $q = (l, \nu)$ (location, valuation of clocks).

- $(l_0, x = 0) \overset{(3, req)}{\rightarrow} (l_1, x = 0)$

Timed automata semantics

Semantics of a timed automaton

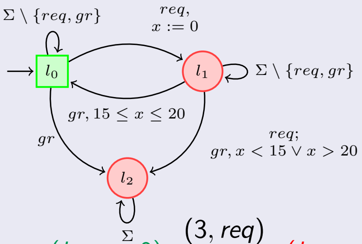


- Timed transition system.
- States of the form $q = (l, \nu)$ (location, valuation of clocks).

- $(l_0, x = 0) \xrightarrow{(3, req)} (l_1, x = 0) \xrightarrow{(18, gr)} (l_0, x = 15)$

Timed automata semantics

Semantics of a timed automaton

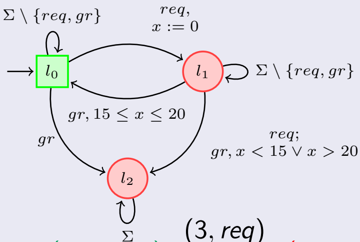


- Timed transition system.
- States of the form $q = (l, \nu)$ (location, valuation of clocks).

- $(l_0, x = 0) \overset{(3, req)}{\rightarrow} (l_1, x = 0) \overset{(18, gr)}{\rightarrow} (l_0, x = 15) \overset{(23, req)}{\rightarrow} (l_1, x = 0) \overset{(42, gr)}{\rightarrow} (l_0, x = 19)$

Timed automata semantics

Semantics of a timed automaton

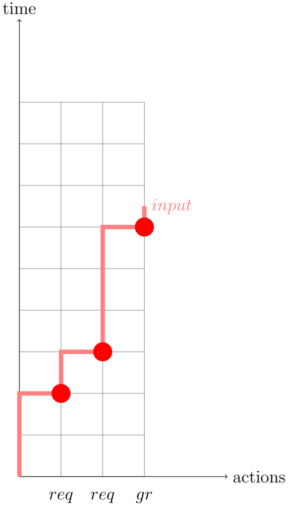


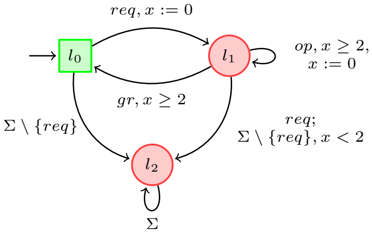
- Timed transition system.
- States of the form $q = (l, \nu)$ (location, valuation of clocks).

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

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

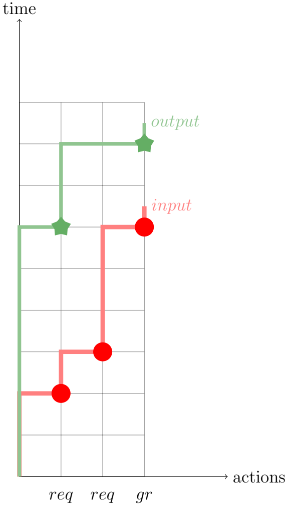
Enforcement mechanism behavior: example

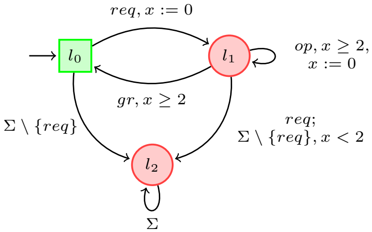




- input: $(2, req) \cdot (3, req) \cdot (6, gr)$

Enforcement mechanism behavior: example

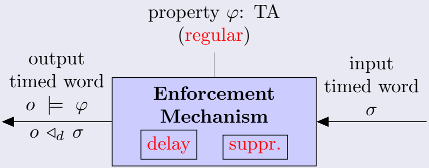




- input: $(2, req) \cdot (3, req) \cdot (6, gr)$
- output: $(6, req) \cdot (8, gr)$

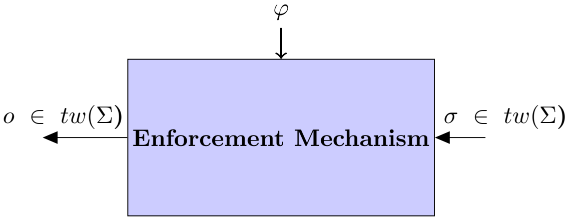
Enforcement mechanism behavior

What can an enforcement mechanism do?

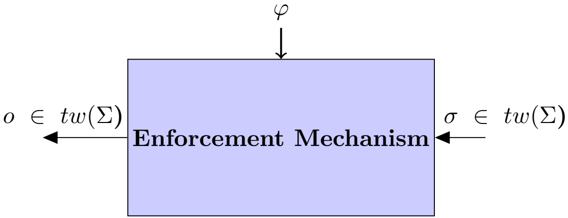


- CAN increase the dates of events to satisfy φ.
- CAN suppress events if no future can satisfy φ.

