Runtime Enforcement of Regular Timed Properties

Srinivas Pinisetty, Yliès Falcone[2], Thierry Jéron[1], Hervé Marchand[1]

INRIA Rennes - Bretagne Atlantique, France

Université Grenoble I, Laboratoire d'Informatique de Grenoble, France

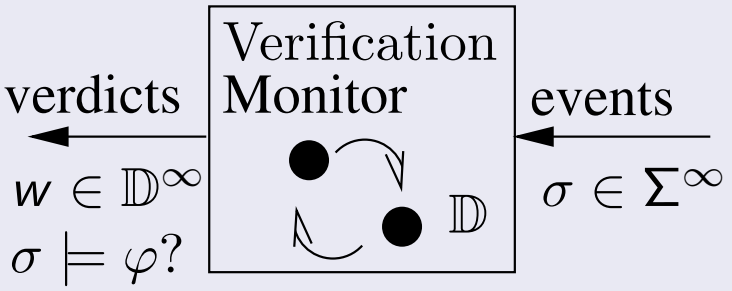
SAC-SVT 2014 (Gyeongju, Korea),
extended to Science of Computer Programming

Runtime verification and enforcement (monitors)

Runtime verification and enforcement:

- Monitor observing a stream σ of system events (e.g., trace, log, messages).
- Requested property φ.
- No system model.

Runtime verification



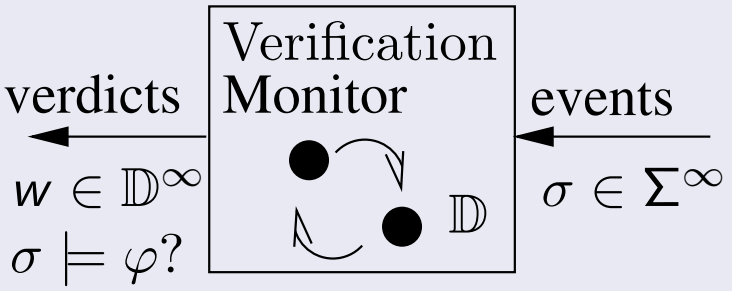
- Does σ satisfy φ ?
- Output: stream of **verdicts**.

Runtime verification and enforcement (monitors)

Runtime verification and enforcement:

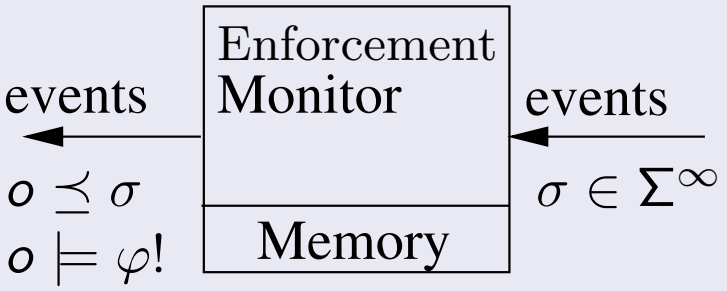
- Monitor observing a stream σ of system events (e.g., trace, log, messages).
- Requested property φ.
- No system model.

Runtime verification



- Does σ satisfy φ ?
- Output: stream of **verdicts**.

Runtime enforcement



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

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

Related work on monitoring

Runtime Enforcement of **Untimed** properties

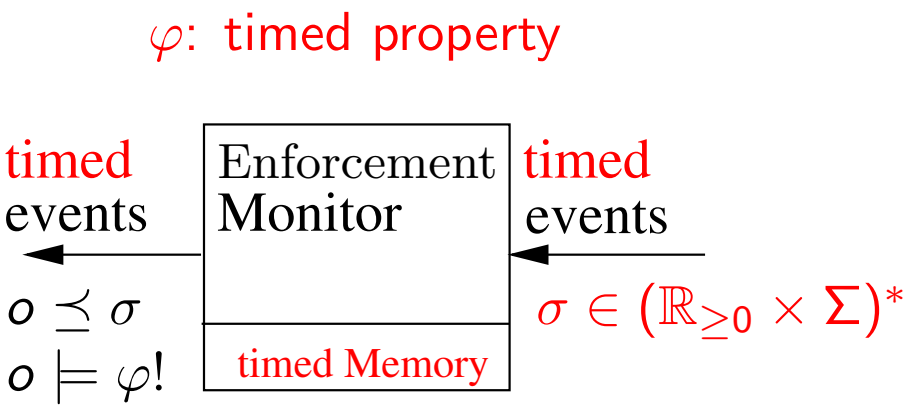
- Enforceable security policies – F. B. Schneider et al-2000.
- Enforcement Monitoring wrt. the Safety-Progress Classification of Properties – Y. Falcone et al-2010.
- Runtime enforcement of non-safety policies – J. Ligatti et al-2009.

Runtime Verification of Timed properties

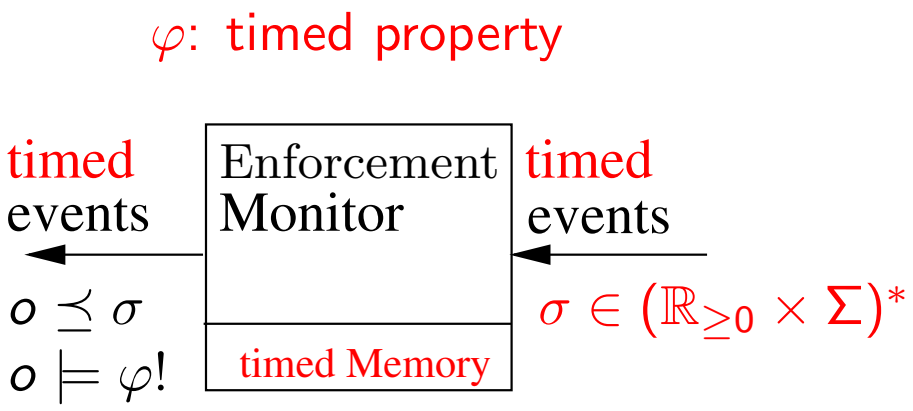
Efforts mainly to *verify* timed properties at runtime:

- Runtime verification of TLTL – A. Bauer 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.

Problem and Contributions



Problem and Contributions



A formal framework for runtime enforcement of timed properties

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

Outline

1. Specifying Timed Properties
2. Runtime Enforcement of Regular Timed Properties
   - Requirements on an Enforcement Mechanism
   - Functional Definition of an Enforcement Mechanism
   - Operational Description of an Enforcement Mechanism
   - Algorithmic Description of an Enforcement Mechanism
3. Conclusions and Future Work

1 Specifying Timed Properties

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

3 Conclusions and Future Work

Specifying timed properties

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

Specifying timed properties

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

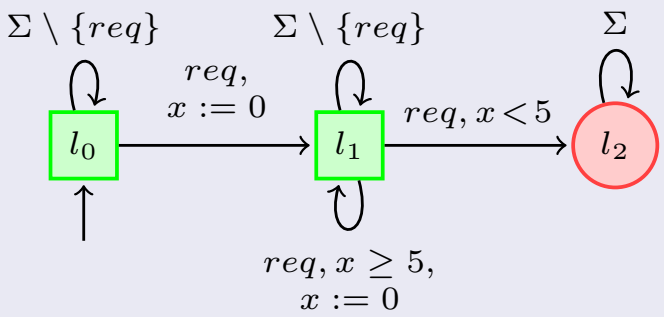
Safety, co-safety and response properties specified by TAs

Specifying timed properties

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

Safety, co-safety and response properties specified by TAs

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



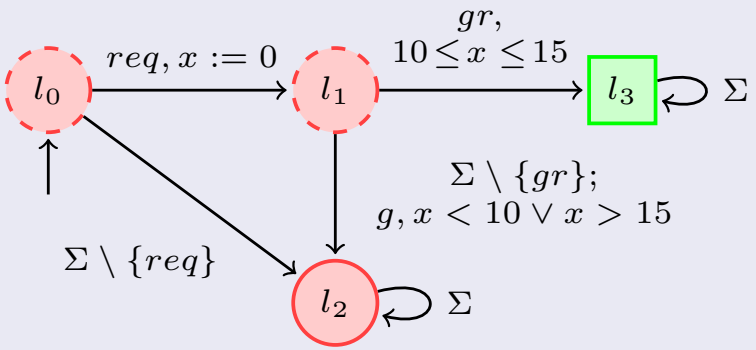
$\Sigma = \{req\}$
"A delay of 5 t.u. between any two requests."

Specifying timed properties

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

Safety, co-safety and response properties specified by TAs

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



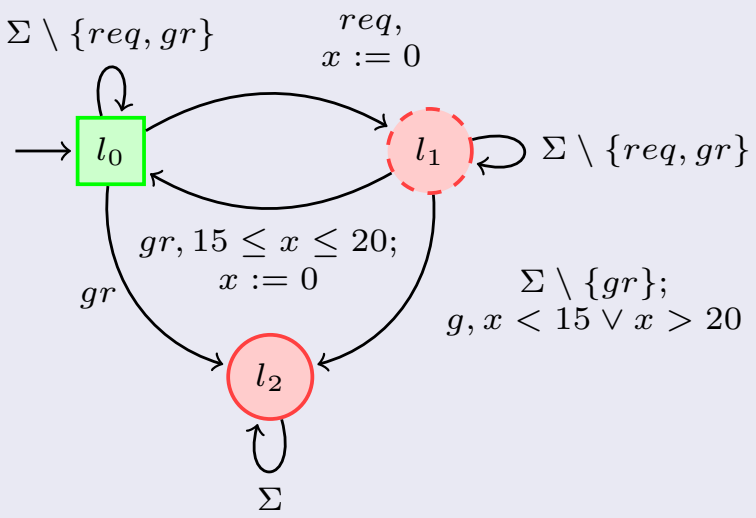
$\Sigma = \{req, gr\}$
"A request, and then a grant should arrive between 10 and 15 t.u."

