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

Runtime verification and enforcement (monitors)

Runtime verification and enforcement:

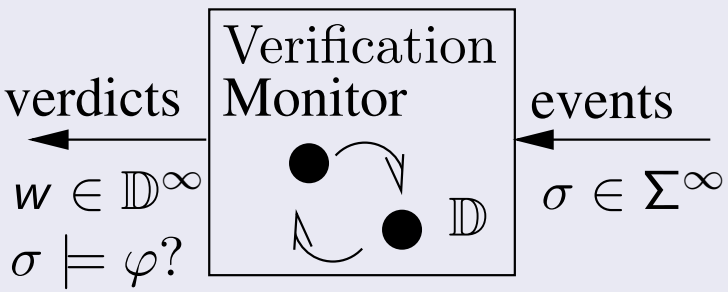
- A monitor observes the execution of a system (e.g., trace, log, messages).
- No system model.
- A correctness property φ.

Runtime verification and enforcement (monitors)

Runtime verification and enforcement:

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

Runtime verification



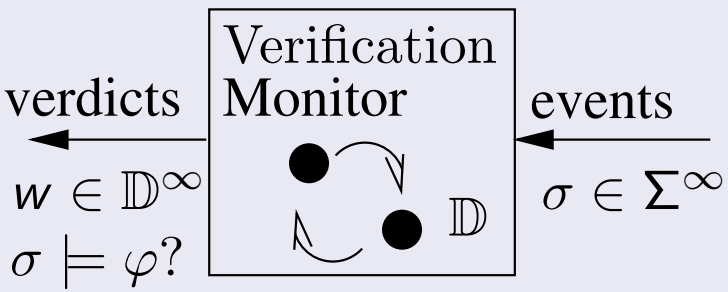
- Does the run satisfy the property?
- Input: stream of events.
- Output: stream of **verdicts**.

Runtime verification and enforcement (monitors)

Runtime verification and enforcement:

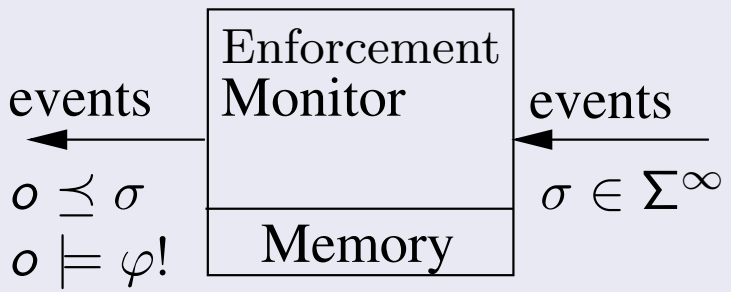
- A monitor observes the execution of a system (e.g., trace, log, messages).
- No system model.
- A correctness property φ.

Runtime verification



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

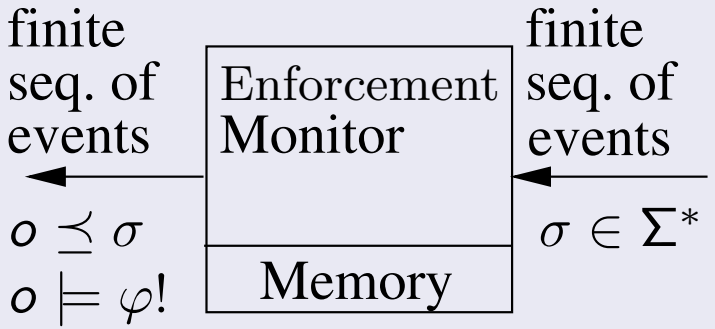
Runtime enforcement



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

Enforcement monitoring - **untimed case**

- Dedicated to a property φ.
- Possibly augmented with a **memorization mechanism**.



Enforcement mechanism (EM)

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

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

An EM behaves as a function $E : \Sigma^* \to \Sigma^*$.

Motivations for *timed* enforcement

Motivations for *timed* enforcement

Specifying the timing behavior

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

Motivations for *timed* enforcement

Specifying the timing behavior

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

- After action “a”, action “b” should occur

Motivations for *timed* enforcement

Specifying the timing behavior

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

- After action "a", action "b" should occur with a delay of at least 5 time units between them.

Motivations for *timed* enforcement

Specifying the timing behavior

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

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

Motivations for *timed* enforcement

Specifying the timing behavior

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

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

Many application domains

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

Related work on monitoring

Runtime Enforcement of **Untimed** properties

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

Related work on monitoring

Runtime Enforcement of **Untimed** properties

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

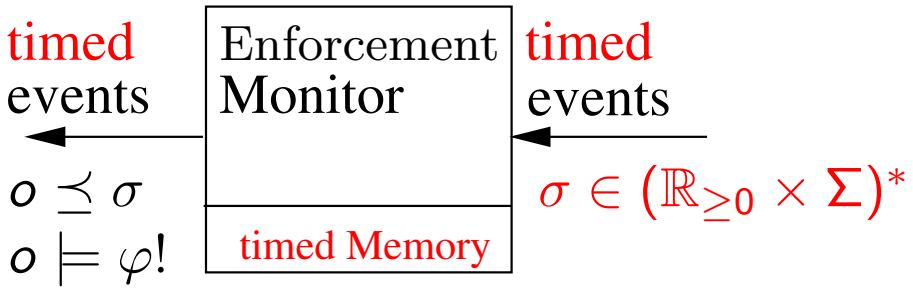
Runtime Verification of Timed properties

Efforts mainly to *verify* timed properties at runtime:

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

Problem tackled and Contributions

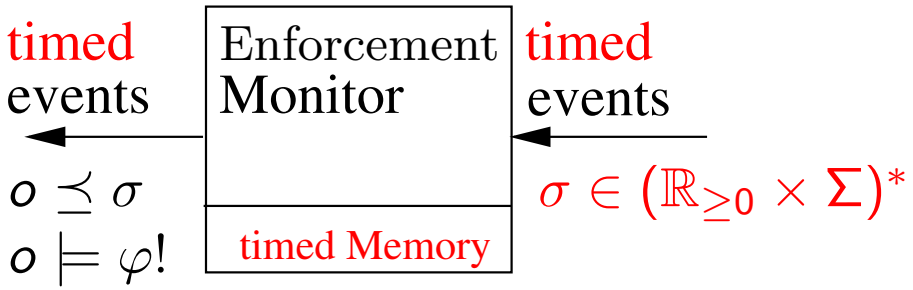
φ is a timed property



A formal framework for runtime enforcement of timed properties

Problem tackled and Contributions

φ is a timed property

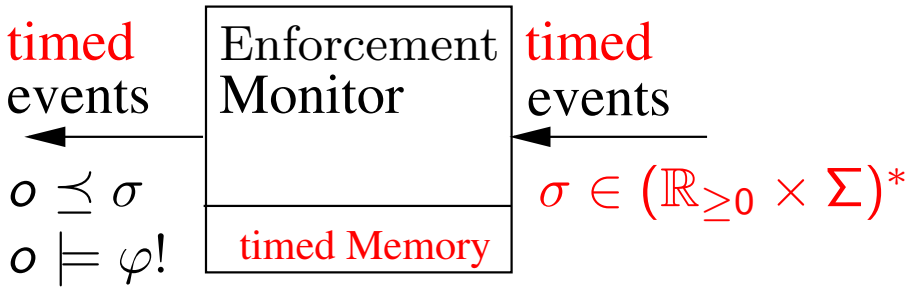


A formal framework for runtime enforcement of timed properties

- Any regular timed property φ as input.