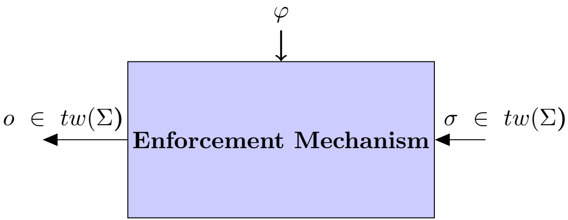
Summary of the approach



Summary of the approach

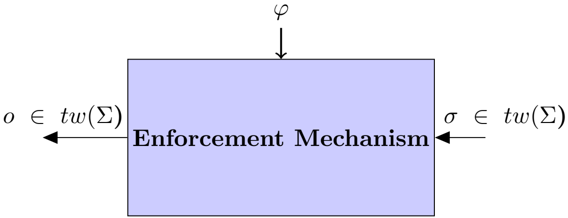


Summary of the approach



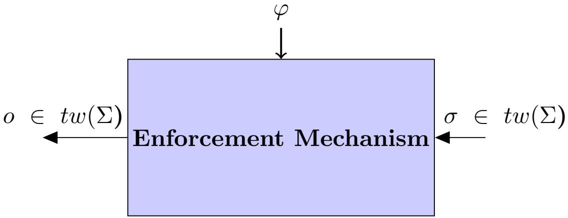
- **Requirements**: Physical, soundness and transparency constraints.

Summary of the approach



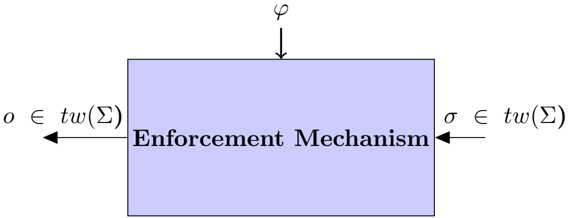
- **Requirements**: Physical, soundness and transparency constraints.
- **Functional definition**:
  - description of the global input/output behavior,
  - satisfying requirements;

Summary of the approach



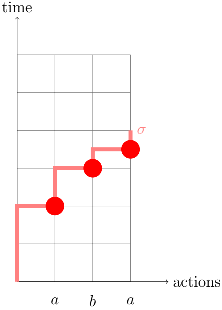
- **Requirements**: Physical, soundness and transparency constraints.
- **Functional definition**:
  - description of the global input/output behavior,
  - satisfying requirements;
- **Enforcement monitor**:
  - refining the functional definition,
  - timed operational behavior as a rule-based transition system,
  - runtime (online) behavior of the enforcement mechanism;

Summary of the approach



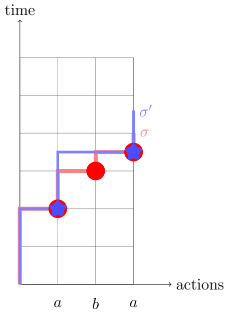
- **Requirements**: Physical, soundness and transparency constraints.
- **Functional definition**:
  - description of the global input/output behavior,
  - satisfying requirements;
- **Enforcement monitor**:
  - refining the functional definition,
  - timed operational behavior as a rule-based transition system,
  - runtime (online) behavior of the enforcement mechanism;
- **Implementation**: translation of the EM semantic rules into algorithms.

Preliminaries



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

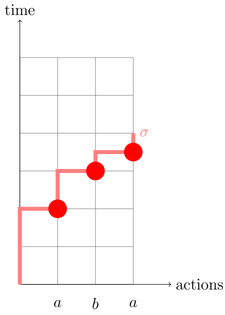
Preliminaries



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

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

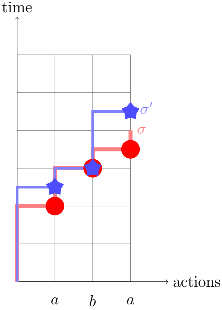
Preliminaries



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

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

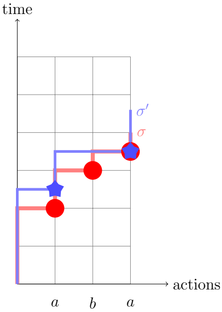
Preliminaries



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

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

Preliminaries



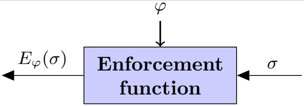
Delaying subsequence order $\triangleleft_d$:

$\sigma' \triangleleft_d \sigma \stackrel{\text{def}}{=} \exists \sigma'' \in \text{tw}(\Sigma) : \sigma'' \triangleleft \sigma \land \sigma' \succcurlyeq_d \sigma''$

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

i.e., σ' obtained from σ by first suppressing some actions, and then increasing the dates of the actions to be kept.

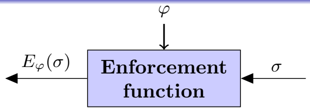
Requirements on an enforcement mechanism



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

Physical constraint: $\forall \sigma, \sigma' \in \text{tw}(\Sigma) : \sigma \preccurlyeq \sigma' \implies E_\varphi(\sigma) \preccurlyeq E_\varphi(\sigma')$.
(where $\preccurlyeq$ is the prefix ordering)
i.e., the output cannot be undone.

Requirements on an enforcement mechanism

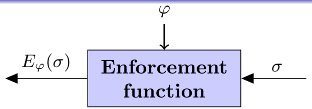


$E_\varphi : \text{tw}(\Sigma) \rightarrow \text{tw}(\Sigma)$

Physical constraint: $\forall \sigma, \sigma' \in \text{tw}(\Sigma) : \sigma \preccurlyeq \sigma' \implies E_\varphi(\sigma) \preccurlyeq E_\varphi(\sigma')$.
(where $\preccurlyeq$ is the prefix ordering)
i.e., the output cannot be undone.