Specifying timed properties

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

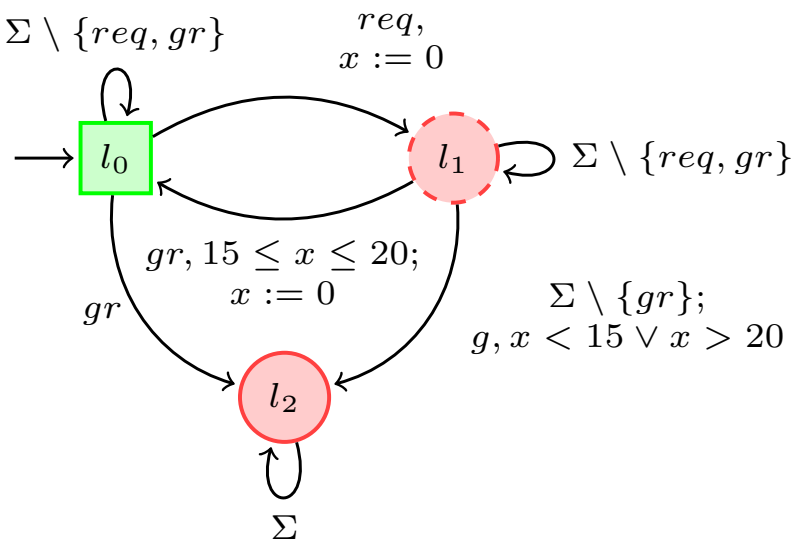
Safety, co-safety and response properties specified by TAs

Response: any property.



$\Sigma = \{req, gr\}$
"Requests and grants should alternate in this order with a delay between 15 and 20 t.u between the request and the grant."

Example: response property



- $\Sigma = \{req, gr\}$
- $(3, req) \cdot (18, gr) \cdot (23, req) \cdot (42, gr)$

$\epsilon \models \varphi$.
$(3, req) \not\models \varphi$.
$(3, req) \cdot (18, gr) \models \varphi$.
$(3, req) \cdot (18, gr) \cdot (23, req) \not\models \varphi$.
$(3, req) \cdot (18, gr) \cdot (23, req) \cdot (42, gr) \models \varphi$.

Response properties are neither prefix nor extension closed.

Semantics of TA

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

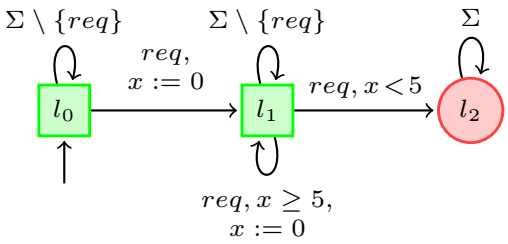
Partition of states

The set of states Q of $[\![\mathcal{A}]\!]$ can be partitionned into $Q = G^C \cup G \cup B^C \cup B$ where $Q_F = G^c \cup G$ and $Q \setminus Q_F = B^C \cup B$ and

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

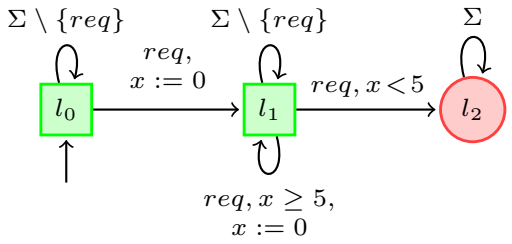
Major Challenges for enforcement

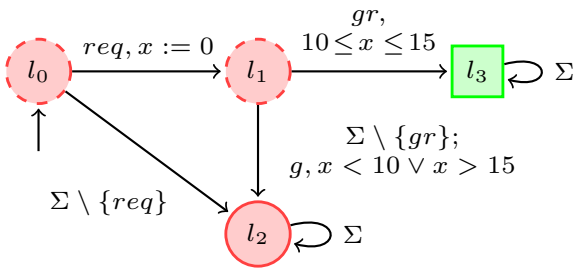
- for safety properties: on-line correction for each event.



Major Challenges for enforcement

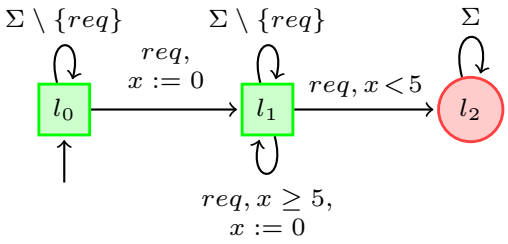
- for safety properties: on-line correction for each event.
- for co-safety properties: wait for an event allowing correction.

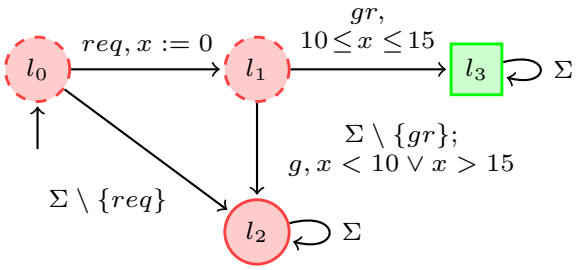


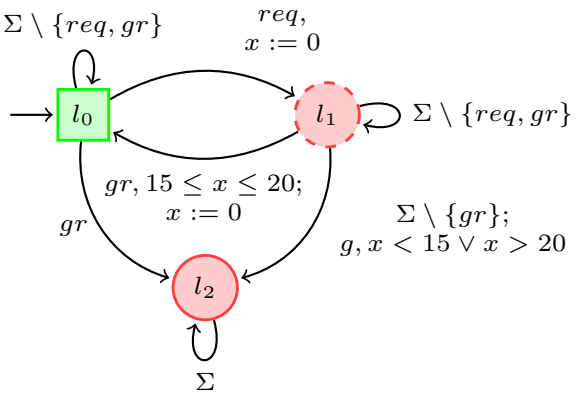


Major Challenges for enforcement

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





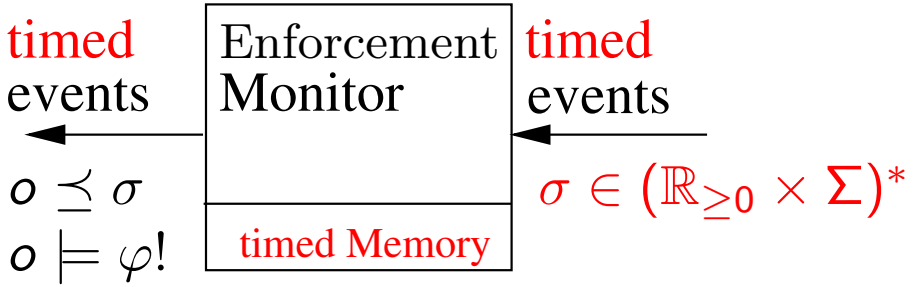


1 Specifying Timed Properties

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

3 Conclusions and Future Work

Summary of the approach

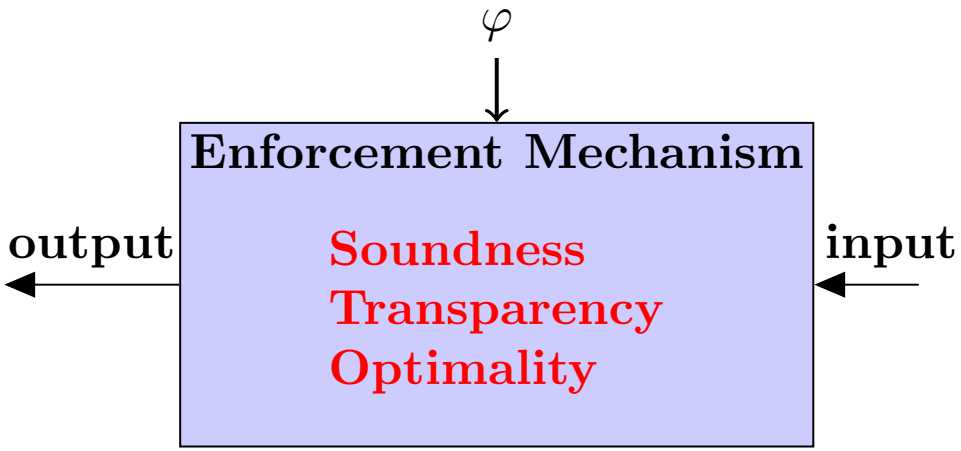


What can an enforcement mechanism do?

- CANNOT change the order of events.
- CANNOT insert events.
- CAN increase the dates of events so satisfy φ.
- CAN suppress events if no future can satisfy φ.