Problem tackled and Contributions

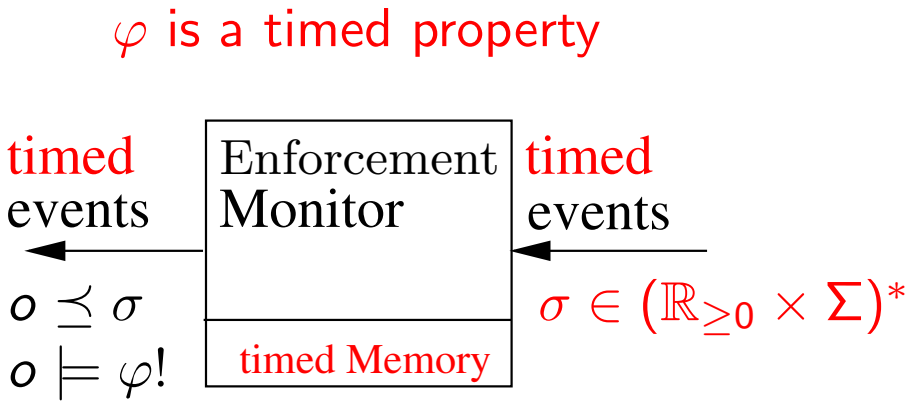
φ is a timed property



A formal framework for runtime enforcement of timed properties

- Any regular timed property φ as input.
- Enforcement mechanism adds additional delays between input actions in order to satisfy the property. – works as a "delayer"

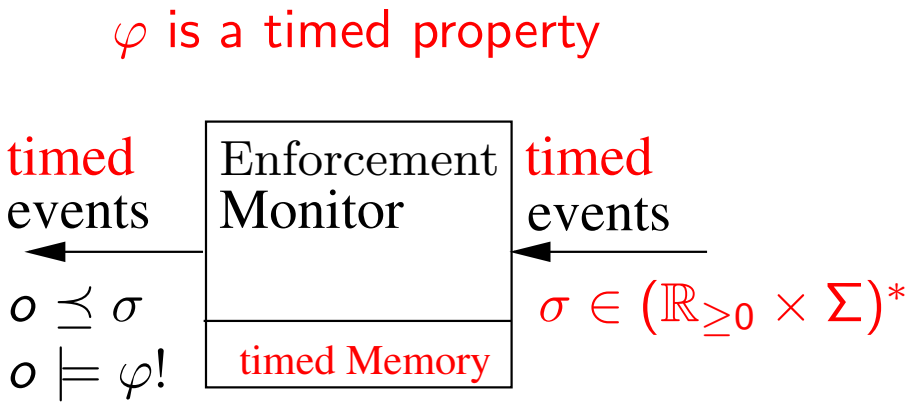
Problem tackled and Contributions



A formal framework for runtime enforcement of timed properties

- Any regular timed property φ as input.
- Enforcement mechanism adds additional delays between input actions in order to satisfy the property. – works as a "delayer"
- A general definition of mechanisms for regular properties.

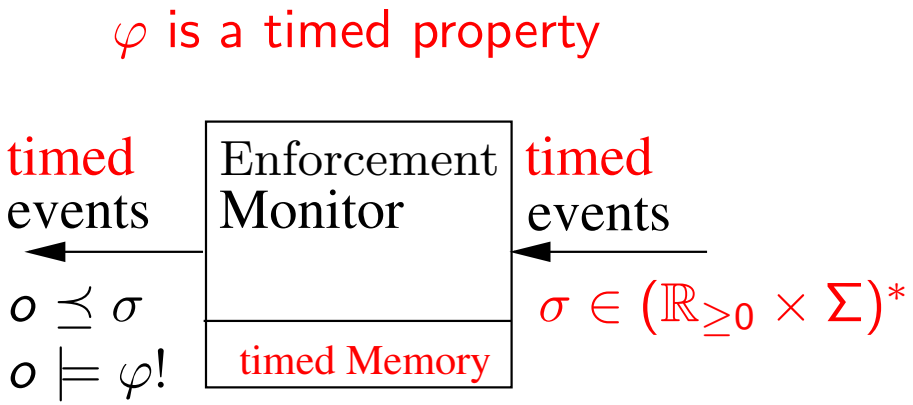
Problem tackled and Contributions



A formal framework for runtime enforcement of timed properties

- Any regular timed property φ as input.
- Enforcement mechanism adds additional delays between input actions in order to satisfy the property. – works as a "delayer"
- A general definition of mechanisms for regular properties.
- Optimizations for safety and co-safety properties.

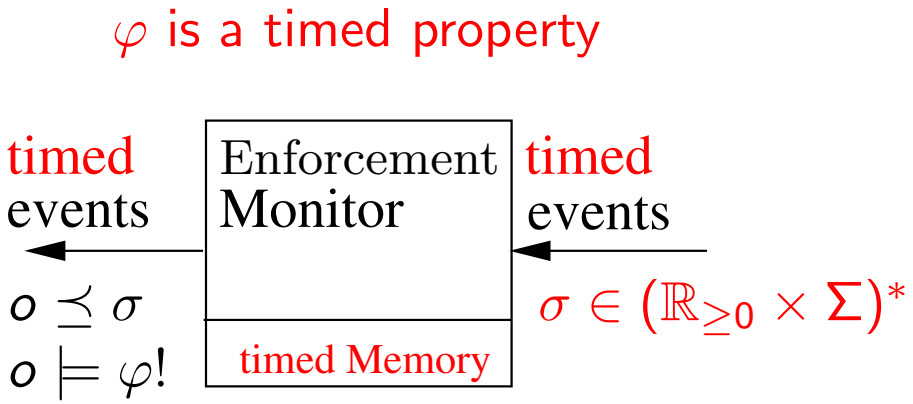
Problem tackled and Contributions



A formal framework for runtime enforcement of timed properties

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

Problem tackled and Contributions



A formal framework for runtime enforcement of timed properties

- Any regular timed property φ as input.
- Enforcement mechanism adds additional delays between input actions in order to satisfy the property. – works as a "delayer"
- A general definition of mechanisms for regular properties.
- Optimizations for safety and co-safety properties.
- Enforcement mechanisms at *several levels of abstraction* (facilitating the design and implementation of such mechanisms).
- Exhibiting a notion of *non-enforceable properties*.

Outline - Runt. Enforcement of Regular Timed Properties

Outline - Runt. Enforcement of Regular Timed Properties

Specifying timed properties

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

Specifying timed properties

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

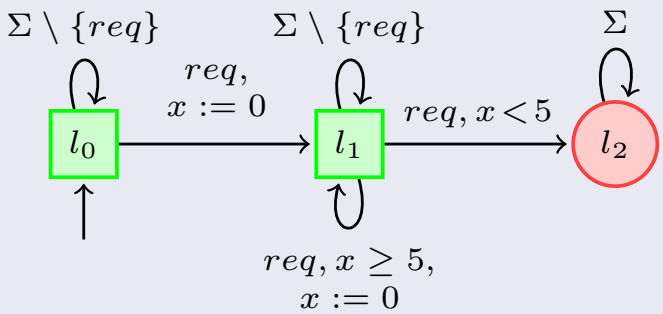
Safety, co-safety and response properties specified by TAs

Specifying timed properties

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

Safety, co-safety and response properties specified by TAs

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



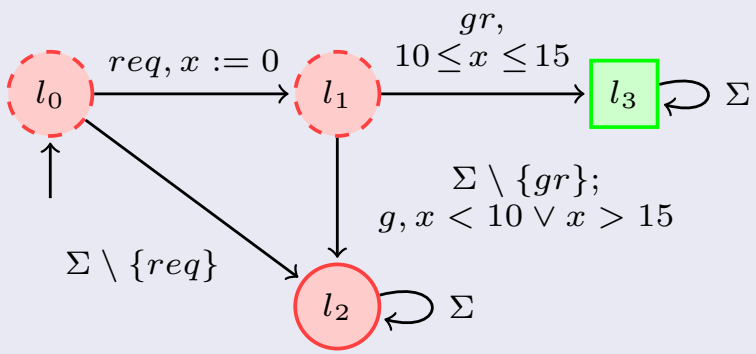
$\Sigma = \{req\}$
"A delay of 5 t.u. between any two requests."