Soundness: $\forall \sigma \in \text{tw}(\Sigma) : E_\varphi(\sigma) \models \varphi \vee E_\varphi(\sigma) = \epsilon$.
i.e., the output either satisfies property φ, or is empty.

Requirements on an enforcement mechanism



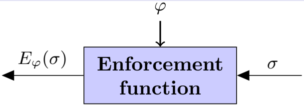
$E_\varphi : \text{tw}(\Sigma) \rightarrow \text{tw}(\Sigma)$

Physical constraint: $\forall \sigma, \sigma' \in \text{tw}(\Sigma) : \sigma \preccurlyeq \sigma' \implies E_\varphi(\sigma) \preccurlyeq E_\varphi(\sigma')$.
(where $\preccurlyeq$ is the prefix ordering)
i.e., the output cannot be undone.

Soundness: $\forall \sigma \in \text{tw}(\Sigma) : E_\varphi(\sigma) \models \varphi \vee E_\varphi(\sigma) = \epsilon$.
i.e., the output either satisfies property φ, 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.

Requirements on an enforcement mechanism



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

Physical constraint: $\forall \sigma, \sigma' \in \mathrm{tw}(\Sigma) : \sigma \preccurlyeq \sigma' \implies E_\varphi(\sigma) \preccurlyeq E_\varphi(\sigma')$.
(where $\preccurlyeq$ is the prefix ordering)
i.e., the output cannot be undone.

Soundness: $\forall \sigma \in \mathrm{tw}(\Sigma) : E_\varphi(\sigma) \models \varphi \vee E_\varphi(\sigma) = \epsilon$.
i.e., the output either satisfies property φ, or is empty.

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

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

$$E_\varphi(\sigma) = \Pi_1\left(\mathrm{store}_\varphi\left(\sigma\right)\right)$$

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

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

Functional definition: principle

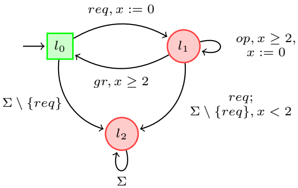
$E_\varphi : \mathrm{tw}(\Sigma) \to \mathrm{tw}(\Sigma)$

$$E_\varphi(\sigma) = \Pi_1 \left(\mathrm{store}_\varphi\left(\sigma\right)\right)$$

$\mathrm{store}_\varphi : \mathrm{tw}(\Sigma) \to \mathrm{tw}(\Sigma) \times \mathrm{tw}(\Sigma)$

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

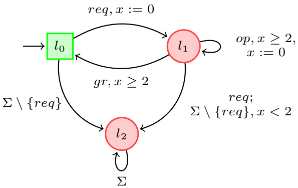
Functional definition: example



σ

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

Functional definition: example



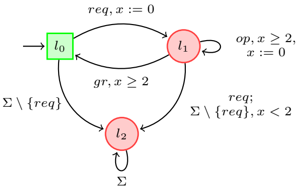
σ

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

$(2, req)$

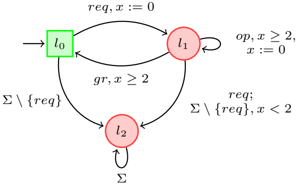
$\sigma = (2, req)$
$\sigma'_c = \epsilon \cdot (2, req)$
$\text{store}_\varphi(\sigma) = (\sigma_s, \sigma_c) = (\epsilon, (2, req))$
Do not output, but store!!

Functional definition: example



σ	$\sigma = \epsilon$ $\text{store}_\varphi(\sigma) = (\sigma_s, \sigma_c) = (\epsilon, \epsilon)$ $E_\varphi(\sigma) = \Pi_1(\text{store}_\varphi(\sigma)) = \sigma_s$
$(2, req)$	$\sigma = (2, req)$ $\sigma'_c = \epsilon \cdot (2, req)$ $\text{store}_\varphi(\sigma) = (\sigma_s, \sigma_c) = (\epsilon, (2, req))$ Do not output, but store!!
$(3, req)$	$\sigma = (2, req) \cdot (3, req)$ $\sigma'_c = (2, req) \cdot (3, req)$ $\text{store}_\varphi(\sigma) = (\sigma_s, \sigma_c) = (\epsilon, (2, req))$ Suppress!!

Functional definition: example



σ	$\sigma = \epsilon$ $\text{store}_\varphi(\sigma) = (\sigma_s, \sigma_c) = (\epsilon, \epsilon)$ $E_\varphi(\sigma) = \Pi_1(\text{store}_\varphi(\sigma)) = \sigma_s$
$(2, req)$	$\sigma = (2, req)$ $\sigma'_c = \epsilon \cdot (2, req)$ $\text{store}_\varphi(\sigma) = (\sigma_s, \sigma_c) = (\epsilon, (2, req))$ Do not output, but store!!
$(3, req)$	$\sigma = (2, req) \cdot (3, req)$ $\sigma'_c = (2, req) \cdot (3, req)$ $\text{store}_\varphi(\sigma) = (\sigma_s, \sigma_c) = (\epsilon, (2, req))$ Suppress!!
$(6, gr)$	$\sigma = (2, req) \cdot (3, req) \cdot (6, gr)$ $\sigma'_c = (2, req) \cdot (6, gr)$ $\text{store}_\varphi(\sigma) = (\sigma_s, \sigma_c) = (\epsilon \cdot (6, req) \cdot (8, gr), \epsilon)$ Delay, add to output!!