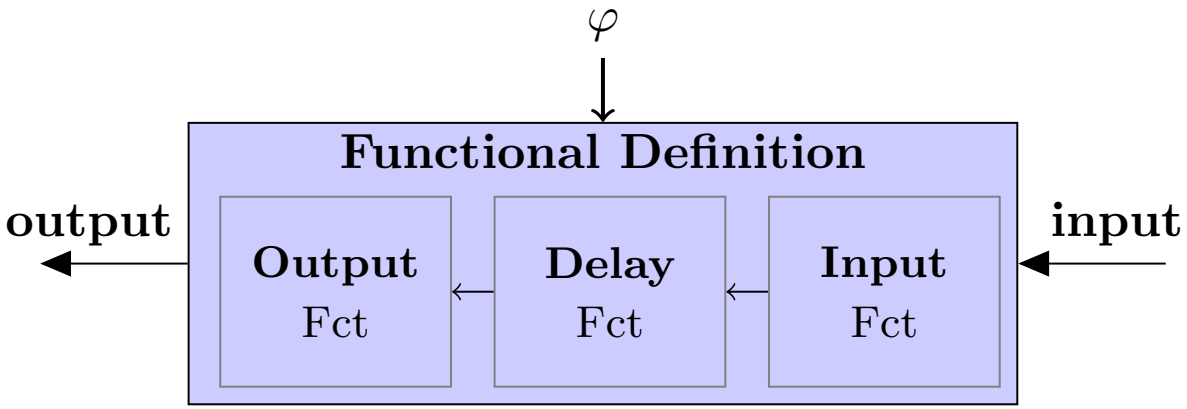
$\hookrightarrow$ the enforcement monitor is a "delayer with suppression".

Summary of the approach



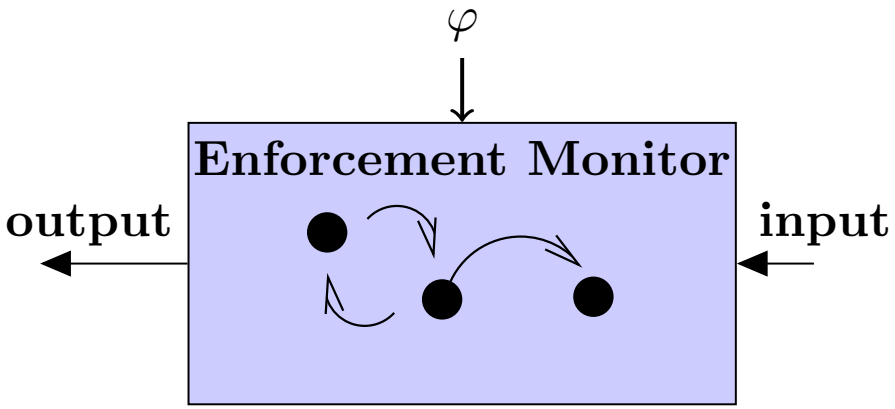
- **Requirements** for any enforcement mechanism for φ
- **Functional definition**:
  - description of the global input/output behavior,
  - satisfying requirements;
- **Enforcement monitor**:
  - timed operational behavior as a rule-based transition system,
  - refining the functional definition;
-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

Summary of the approach



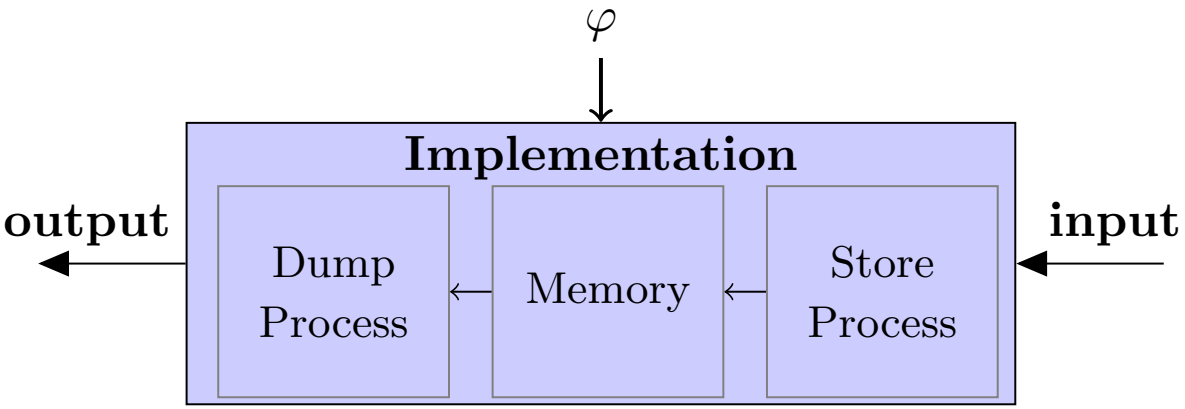
- **Requirements** for any enforcement mechanism for φ
- **Functional definition**:
  - description of the global input/output behavior,
  - satisfying requirements;
- **Enforcement monitor**:
  - timed operational behavior as a rule-based transition system,
  - refining the functional definition;
-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

Summary of the approach



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

Summary of the approach



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

Preliminaries: orders on timed words

Subsequence: $(3, a) \cdot (8, c) \lhd (3, a) \cdot (5, b) \cdot (8, c)$
$w' \lhd w$ if w' obtained from w by suppressions.

Delaying order $\succcurlyeq_d$: $(4, a) \cdot (7, b) \cdot (9, c) \succcurlyeq_d (3, a) \cdot (5, b) \cdot (8, c)$.
$\sigma' \succcurlyeq_d \sigma$ if they have same untimed projections but dates in σ' exceed corresponding dates in σ.

Delaying subsequence order $\lhd_d$: $(4, a) \cdot (9, c) \lhd_d (3, a) \cdot (5, b) \cdot (8, c)$
$\sigma' \lhd_d \sigma \stackrel{\text{def}}{=} \exists \sigma'' \in \mathsf{tw}(\Sigma) : \sigma'' \lhd \sigma \wedge \sigma' \succcurlyeq_d \sigma''$
i.e., σ' obtained from σ by first suppressing some actions, and then increasing the dates of the actions to be kept.

Lexical order $\preceq_{\text{lex}}$: $(3, a) \cdot (5, b) \cdot (8, c) \preceq_{\text{lex}} (3, a) \cdot (6, b) \cdot (7, c)$.
$\epsilon \preceq_{\text{lex}} \epsilon$, and for two events with identical actions (t, a) and (t', a), $(t, a) \cdot \sigma \preceq_{\text{lex}} (t', a) \cdot \sigma'$ if $t \leq t' \vee (t = t' \wedge \sigma \preceq_{\text{lex}} \sigma')$.

Requirements on an Enforcement Mechanism (1)

An **enforcement function** for φ is a function $E_\varphi : \mathrm{tw}(\Sigma) \to \mathrm{tw}(\Sigma)$ which transforms timed words, and satisfies the following requirements:

Requirements on an Enforcement Mechanism (1)

An **enforcement function** for φ is a function $E_\varphi : \mathrm{tw}(\Sigma) \to \mathrm{tw}(\Sigma)$ which transforms timed words, and satisfies the following requirements:

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.

Requirements on an Enforcement Mechanism (1)

An **enforcement function** for φ is a function $E_\varphi : \mathrm{tw}(\Sigma) \to \mathrm{tw}(\Sigma)$ which transforms timed words, and satisfies the following requirements:

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 is either empty or satisfies φ.

Requirements on an Enforcement Mechanism (1)

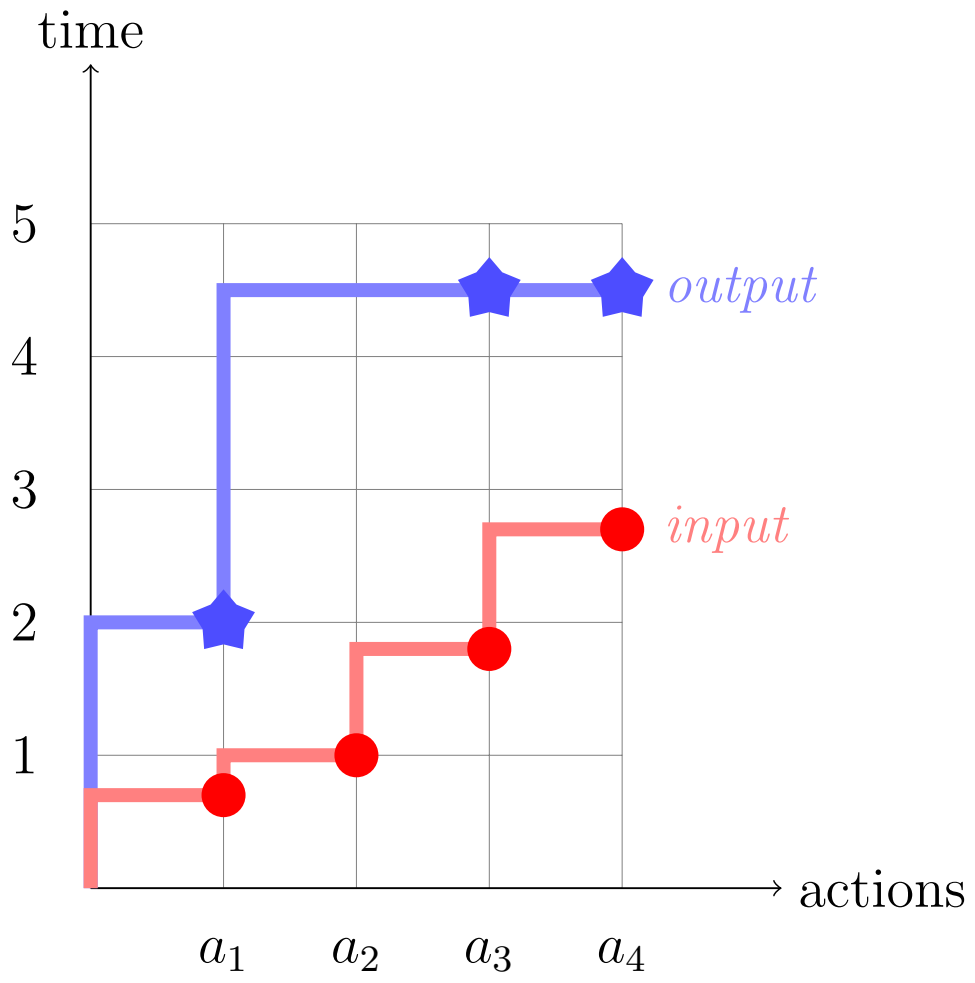
An **enforcement function** for φ is a function $E_\varphi : \mathrm{tw}(\Sigma) \to \mathrm{tw}(\Sigma)$ which transforms timed words, and satisfies the following requirements:

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.