Specifying timed properties

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

Safety, co-safety and response properties specified by TAs

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



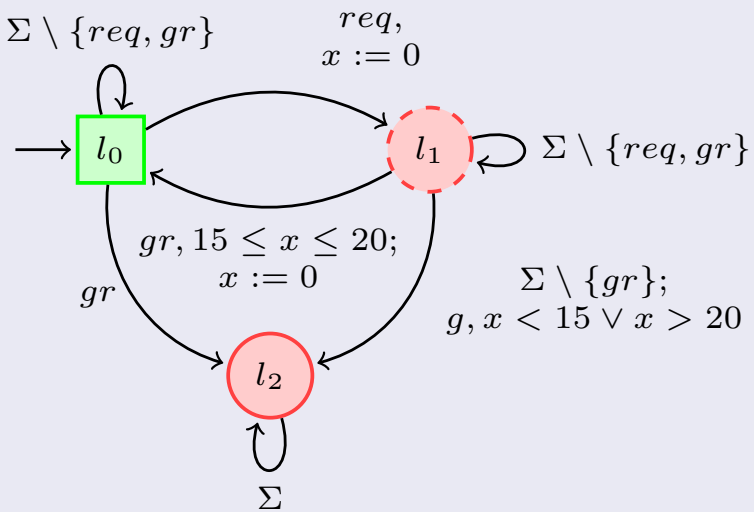
$\Sigma = \{req, gr\}$
"A request, and then a grant should arrive between 10 and 15 t.u."

Specifying timed properties

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

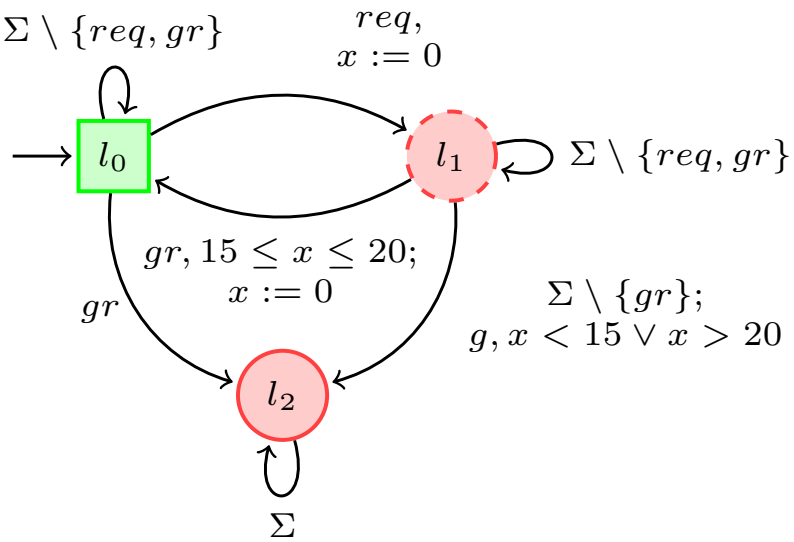
Safety, co-safety and response properties specified by TAs

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."

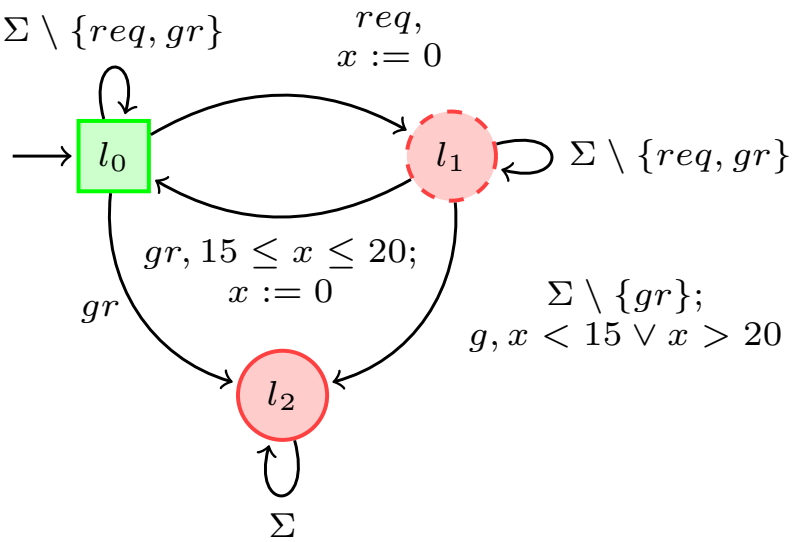
Example: response property



- $\Sigma = \{req, gr\}$
- $(3, req) \cdot (15, gr) \cdot (5, req) \cdot (19, gr)$

$\epsilon \models \varphi$.

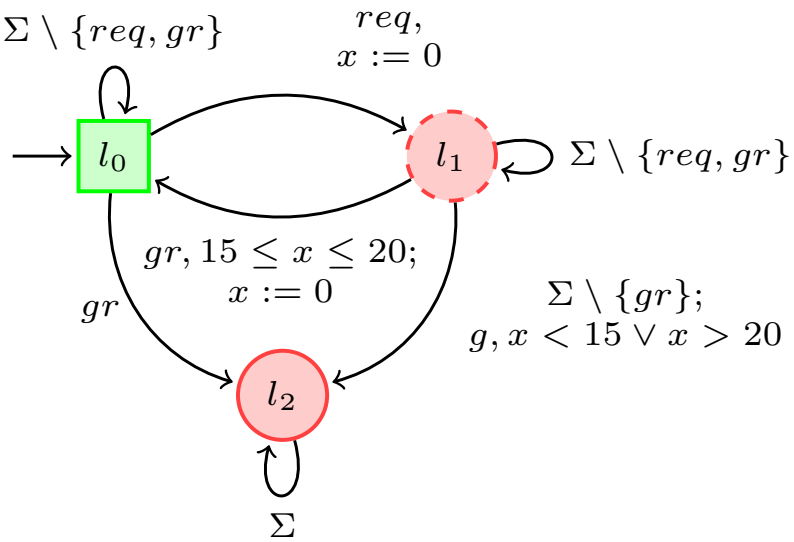
Example: response property



- $\Sigma = \{req, gr\}$
- $(3, req) \cdot (15, gr) \cdot (5, req) \cdot (19, gr)$

$\epsilon \models \varphi$.
$(3, req) \not\models \varphi$.

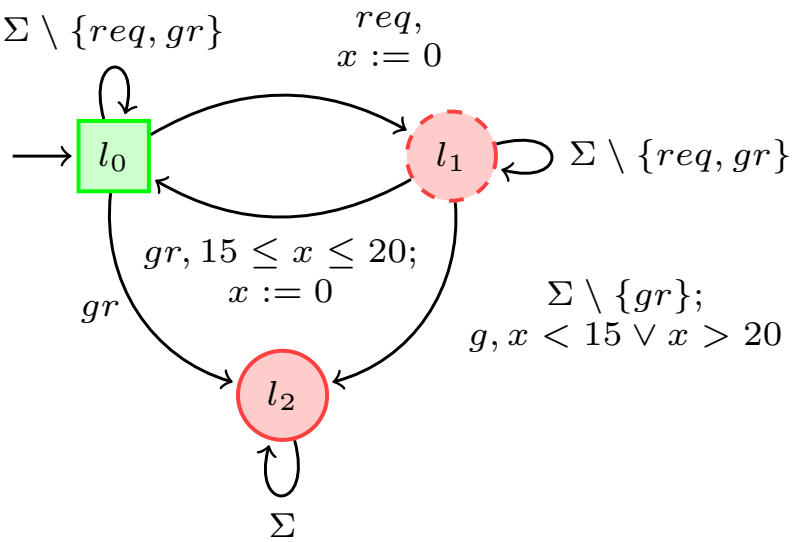
Example: response property



- $\Sigma = \{req, gr\}$
- $(3, req) \cdot (15, gr) \cdot (5, req) \cdot (19, gr)$

$\epsilon \models \varphi$.
$(3, req) \not\models \varphi$.
$(3, req) \cdot (15, gr) \models \varphi$.