Functional definition: formal definition

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

$$E_\varphi(\sigma) = \Pi_1\left(\mathrm{store}_\varphi(\sigma)\right)$$

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

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

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

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

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

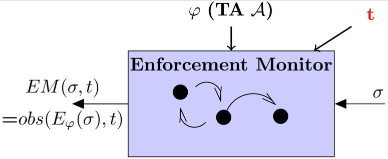
- $\mathrm{CanD}(\sigma'_c) = \{w \in \mathrm{tw}(\Sigma) \mid w \succcurlyeq_d \sigma'_c \wedge \mathrm{start}(w) \geq \mathrm{end}(\sigma'_c)\}$
  i.e., sequences delaying σ'_c and starting after t.
- $\sigma_s^{-1} \cdot \varphi = \{w \in \mathrm{tw}(\Sigma) \mid \sigma_s \cdot w \models \varphi\}$.
- $\sigma_s^{-1} \cdot \mathrm{pref}(\varphi) = \{w \in \mathrm{tw}(\Sigma) \mid \exists w' \in \mathrm{tw}(\Sigma) : \sigma_s \cdot w \cdot w' \models \varphi\}$.

The enforcement function satisfies the requirements

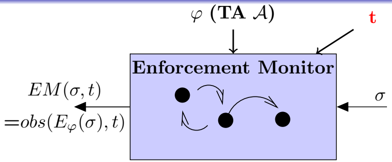
Proposition: Enforcement function vs requirements

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

Enforcement monitor



Enforcement monitor

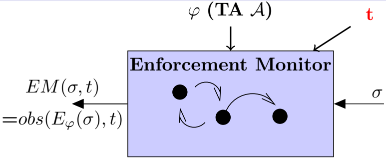


A *rule-based transition system*:

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

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

Enforcement monitor



A *rule-based transition system*:

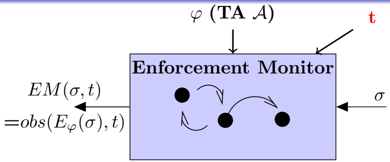
EM configuration: $(\sigma_{ms}, \sigma_{mc}, t, q)$

- σ_{ms}: corrected sequence (yet to be released),
- σ_{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_φ

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

Enforcement monitor



A *rule-based transition system*:

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

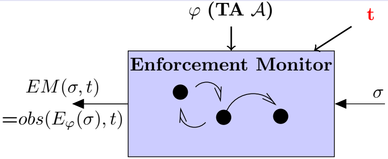
- σ_{ms}: corrected sequence (yet to be released),
- σ_{mc}: input sequence read by the EM (yet to be corrected),
- t: clock indicating the current time instant,
- q: current state of $[\![\mathcal{A}]\!]$.

Correspondence with E_φ

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

- enforcement operations (cf. next slide)

Enforcement monitor



A *rule-based transition system*:

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

- σ_{ms}: corrected sequence (yet to be released),
- σ_{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_φ

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

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

Enforcement monitor: Operations

1. *store*-φ

when update returns ok, $(\sigma_{\mathrm{ms}}, \sigma_{\mathrm{mc}}, t, q) \overset{(t,a)/\mathrm{store}-\varphi(t,a)/\epsilon}{\hookrightarrow_{\mathcal{E}}} (\sigma_{\mathrm{ms}} \cdot w, \epsilon, t, q')$

2. *store*-sup-$\overline{\varphi}$

when update returns bad, $(\sigma_{\mathrm{ms}}, \sigma_{\mathrm{mc}}, t, q) \overset{(t,a)/\mathrm{store}_{\mathrm{sup}}-\overline{\varphi}(t,a)/\epsilon}{\hookrightarrow_{\mathcal{E}}} (\sigma_{\mathrm{ms}}, \sigma_{\mathrm{mc}}, t, q)$

3. *store*-$\overline{\varphi}$

when update returns c-bad, $(\sigma_{\mathrm{ms}}, \sigma_{\mathrm{mc}}, t, q) \overset{(t,a)/\mathrm{store}-\overline{\varphi}(t,a)/\epsilon}{\hookrightarrow_{\mathcal{E}}} (\sigma_{\mathrm{ms}}, \sigma_{\mathrm{mc}} \cdot (t, a), t, q)$

Enforcement monitor: Operations

1. *store*-φ

when update returns ok, $(\sigma_{\mathrm{ms}}, \sigma_{\mathrm{mc}}, t, q) \overset{(t,a)/\mathrm{store}-\varphi(t,a)/\epsilon}{\hookrightarrow_{\mathcal{E}}} (\sigma_{\mathrm{ms}} \cdot w, \epsilon, t, q')$

2. *store*-sup-$\overline{\varphi}$

when update returns bad, $(\sigma_{\mathrm{ms}}, \sigma_{\mathrm{mc}}, t, q) \overset{(t,a)/\mathrm{store}_{\mathrm{sup}}-\overline{\varphi}(t,a)/\epsilon}{\hookrightarrow_{\mathcal{E}}} (\sigma_{\mathrm{ms}}, \sigma_{\mathrm{mc}}, t, q)$

3. *store*-$\overline{\varphi}$

when update returns c-bad, $(\sigma_{\mathrm{ms}}, \sigma_{\mathrm{mc}}, t, q) \overset{(t,a)/\mathrm{store}-\overline{\varphi}(t,a)/\epsilon}{\hookrightarrow_{\mathcal{E}}} (\sigma_{\mathrm{ms}}, \sigma_{\mathrm{mc}} \cdot (t, a), t, q)$