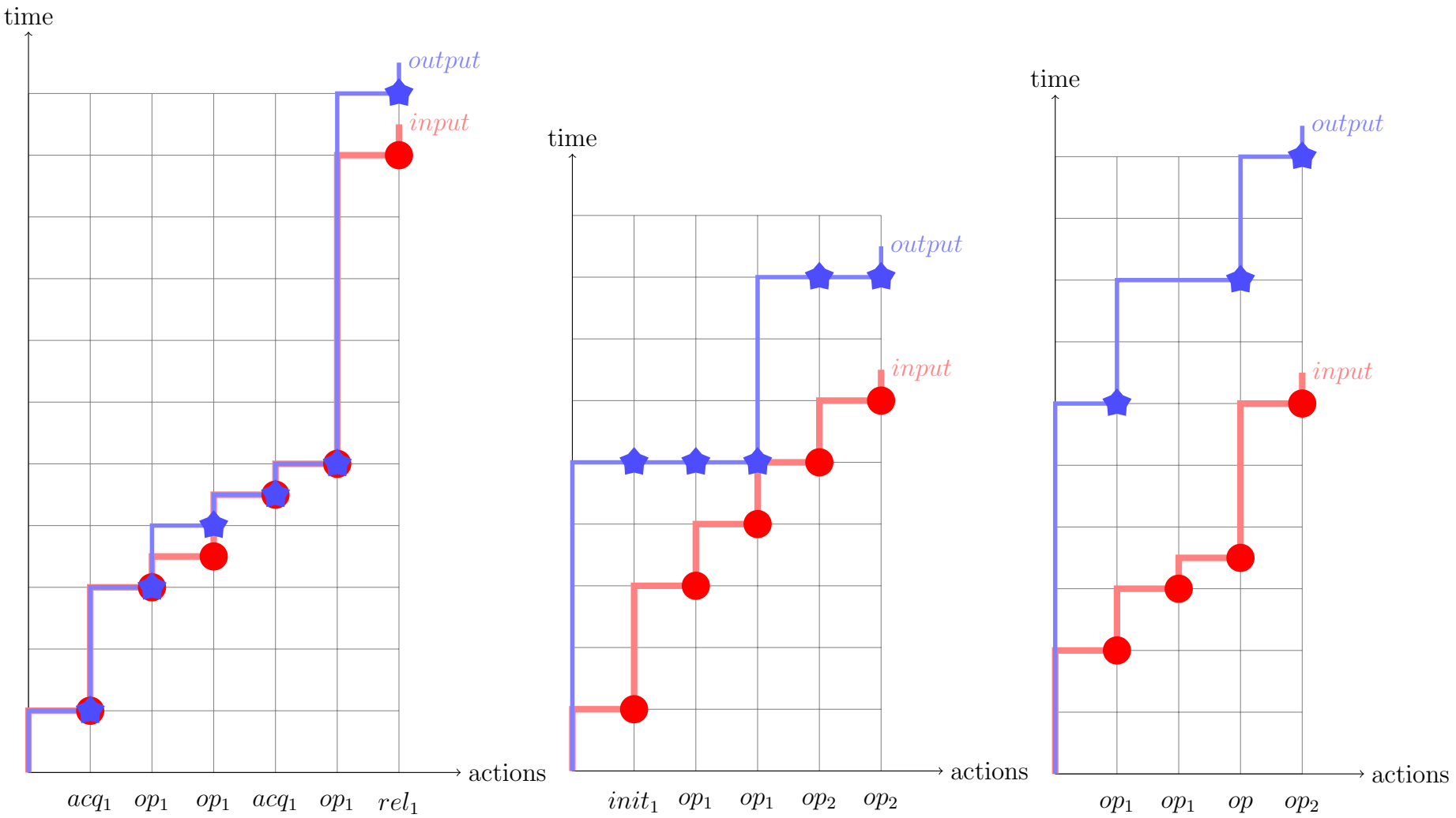
Soundness: $\forall \sigma \in \mathrm{tw}(\Sigma) : E_\varphi(\sigma) \models \varphi \vee E_\varphi(\sigma) = \epsilon$.
i.e., the output is either empty or satisfies φ.

Transparency: $\forall \sigma \in \mathrm{tw}(\Sigma) : E_\varphi(\sigma) \lhd_d \sigma$.
i.e., the output ia a delaying subsequence of the input.

Transparency



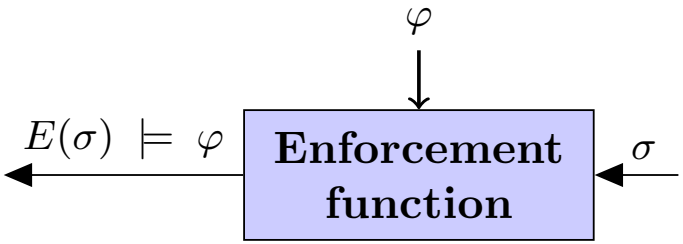
Transparency (2)



Safety, co-safety and response

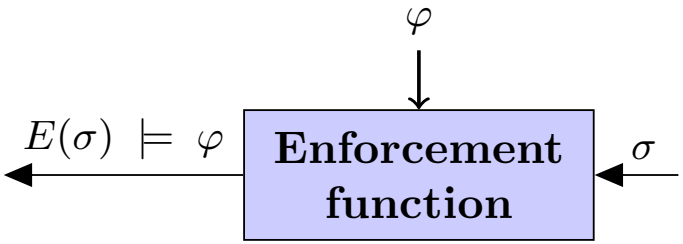
Functional definition: general scheme

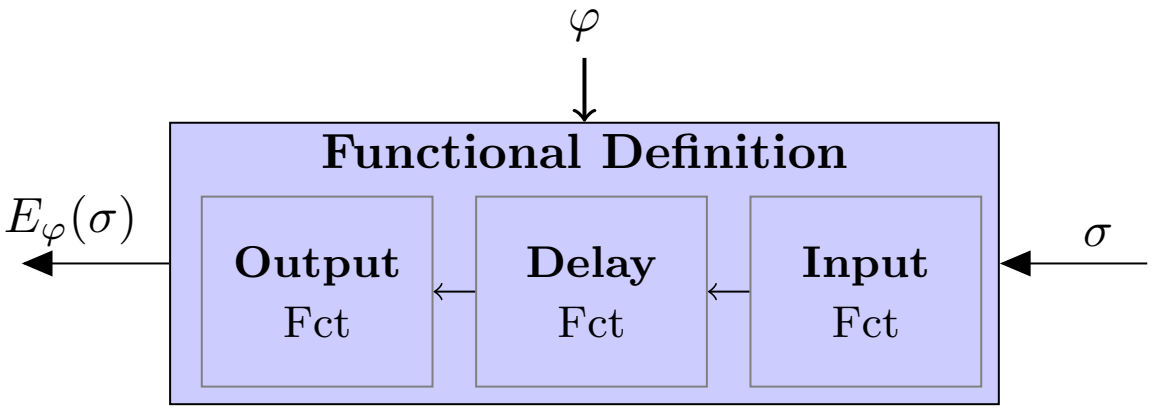
$$E_\varphi : \mathrm{tw}(\Sigma) \to \mathrm{tw}(\Sigma).$$



Functional definition: general scheme

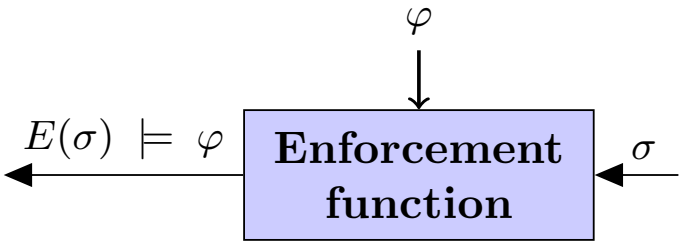
$$E_\varphi : \mathrm{tw}(\Sigma) \to \mathrm{tw}(\Sigma).$$