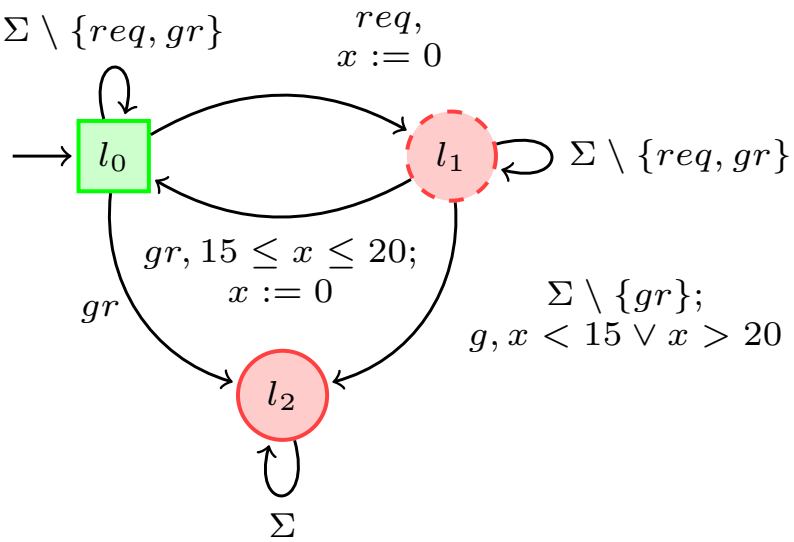
Example: response property



- $\Sigma = \{req, gr\}$
- $(3, req) \cdot (15, gr) \cdot (5, req) \cdot (19, gr)$

$\epsilon \models \varphi$.
$(3, req) \not\models \varphi$.
$(3, req) \cdot (15, gr) \models \varphi$.
$(3, req) \cdot (15, gr) \cdot (5, req) \not\models \varphi$.

Example: response property



- $\Sigma = \{req, gr\}$
- $(3, req) \cdot (15, gr) \cdot (5, req) \cdot (19, gr)$

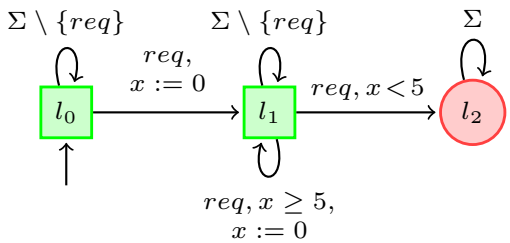
$\epsilon \models \varphi$.
$(3, req) \not\models \varphi$.
$(3, req) \cdot (15, gr) \models \varphi$.
$(3, req) \cdot (15, gr) \cdot (5, req) \not\models \varphi$.
$(3, req) \cdot (15, gr) \cdot (5, req) \cdot (19, gr) \models \varphi$.

Remark: response properties are neither prefix nor extension closed.

Major Challenges

Major challenges when (possibly) correcting an input sequence:

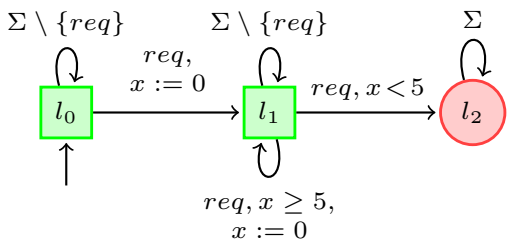
- safety properties: after each event, the decision is made (whether it can be corrected or not).

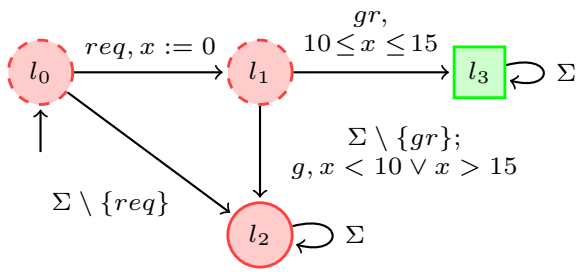


Major Challenges

Major challenges when (possibly) correcting an input sequence:

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

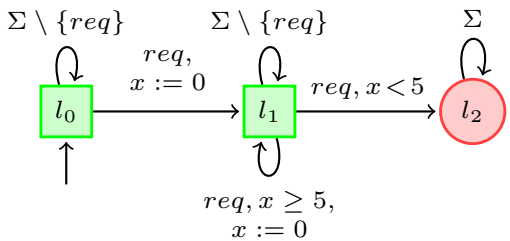


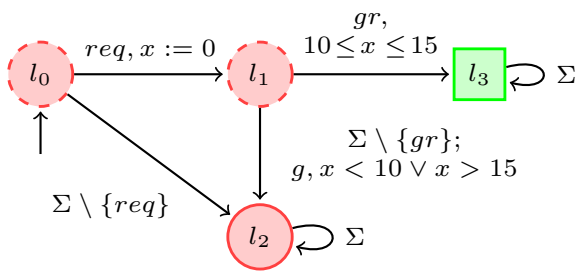


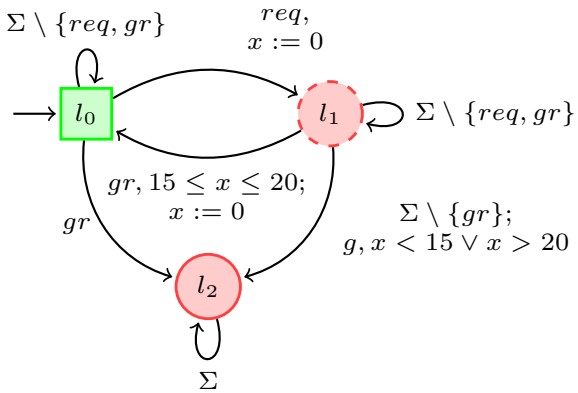
Major Challenges

Major challenges when (possibly) correcting an input sequence:

- safety properties: after each event, the decision is made (whether it can be corrected or not).
- co-safety properties: after each event, we check starting from the first event, whether the entire sequence can be corrected.
- response properties:
  - we cannot decide for each event soon after it is observed.
  - we do not check/correct from the first event since we want to correct and output chunk of sequences as soon as possible.



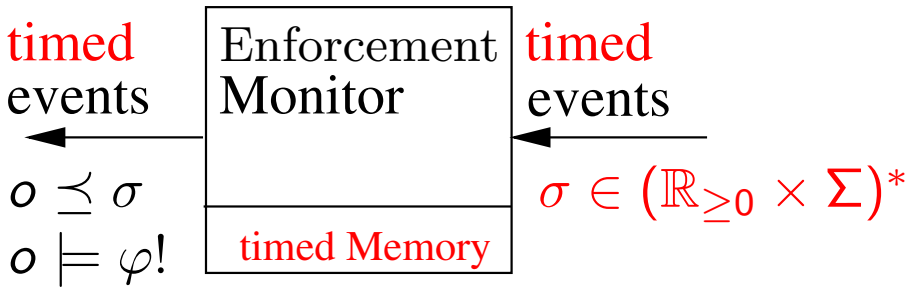




Outline - Runt. Enforcement of Regular Timed Properties

Summary of the approach

Given some timed property φ:

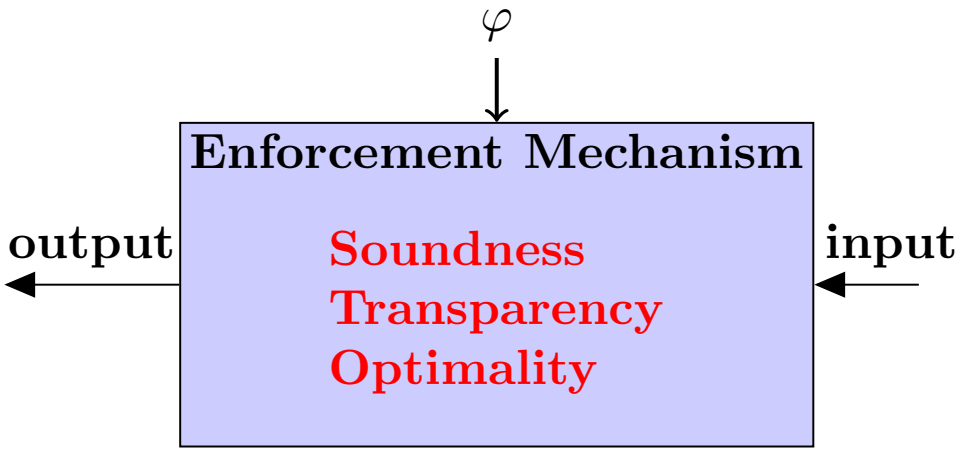


What can an enforcement mechanism do?