4. *dump*

at t, $((t, a) \cdot \sigma'_{\mathrm{ms}}, \sigma_{\mathrm{mc}}, t, q) \overset{\epsilon/\mathrm{dump}(t,a)/(t,a)}{\hookrightarrow_{\mathcal{E}}} (\sigma'_{\mathrm{ms}}, \sigma_{\mathrm{mc}}, t, q)$

Enforcement monitor: Operations

1. *store*-φ

when update returns ok, $(\sigma_{\mathrm{ms}}, \sigma_{\mathrm{mc}}, t, q) \overset{(t,a)/\mathrm{store}-\varphi(t,a)/\epsilon}{\hookrightarrow_{\mathcal{E}}} (\sigma_{\mathrm{ms}} \cdot w, \epsilon, t, q')$

2. *store*-sup-$\overline{\varphi}$

when update returns bad, $(\sigma_{\mathrm{ms}}, \sigma_{\mathrm{mc}}, t, q) \overset{(t,a)/\mathrm{store}_{\mathrm{sup}}-\overline{\varphi}(t,a)/\epsilon}{\hookrightarrow_{\mathcal{E}}} (\sigma_{\mathrm{ms}}, \sigma_{\mathrm{mc}}, t, q)$

3. *store*-$\overline{\varphi}$

when update returns c-bad, $(\sigma_{\mathrm{ms}}, \sigma_{\mathrm{mc}}, t, q) \overset{(t,a)/\mathrm{store}-\overline{\varphi}(t,a)/\epsilon}{\hookrightarrow_{\mathcal{E}}} (\sigma_{\mathrm{ms}}, \sigma_{\mathrm{mc}} \cdot (t, a), t, q)$

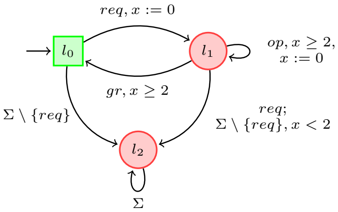
4. *dump*

at t, $((t, a) \cdot \sigma'_{\mathrm{ms}}, \sigma_{\mathrm{mc}}, t, q) \overset{\epsilon/\mathrm{dump}(t,a)/(t,a)}{\hookrightarrow_{\mathcal{E}}} (\sigma'_{\mathrm{ms}}, \sigma_{\mathrm{mc}}, t, q)$

5. *idle*

when no other rule applies, $(\sigma_{\mathrm{ms}}, \sigma_{\mathrm{mc}}, t, q) \overset{\epsilon/\mathrm{idle}(\delta)/\epsilon}{\hookrightarrow_{\mathcal{E}}} (\sigma_{\mathrm{ms}}, \sigma_{\mathrm{mc}}, t + \delta, q)$

Enforcement monitor: example

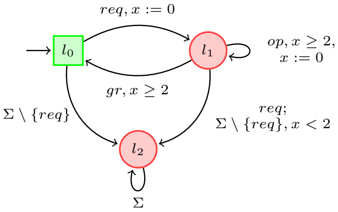


Operation: none

$\epsilon \longleftarrow$ $\longleftarrow (2, req) \cdot (3, req) \cdot (6, gr)$

t	q	σ_{ms}	σ_{mc}
0	$(l_0, 0)$	ϵ	ϵ

Enforcement monitor: example

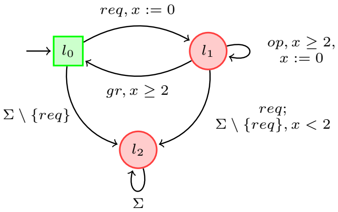


Operation: idle(2)

ϵ $\longleftarrow$ $\longleftarrow (2, req) \cdot (3, req) \cdot (6, gr)$

t	q	σ_{ms}	σ_{mc}
2	$(l_0, 0)$	ϵ	ϵ

Enforcement monitor: example



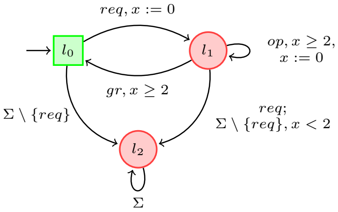
Operation: *store-$\overline{\varphi}$*

$\epsilon \quad \longleftarrow$

t	q	σ_{ms}	σ_{mc}
2	$(l_0, 0)$	ϵ	$(2, req)$

$\longleftarrow \quad (3, req) \cdot (6, gr)$

Enforcement monitor: example



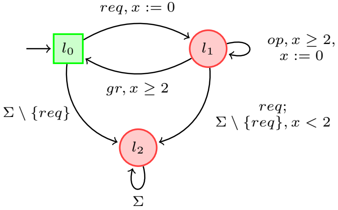
Operation: idle(1)

$\epsilon \quad \longleftarrow$

t	q	σ_{ms}	σ_{mc}
3	$(l_0, 0)$	ϵ	$(2, req)$

$\longleftarrow \quad (3, req) \cdot (6, gr)$

Enforcement monitor: example



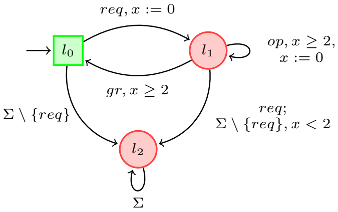
Operation: *store-sup-*$\overline{\varphi}$