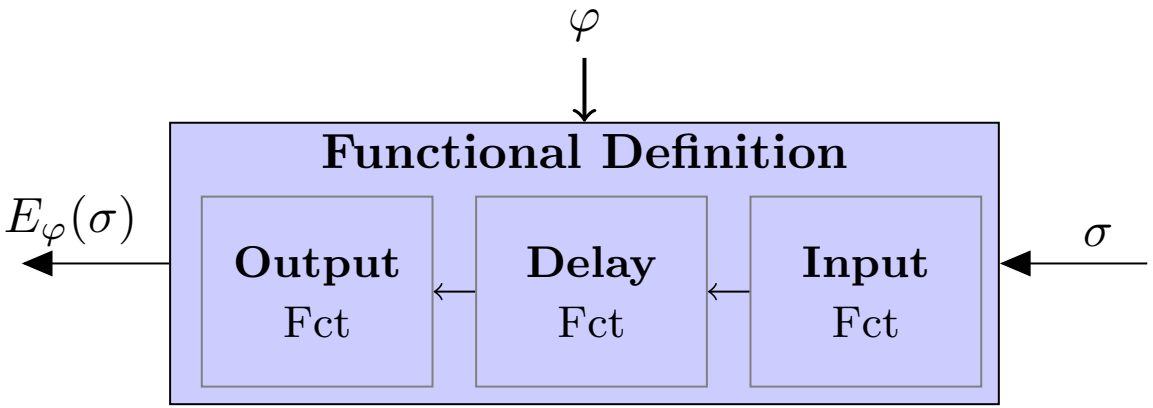




Functional definition: general scheme

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





Input and output functions are realized by the observation function:

$$\mathrm{obs}(\sigma, t) = \max_{\preccurlyeq}\{\sigma' \mid \sigma' \preccurlyeq \sigma \wedge \mathrm{time}(\sigma') \leq t\}.$$

Functional definition: principle

$$E_\varphi : \mathrm{tw}(\Sigma) \to \mathrm{tw}(\Sigma) \text{ is defined as } E_\varphi(\sigma) = \Pi_1\left(\mathrm{store}_\varphi(\sigma)\right)$$

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

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

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

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

- if $\sigma_s \cdot \sigma_c \cdot (t, a)$ allows to satisfy φ by delaying $\sigma_c \cdot (t, a)$ ($Q_F = G^C \cup G$ reached by some minimal delaying K): $(\sigma_s \cdot K, \epsilon)$
- if no continuation of $\sigma_s \cdot \sigma_c \cdot (t, a)$ allows to satisfy φ (*Bad* reached by all delayings), (t, a) suppressed: (σ_s, σ_c)
- otherwise some continuation may still allow to satisfy φ (Q_F is reachable): $(\sigma_s, \sigma_c \cdot (t, a))$.

Functional definition: formal definition

$$E_\varphi : \mathsf{tw}(\Sigma) \to \mathsf{tw}(\Sigma) \text{ is defined as } E_\varphi(\sigma) = \Pi_1\left(\mathrm{store}_\varphi(\sigma)\right)$$

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

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

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

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

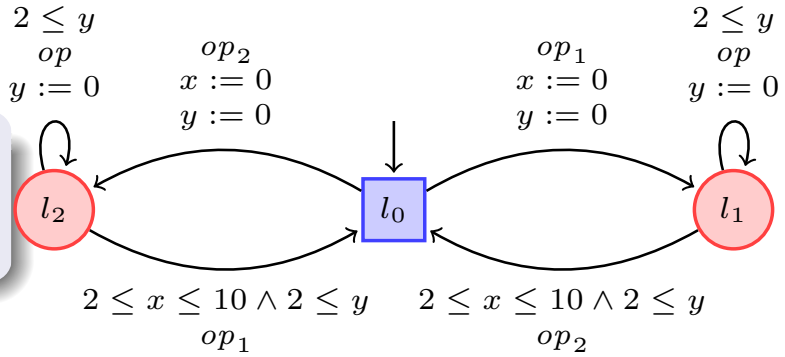
with $\kappa_\varphi(\sigma_s, \sigma'_c) = \mathsf{CanD}(\sigma'_c) \cap \sigma_s^{-1} \cdot \varphi$ (or $\sigma_s^{-1} \cdot \mathrm{pref}(\varphi)$ for $\kappa_{\mathrm{pref}(\varphi)}$)

and
$$\begin{cases} \mathsf{CanD}(\sigma'_c) = \{w \in \mathsf{tw}(\Sigma) \mid w \succcurlyeq_d \sigma'_c \wedge \mathrm{start}(w) \geq \mathrm{end}(\sigma'_c)\} \\ \quad \text{i.e., sequences delaying } \sigma'_c \text{ and starting after } t \\ \sigma_s^{-1} \cdot \varphi = \{w \in \mathsf{tw}(\Sigma) \mid \sigma_s \cdot w \models \varphi\} \\ \quad \text{i.e., what remains to be satisfied from } \varphi \text{ after reading } \sigma_s \end{cases}$$

Functional definition: Example and Evolution over time

Let $\mathrm{obs}(\sigma, t) = \max_{\preccurlyeq}\{\sigma' \mid \sigma' \preccurlyeq \sigma \wedge \mathrm{time}(\sigma') \leq t\}$.

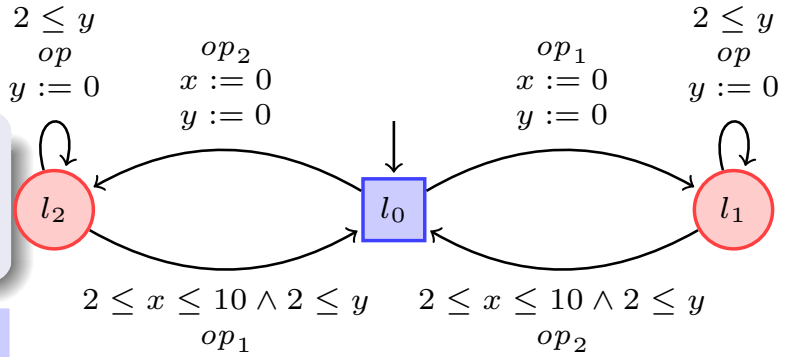
$\Sigma = \{op1, op2, op\}$.
$\sigma = (2, op1) \cdot (3, op1) \cdot (3.5, op) \cdot (6, op2)$.



Functional definition: Example and Evolution over time

Let $\mathrm{obs}(\sigma, t) = \max_{\preccurlyeq}\{\sigma' \mid \sigma' \preccurlyeq \sigma \wedge \mathrm{time}(\sigma') \leq t\}$.

$\Sigma = \{op1, op2, op\}$.
$\sigma = (2, op1) \cdot (3, op1) \cdot (3.5, op) \cdot (6, op2)$.



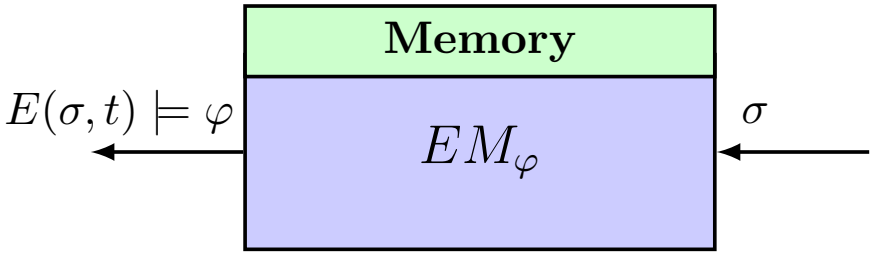
$t \in [0, 2[$	$\mathrm{obs}(\sigma_3, t) = \epsilon$ $\mathrm{store}_{\varphi_3}(\mathrm{obs}(\sigma_3, t)) = (\epsilon, \epsilon)$ $\mathrm{obs}\left(E_{\varphi_3}\left(\mathrm{obs}\left(\sigma_3, t\right)\right)\right) = \mathrm{obs}(\epsilon, t) = \epsilon$
$t \in [2, 3[$	$\mathrm{obs}(\sigma_3, t) = (2, op_1)$ $\mathrm{store}_{\varphi_3}(\mathrm{obs}(\sigma_3, t)) = (\epsilon, (2, op_1))$ $\mathrm{obs}\left(E_{\varphi_3}\left(\mathrm{obs}\left(\sigma_3, t\right)\right)\right) = \mathrm{obs}(\epsilon, t) = \epsilon$
$t \in [3, 3.5[$	$\mathrm{obs}(\sigma_3, t) = (2, op_1) \cdot (3, op_1)$ $\mathrm{store}_{\varphi_3}(\mathrm{obs}(\sigma_3, t)) = (\epsilon, (2, op_1))$ $\mathrm{obs}\left(E_{\varphi_3}\left(\mathrm{obs}\left(\sigma_3, t\right)\right)\right) = \mathrm{obs}(\epsilon, t) = \epsilon$
$t \in [3.5, 6[$	$\mathrm{obs}(\sigma_3, t) = (2, op_1) \cdot (3, op_1) \cdot (3.5, op)$ $\mathrm{store}_{\varphi_3}(\mathrm{obs}(\sigma_3, t)) = (\epsilon, (2, op_1) \cdot (3.5, op))$ $\mathrm{obs}\left(E_{\varphi_3}\left(\mathrm{obs}\left(\sigma_3, t\right)\right)\right) = \mathrm{obs}\left(\epsilon, t\right) = \epsilon$
$t \in [6, \infty[$	$\mathrm{obs}(\sigma_3, t) = (2, op_1) \cdot (3, op_1) \cdot (3.5, op) \cdot (6, op_2)$ $\mathrm{store}_{\varphi_3}(\mathrm{obs}(\sigma_3, t)) = ((6, op_1) \cdot (8, op) \cdot (10, op_2), \epsilon)$ $\mathrm{obs}\left(E_{\varphi_3}\left(\mathrm{obs}\left(\sigma_3, t\right)\right)\right) = \mathrm{obs}\left((6, op_1) \cdot (8, op) \cdot (10, op_2)\right, t)$

The enforcement function satisfies the requirements

Proposition: Enforcement function vs requirements

The proposed definition of enforcement function satisfies the **soundness**, **transparency**, and **optimality** requirements.