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

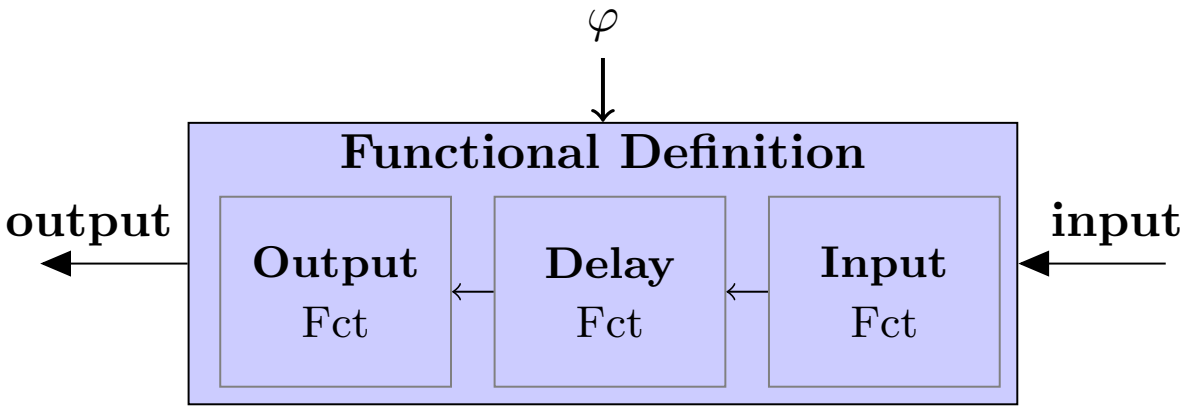
$\hookrightarrow$ the enforcement monitor is a "delayer".

Summary of the approach



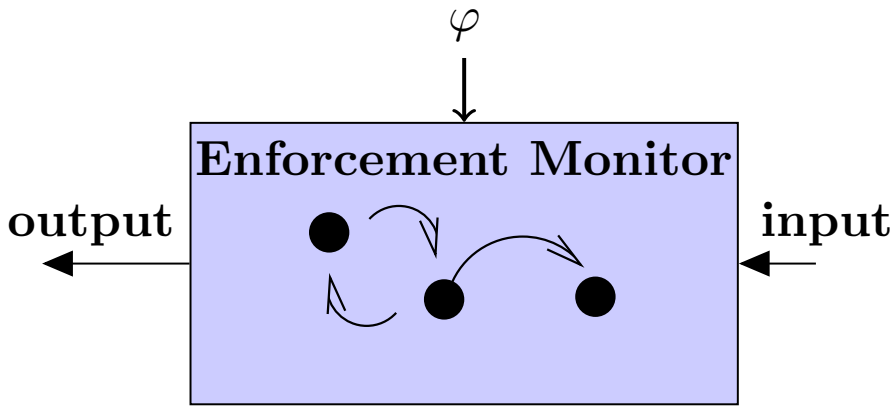
- **Requirements** for any enforcement mechanism for φ
- **Functional definition** (satisfies the requirements):
  - description of the input/output behavior ;
  - composition of 3 functions: process input, computing the delayed timed word, and process output,
- **Enforcement monitor**:
  - description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

Summary of the approach



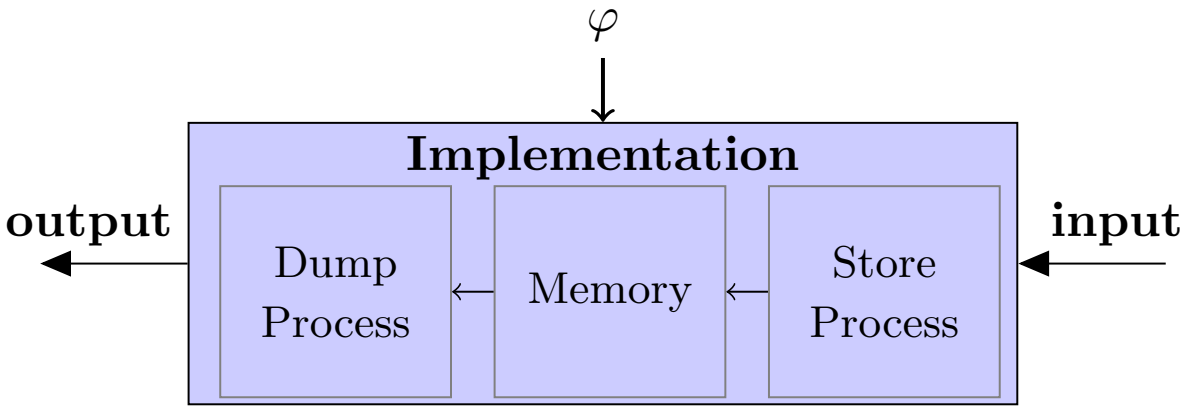
- **Requirements** for any enforcement mechanism for φ
- **Functional definition** (satisfies the requirements):
  - description of the input/output behavior ;
  - composition of 3 functions: process input, computing the delayed timed word, and process output,
- **Enforcement monitor**:
  - description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

Summary of the approach



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

Summary of the approach



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

Outline - Runt. Enforcement of Regular Timed Properties

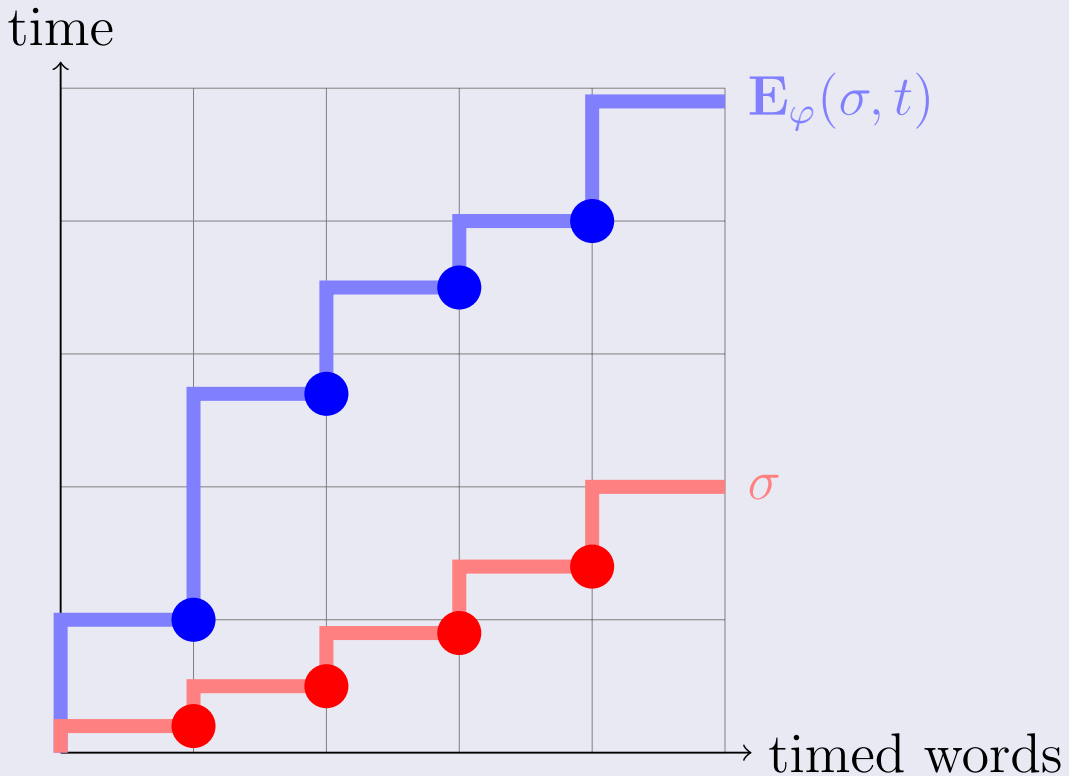
Requirements on an Enforcement Mechanism (1)