$\epsilon \longleftarrow$

t	q	σ_{ms}	σ_{mc}
3	$(l_0, 0)$	ϵ	$(2, req)$

$\longleftarrow (6, gr)$

Enforcement monitor: example

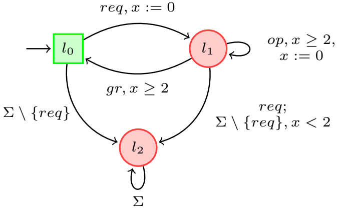


Operation: idle(3)

ϵ $\longleftarrow$ $\longleftarrow$ $(6, gr)$

t	q	σ_{ms}	σ_{mc}
6	$(l_0, 0)$	ϵ	$(2, req)$

Enforcement monitor: example

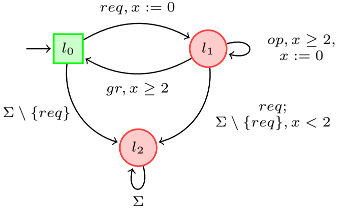


Operation: *store-φ*

ϵ $\longleftarrow$ $\longleftarrow$ ϵ

t	q	σ_{ms}	σ_{mc}
6	$(l_0, 2)$	$(6, req) \cdot (8, gr)$	ϵ

Enforcement monitor: example



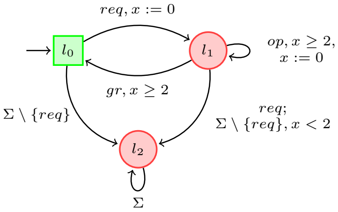
Operation: *dump*

$(6, req)$ $\longleftarrow$

t	q	σ_{ms}	σ_{mc}
6	$(l_0, 2)$	$(8, gr)$	ϵ

$\longleftarrow$ ϵ

Enforcement monitor: example



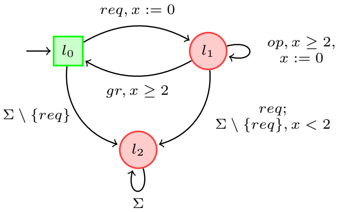
Operation: idle(2)

$(6, req)$ $\longleftarrow$

t	q	σ_{ms}	σ_{mc}
8	$(l_0, 2)$	$(8, gr)$	ϵ

$\longleftarrow$ ϵ

Enforcement monitor: example



Operation: *dump*

$(6, req) \cdot (8, gr) \longleftarrow$ $\longleftarrow$ ϵ

t	q	σ_{ms}	σ_{mc}
8	$(l_0, 2)$	ϵ	ϵ

Enforcement monitor: correctness

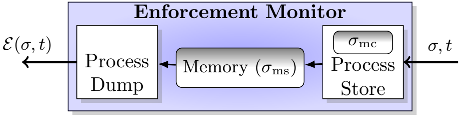
Implementation relation between Enforcement Monitor and Enforcement Function

Given some property φ, 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., $\mathrm{obs}(E_\varphi(\sigma), t)$.

Corollary

Enforcement Monitors respect physical, soundness, and transparency constraints.

Enforcement algorithms



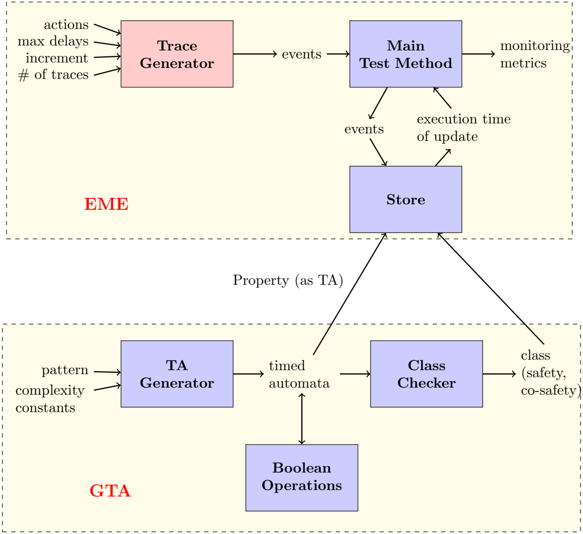
Algorithm: StoreProcess

$(t, q) \leftarrow (0, q_0)$
$(\sigma_{ms}, \sigma_{mc}) \leftarrow (\epsilon, \epsilon)$
while `tt` **do**
  $(t, a) \leftarrow$ await $(event)$
  $(q', \sigma'_{mc}, isPath) \leftarrow \text{update}(q, \sigma_{mc}, (t, a))$
  if $isPath =$ `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

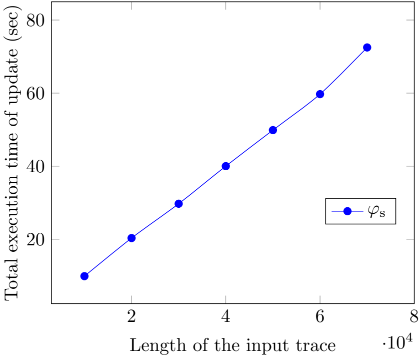
$d \leftarrow 0$
while `tt` **do**
  await $(\sigma_{ms} \neq \epsilon)$
  $(t, a) \leftarrow$ dequeue (σ_{ms})
  wait $(t - d)$
  dump (a)
end while

TIPEX tool


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

Evaluation

"There should be a delay of at least 5 time units between any two request actions".