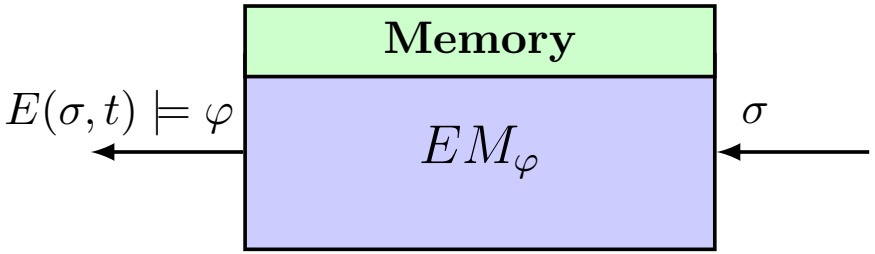
Enforcement monitor



A *rule-based transition system*:

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

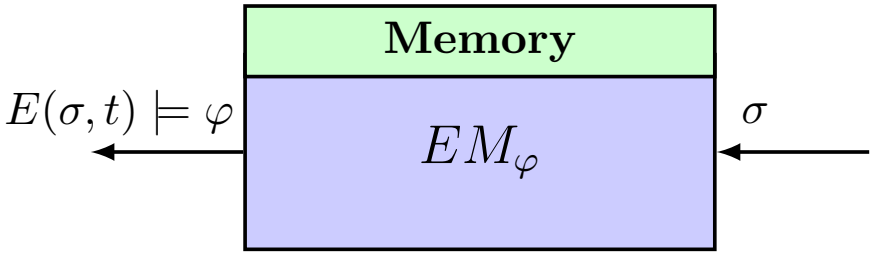
Enforcement monitor



A *rule-based transition system*:

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

Enforcement monitor



A *rule-based transition system*:

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

Enforcement monitor: operations

1. *store-*φ

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

Enforcement monitor: operations

1. *store*-φ

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

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

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

Enforcement monitor: operations

1. *store*-φ

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

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

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

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

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

Enforcement monitor: operations

1. *store*-φ

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

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

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

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

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

4. *dump*

at t when $\sigma_{ms} = (t, a) \cdot .\sigma'_{ms}$, event (t, a) is released.

5. *idle*

when no other rule applies, time elapses of δ.

Enforcement Monitor: correctness

Implementation relation between Enforcement Monitor and Enforcement Function

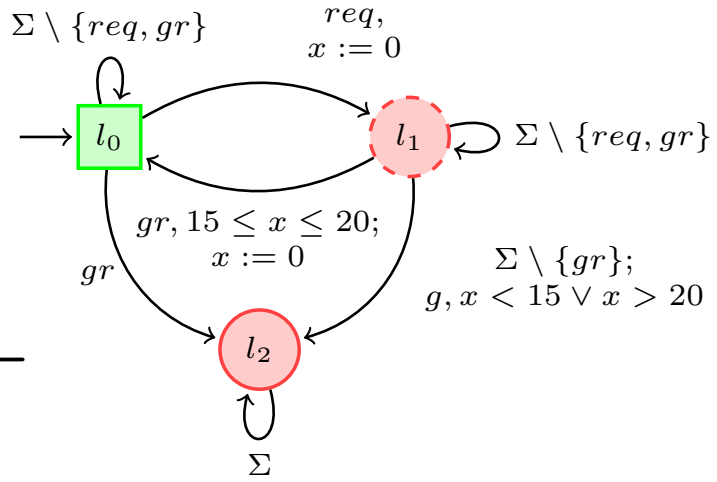
Given some property φ, at any time t, the input/output behavior of the synthesized enforcement monitor is the same as the one of the corresponding enforcement function at time t, i.e.,$\mathrm{obs}(E_\varphi(\sigma), t)$.

Corollary

Enforcement Monitors respect soundness, transparency, and optimality.

Enforcement Monitor: example

$t = 0$ - Operation: none

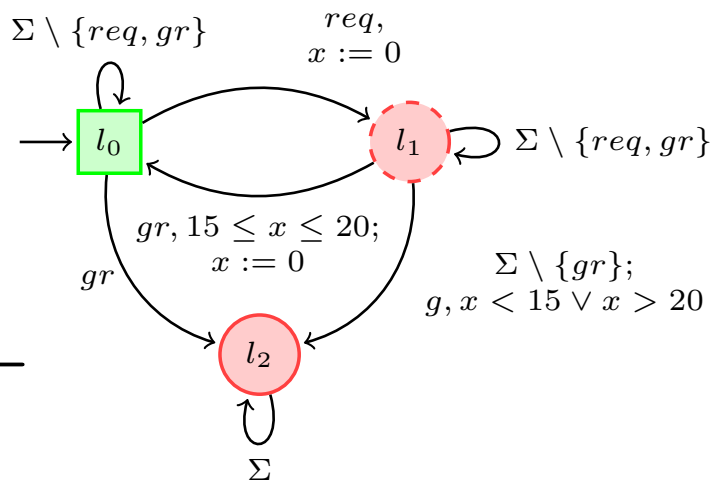


$\epsilon \longleftarrow$ $\longleftarrow (3, r) \cdot (13, g) \cdot (16, g) \cdot (21, r)$

q	σ_{ms}	σ_{mc}
$(l_0, 0)$	ϵ	ϵ

Enforcement Monitor: example

$t = 3$ - Operation: idle(3)



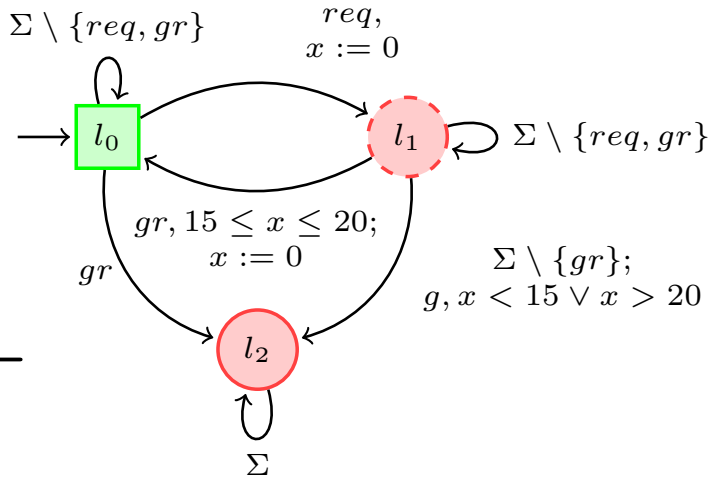
$\epsilon \longleftarrow$ $\longleftarrow (3, r) \cdot (13, g) \cdot (16, g) \cdot (21, r)$