Specified on an enforcement function for φ

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

Requirements on an Enforcement Mechanism (1)

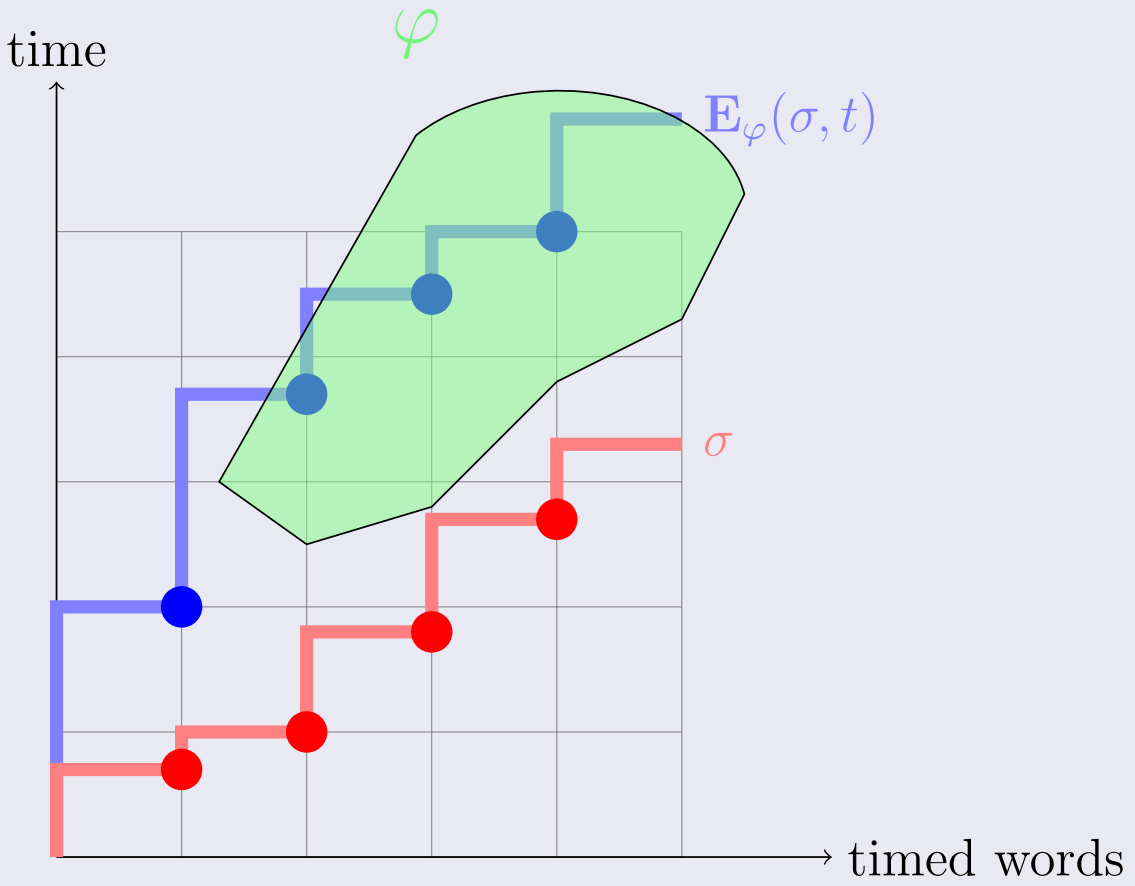
The input and output of the mechanism are timed words



Requirements on an Enforcement Mechanism (2)

Soundness: the output is correct

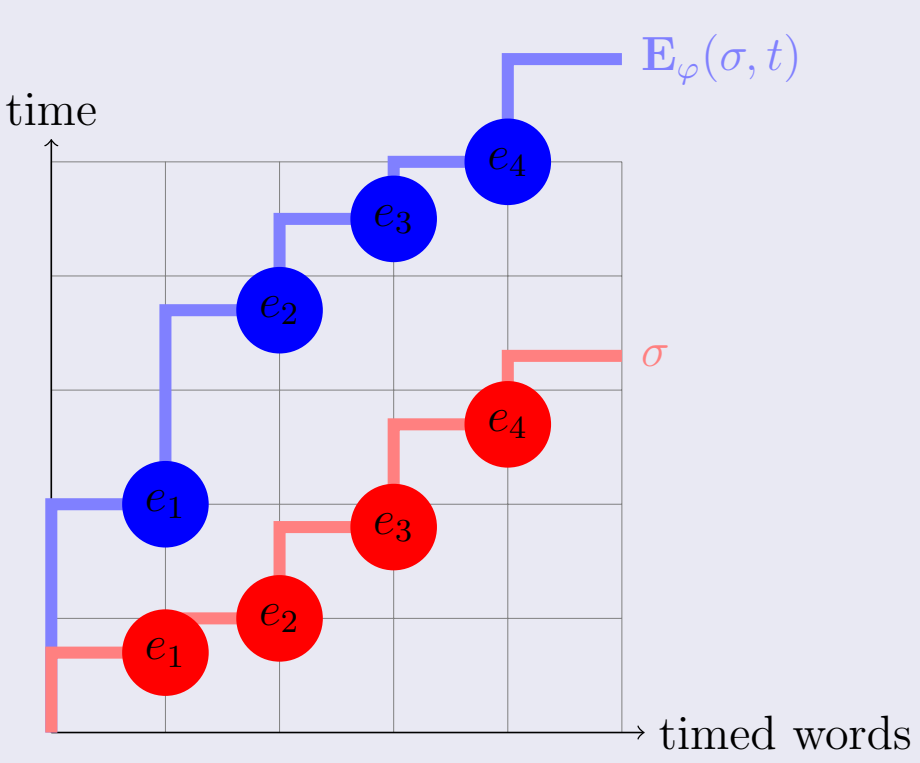
Snd $E_\varphi(\sigma, t) \neq \epsilon \implies \exists t' \geq t : E_\varphi(\sigma, t') \models \varphi.$