φ_s	
$\lvert tr \rvert$	$t_$update
10,000	9.895
20,000	20.323
30,000	29.722
40,000	40.007
50,000	49.869
60,000	59.713
70,000	72.494

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 *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.*

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.*

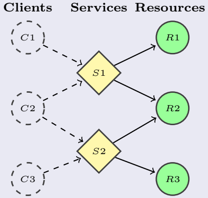
Motivations for enforcement with *time* and data

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



Parameterized Timed Automata with Variables (PTAV)

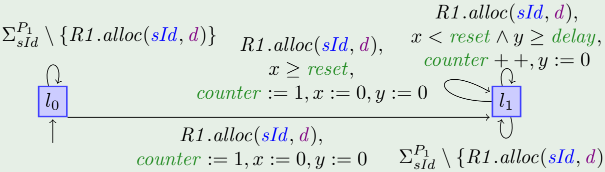
Syntax of PTAV

$\mathcal{A}(p) = (p, V, C, \Theta, L, l_0, F, X, \Sigma_p, \Delta)$.

- 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).

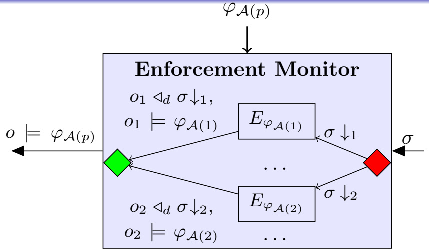
A PTAV is denoted as $A(p)$, and an instance of a PTAV for a value π of p is denoted as $A(\pi)$.

PTAV Example: in the context of resource allocation



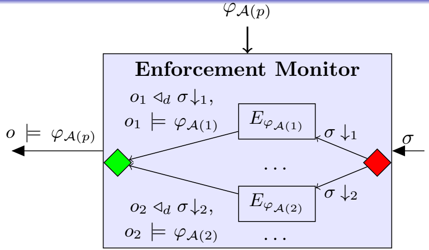
$delay = int\left(\frac{counter * sId}{6}\right)$

Enforcement of **parametric** timed properties



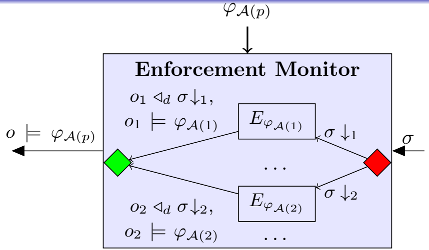
- Input/output timed words: $\sigma = (t_1, a_1(\pi_1, \eta_1)) \cdots (t_n, a_n(\pi_n, \eta_n))$.

Enforcement of **parametric** timed properties



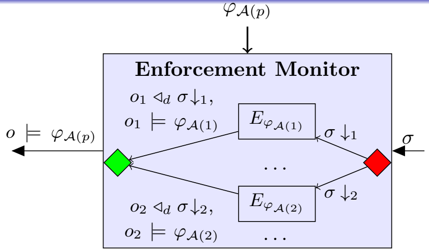
- Input/output timed words: $\sigma = (t_1, a_1(\pi_1, \eta_1)) \cdots (t_n, a_n(\pi_n, \eta_n))$.
- Property φ specified by PTAV.

Enforcement of **parametric** timed properties



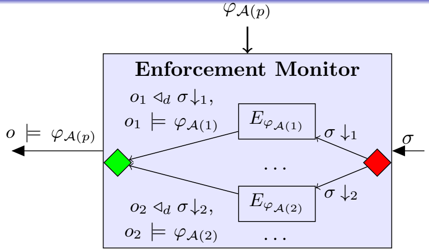
- Input/output timed words: $\sigma = (t_1, a_1(\pi_1, \eta_1)) \cdots (t_n, a_n(\pi_n, \eta_n))$.
- Property φ specified by PTAV.
- An instance of EM per parameter value (takes as input only the events with same parameter value).

Enforcement of **parametric** timed properties



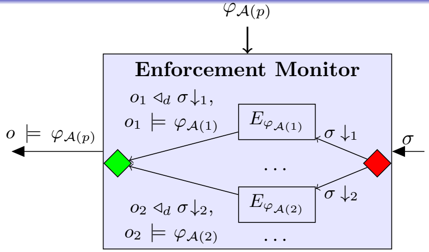
- Input/output timed words: $\sigma = (t_1, a_1(\pi_1, \eta_1)) \cdots (t_n, a_n(\pi_n, \eta_n))$.
- Property φ specified by PTAV.
- An instance of EM per parameter value (takes as input only the events with same parameter value).
- Slicing input.
  - $\sigma = (0.5, a(1, \eta_1)) \cdot (0.8, a(2, \eta_2))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

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 φ 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).