q	σ_{ms}	σ_{mc}
$(l_0, 0)$	ϵ	ϵ

Enforcement Monitor: example

$t = 3$ - Operation: *store-$\overline{\varphi}$*

$\epsilon \longleftarrow$

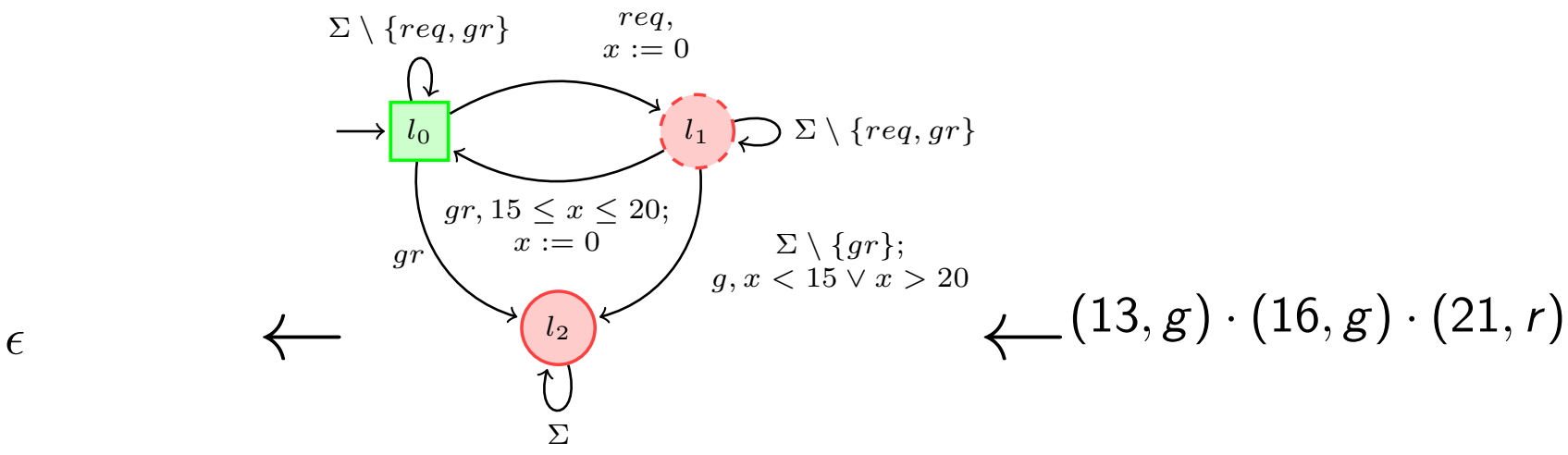


$\longleftarrow (13, g) \cdot (16, g) \cdot (21, r)$

q	σ_{ms}	σ_{mc}
$(l_0, 0)$	ϵ	$(3, r)$

Enforcement Monitor: example

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

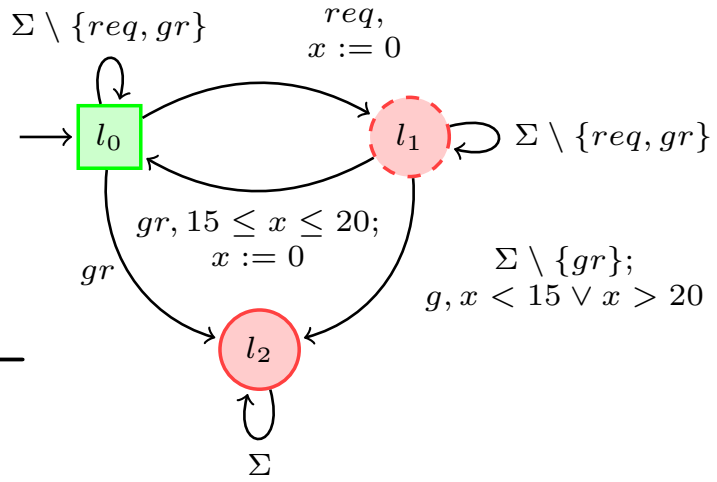


q	σ_{ms}	σ_{mc}
$(l_0, 0)$	ϵ	$(3, r)$

Enforcement Monitor: example

$t = 13$ - Operation: *store-φ*

ϵ ⟵

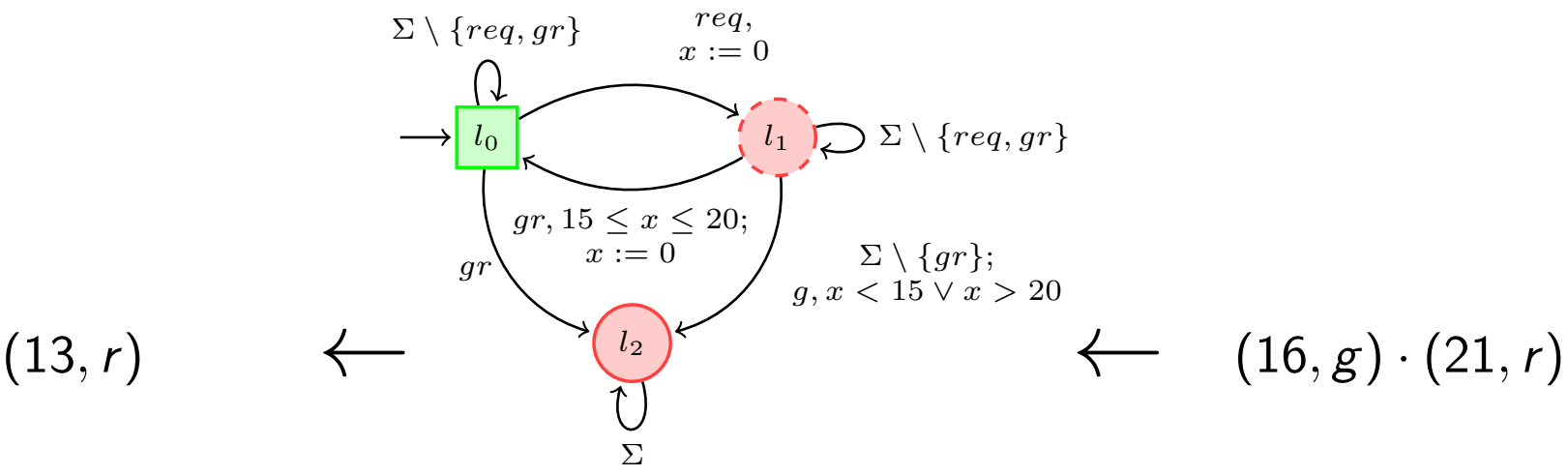


⟵ $(16, g) \cdot (21, r)$

q	σ_{ms}	σ_{mc}
$(l_0, 0)$	$(13, r) \cdot (28, g)$	ϵ

Enforcement Monitor: example

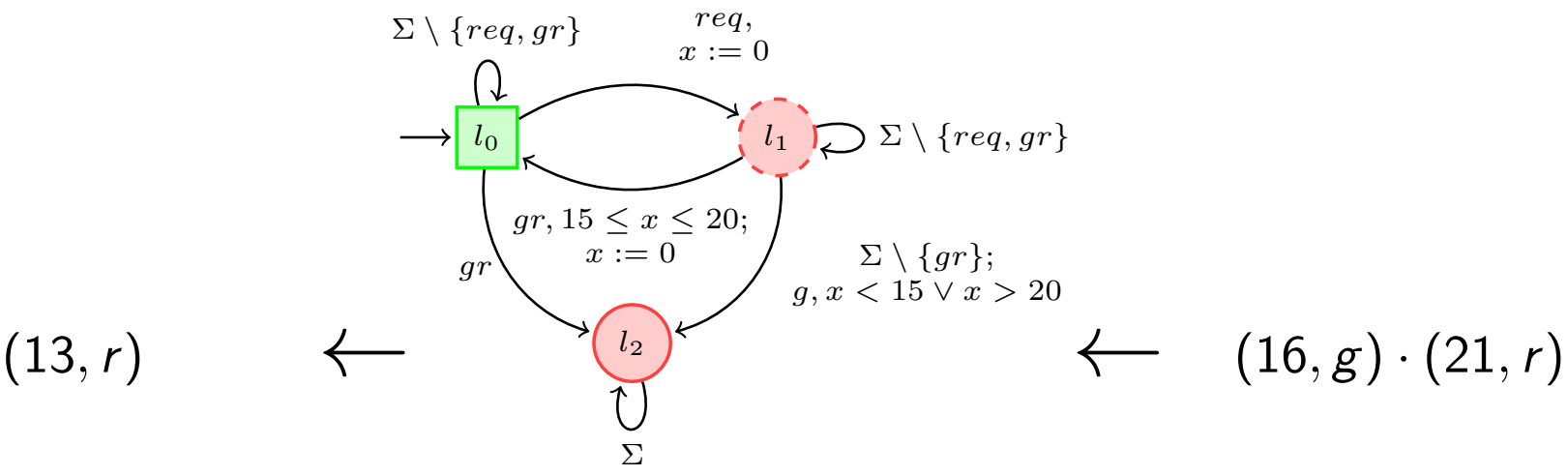
$t = 13$ - Operation: *dump*



q	σ_{ms}	σ_{mc}
$(l_0, 0)$	$(28, g)$	ϵ

Enforcement Monitor: example

$t = 16$ - Operation: idle(3)