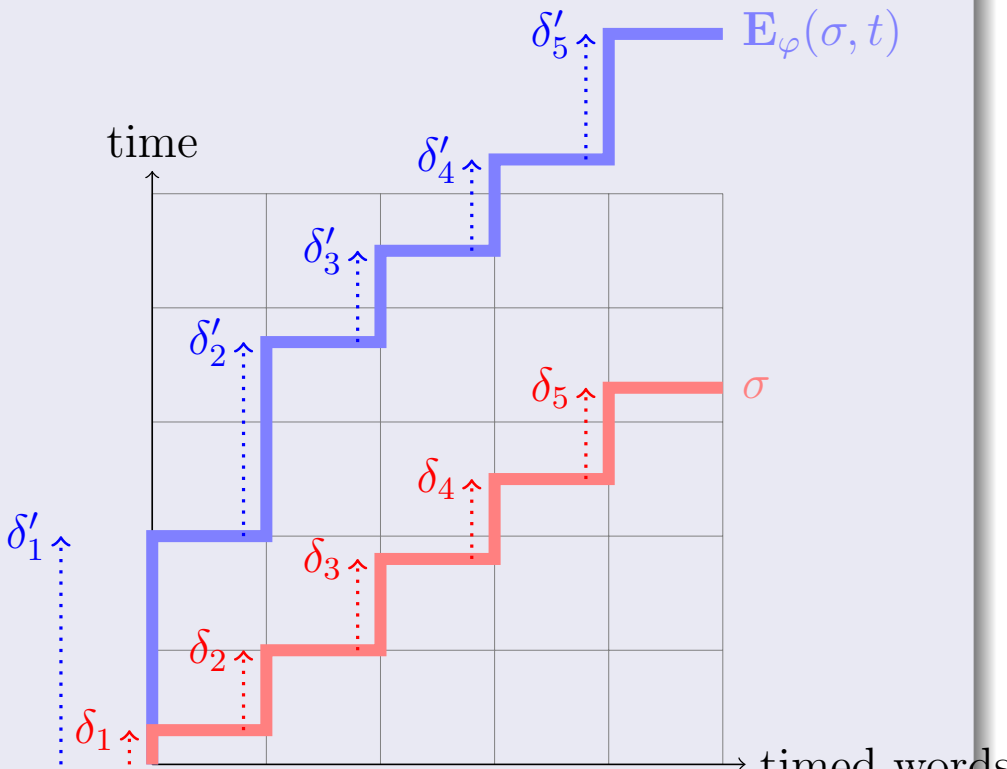


Requirements on an Enforcement Mechanism (1)

Transparency: events are preserved and delayed

Tr $E_\varphi(\sigma, t) \preccurlyeq_d \mathrm{obs}(\sigma, t)$, where $\sigma' \preccurlyeq_d \sigma$ means



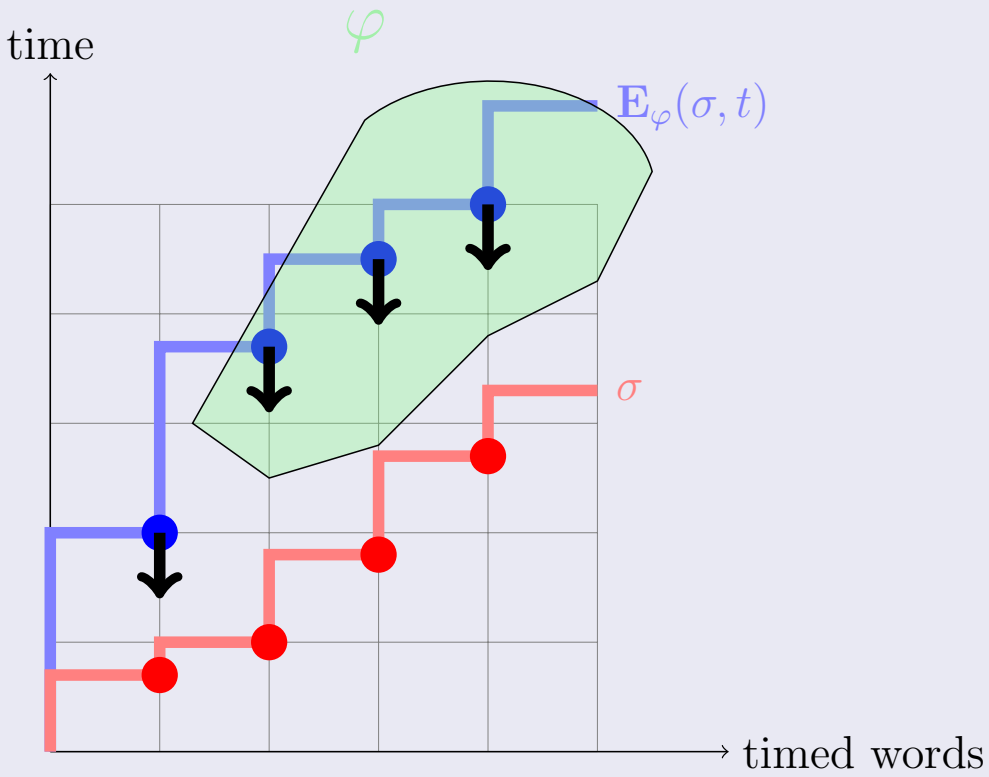


Requirements on an Enforcement Mechanism (2)

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

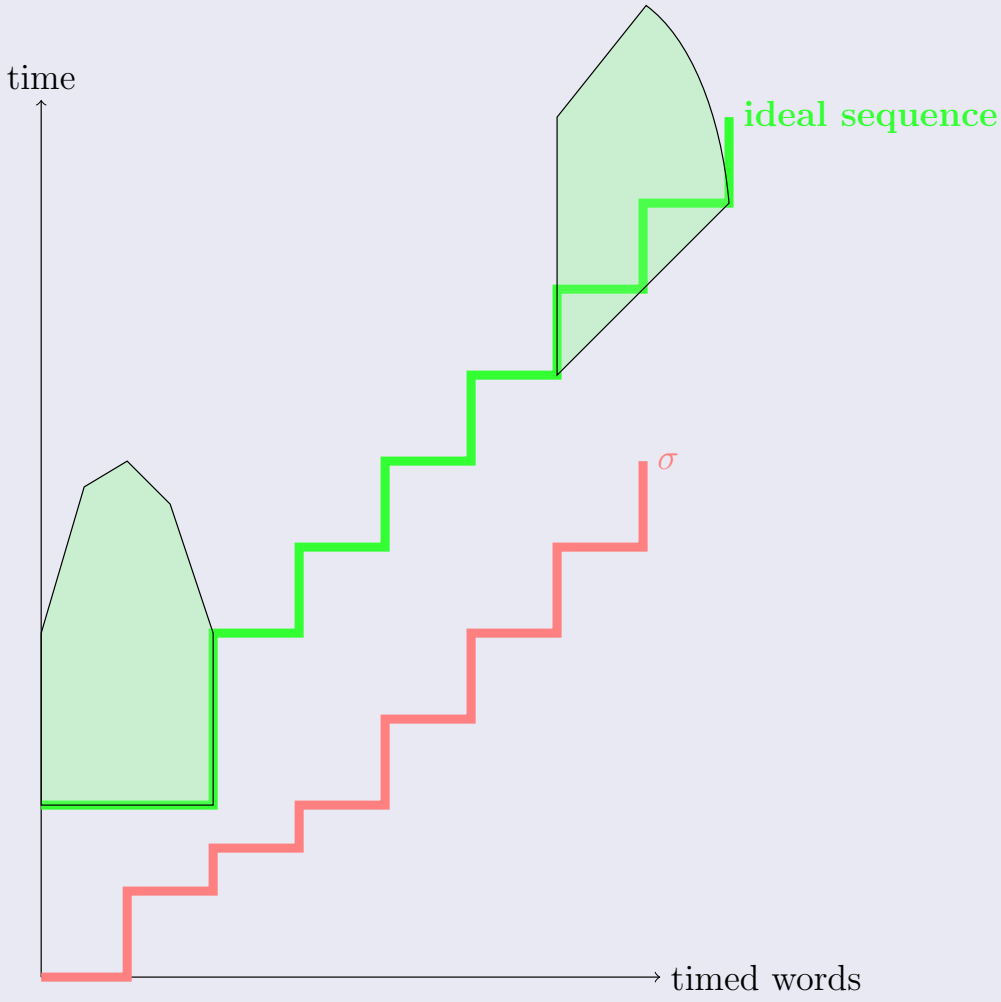
Requirements on an Enforcement Mechanism (2)