Enforcement of **parametric** timed properties

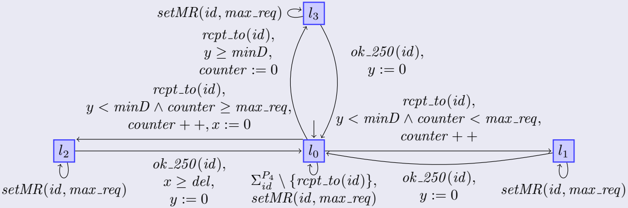


- Input/output timed words: $\sigma = (t_1, a_1(\pi_1, \eta_1)) \cdots (t_n, a_n(\pi_n, \eta_n))$.
- Property φ specified by PTAV.
- An instance of EM per parameter value (takes as input only the events with same parameter value).
- Slicing input.
  - $\sigma = (0.5, a(1, \eta_1)) \cdot (0.8, a(2, \eta_2))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

Other application domains

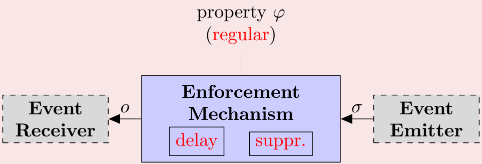
Protecting mail servers

If the number of RCPT_TO messages from a client is greater than maxreq, then there should be a delay of at least del t.u. before responding an OK_250.[1]



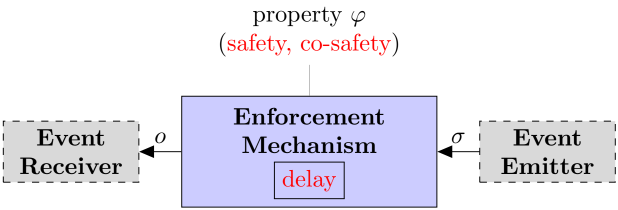
Conclusions

Enforcement monitoring for systems with timing requirements.



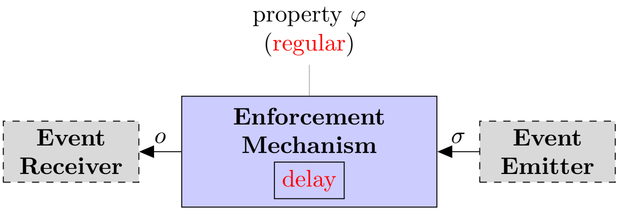
- Formally based runtime enforcement mechanisms for requirements with real-time constraints.
- For all regular timed properties modeled as a timed automaton.
- Enforcer can delay and suppress events.
- Enforcement mechanisms described at several levels of abstraction (enforcement function, enforcement monitor and algorithms).
- Prototype tool implementation (TIPEX).
- Requirements with constraints both on time and data (PTAV).

Summary: Evolution of results



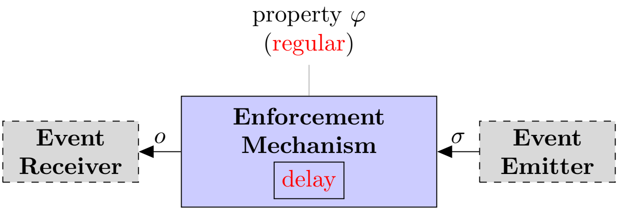
- Runtime Enforcement of Timed Properties **[RV 2012]**.
- Runtime Enforcement of Regular Timed Properties **[SAC-SVT 2014]**.
- Runtime Enforcement of Timed Properties Revisited **[FMSD Journal]**.
- Runtime Enforcement of Regular Timed Properties by Suppressing and Delaying Events **[SCP Journal]**.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

Summary: Evolution of results



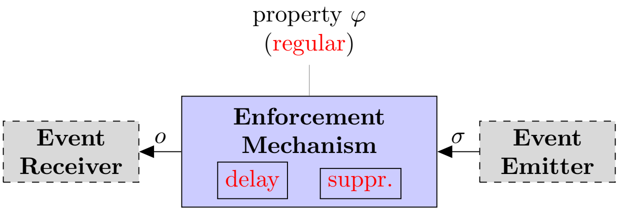
- Runtime Enforcement of Timed Properties **[RV 2012]**.
- Runtime Enforcement of Regular Timed Properties **[SAC-SVT 2014]**.
- Runtime Enforcement of Timed Properties Revisited **[FMSD Journal]**.
- Runtime Enforcement of Regular Timed Properties by Suppressing and Delaying Events **[SCP Journal]**.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

Summary: Evolution of results



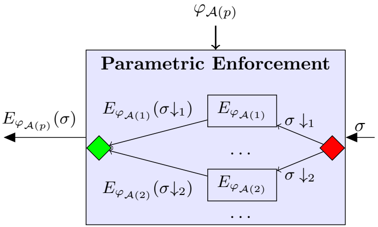
- Runtime Enforcement of Timed Properties **[RV 2012]**.
- Runtime Enforcement of Regular Timed Properties **[SAC-SVT 2014]**.
- Runtime Enforcement of Timed Properties Revisited **[FMSD Journal]**.
- Runtime Enforcement of Regular Timed Properties by Suppressing and Delaying Events **[SCP Journal]**.
- 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]**.

Perspectives

Theoretical extensions

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

Perspectives

Theoretical extensions

- Predictive enforcement.
  - $E_{\varphi/\psi} : \psi \subseteq \mathrm{tw}(\Sigma) \rightarrow \{\epsilon\} \cup \varphi \subseteq \mathrm{tw}(\Sigma)$.
  - 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.

Perspectives

Theoretical extensions

- Predictive enforcement.
  - $E_{\varphi/\psi} : \psi \subseteq \mathrm{tw}(\Sigma) \rightarrow \{\epsilon\} \cup \varphi \subseteq \mathrm{tw}(\Sigma)$.
  - 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.
  - $E_{\varphi/\psi} : \psi \subseteq \text{tw}(\Sigma) \rightarrow \{\epsilon\} \cup \varphi \subseteq \text{tw}(\Sigma)$.
  - 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).

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.
- 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.