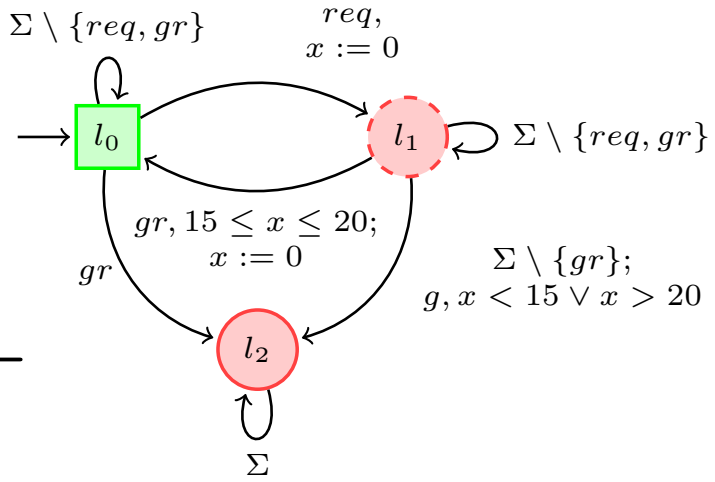


q	σ_{ms}	σ_{mc}
$(l_0, 0)$	$(28, g)$	ϵ

Enforcement Monitor: example

$t = 16$ - Operation: *store-sup-$\overline{\varphi}$*

$(13, r) \leftarrow$

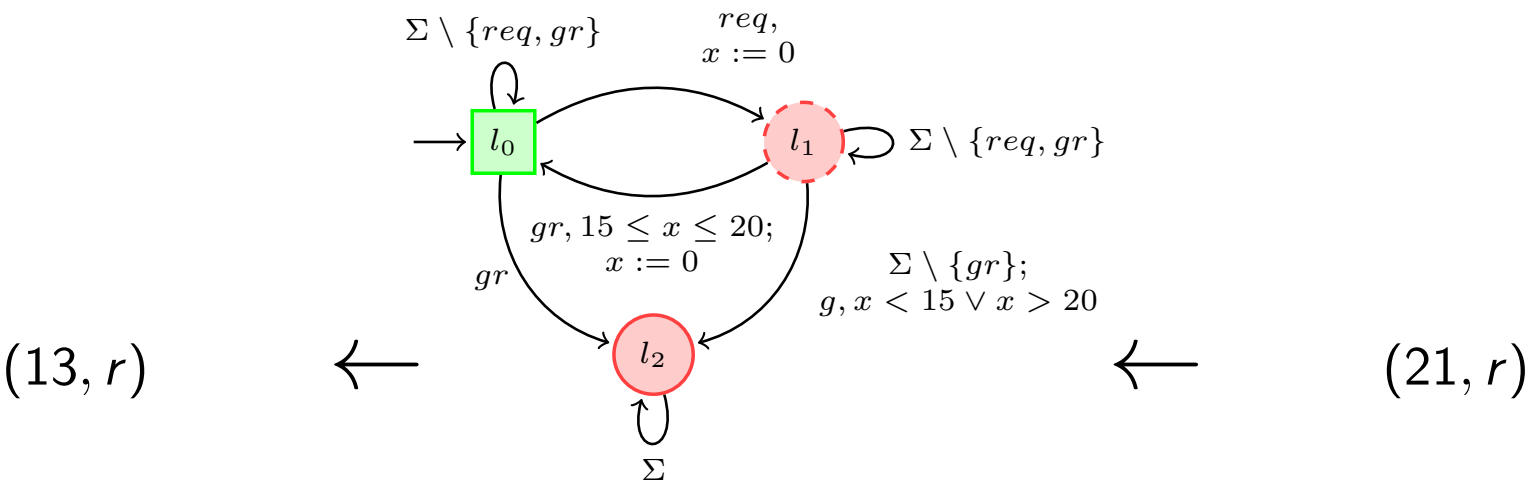


$\leftarrow (21, r)$

q	σ_{ms}	σ_{mc}
$(l_0, 0)$	$(28, g)$	ϵ

Enforcement Monitor: example

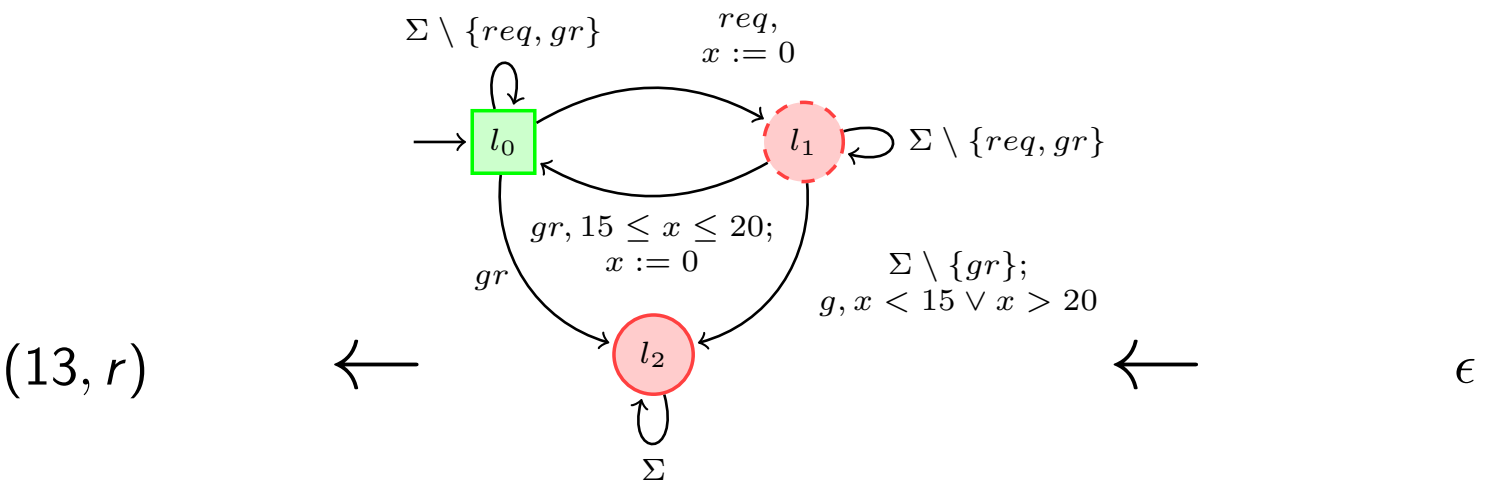
$t = 21$ - Operation: idle(5)



q	σ_{ms}	σ_{mc}
$(l_0, 0)$	$(28, g)$	ϵ

Enforcement Monitor: example

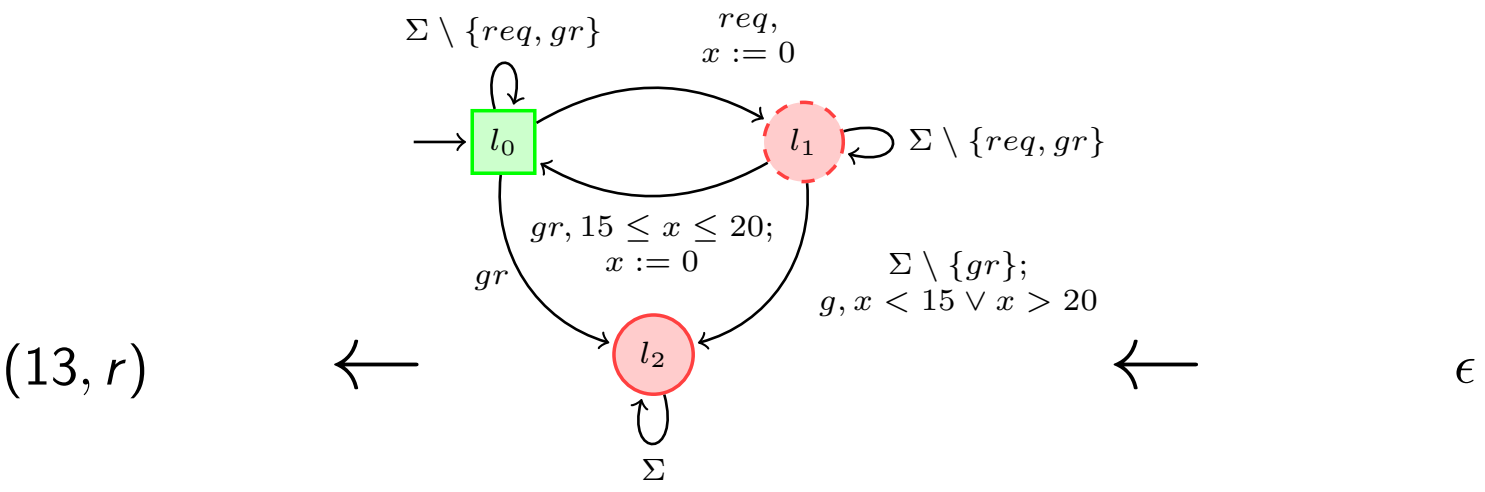
$t = 21$ - Operation: *store-$\overline{\varphi}$*



q	σ_{ms}	σ_{mc}
$(l_0, 0)$	$(28, g)$	$(21, r)$

Enforcement Monitor: example

$t = 28$ - Operation: idle(7)

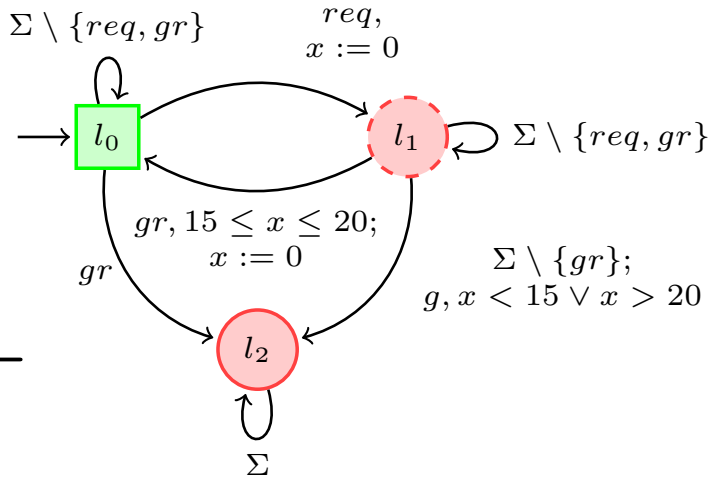


q	σ_{ms}	σ_{mc}
$(l_0, 0)$	$(28, g)$	$(21, r)$

Enforcement Monitor: example

$t = 28$ - Operation: *dump*

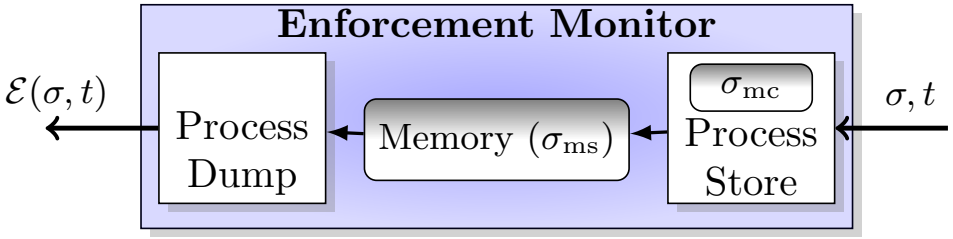
$(13, r) \cdot (28, g) \leftarrow$



$\leftarrow \epsilon$

q	σ_{ms}	σ_{mc}
$(l_0, 0)$	ϵ	$(21, r)$

Implementation



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

1 Specifying Timed Properties

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

3 Conclusions and Future Work

Conclusions and Future Work

Enforcement monitoring for systems with timing requirements.

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

Future Work

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

Some remarks / questions

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