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



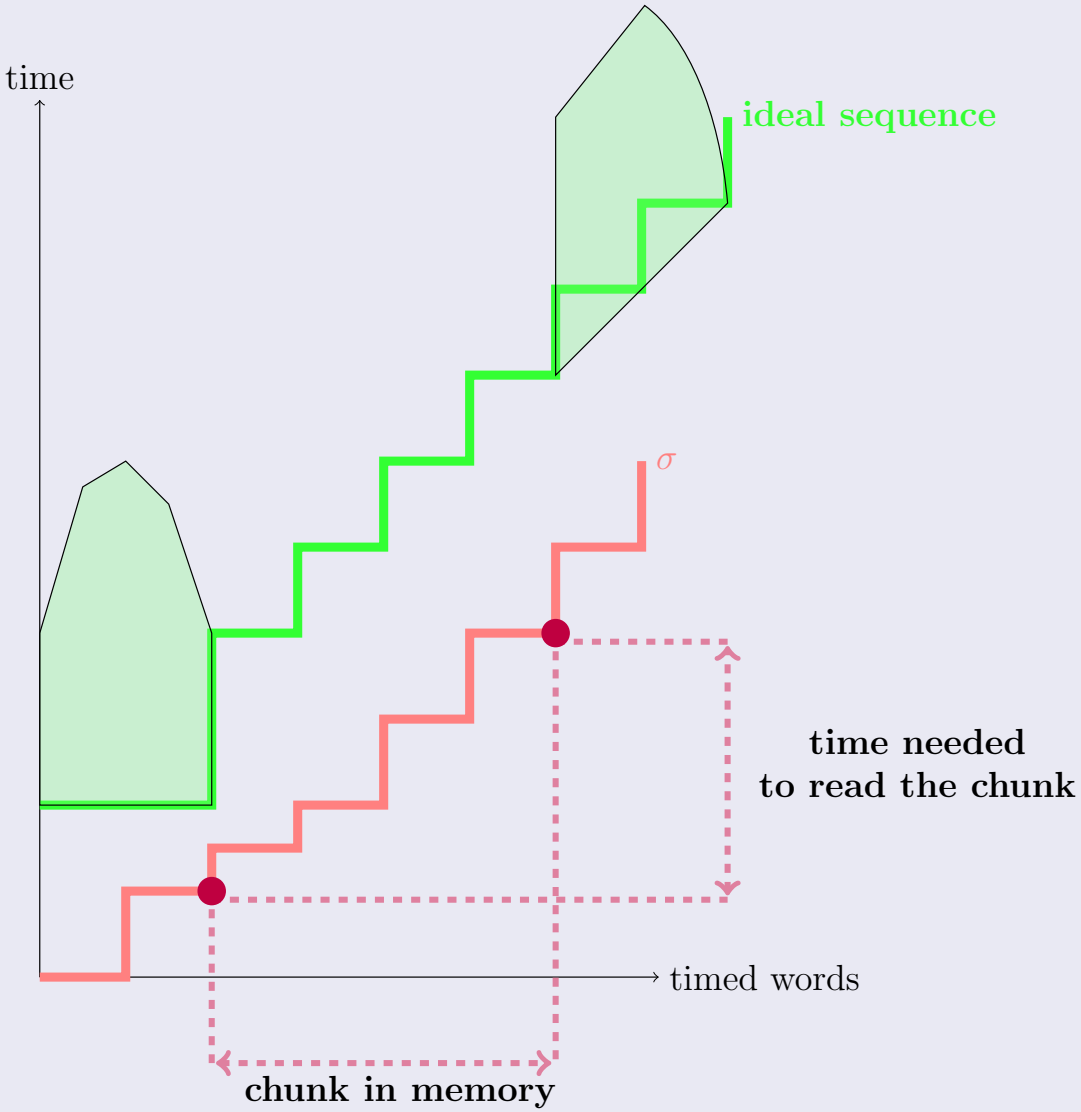
Requirements on an Enforcement Mechanism (2)

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



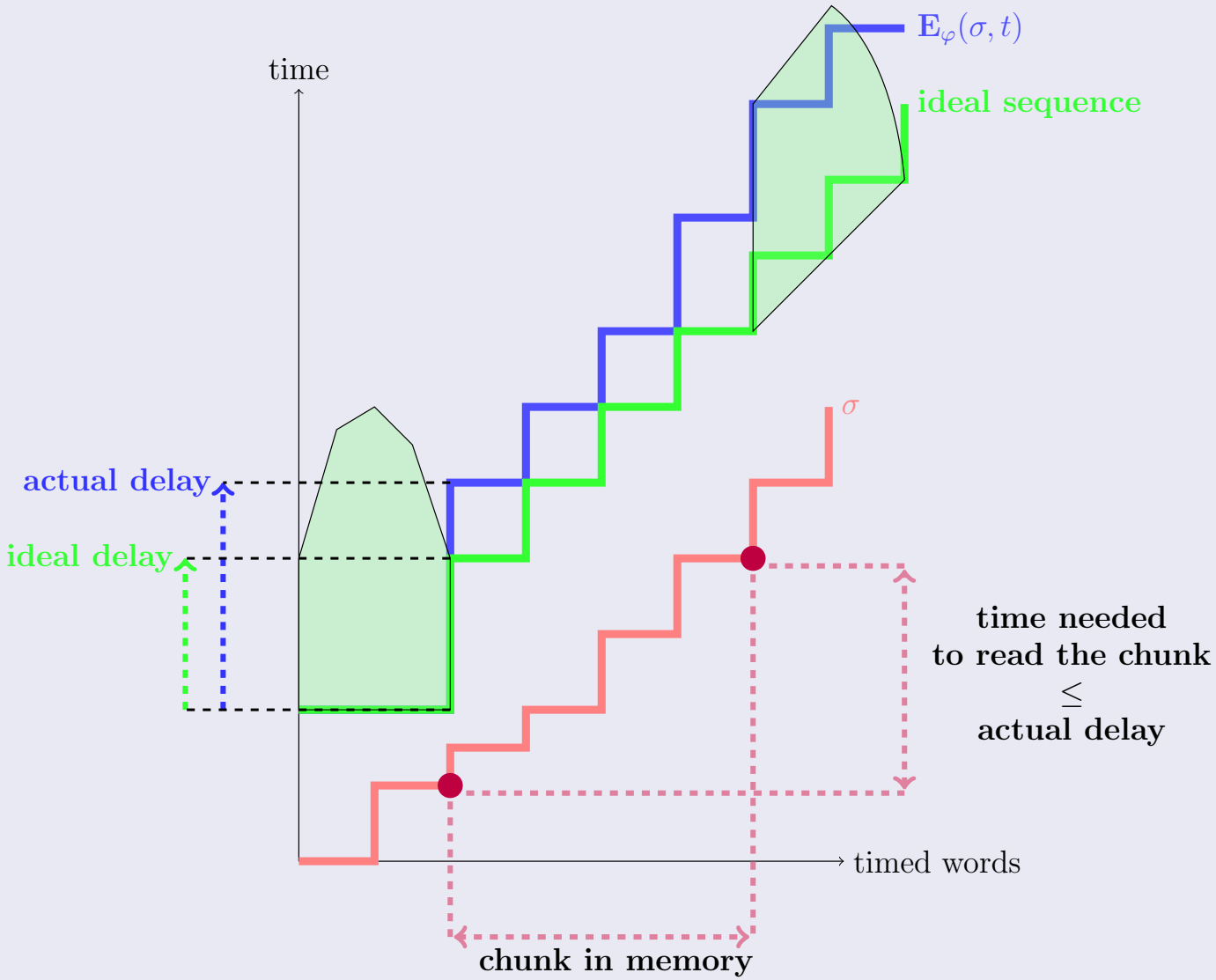
Requirements on an Enforcement Mechanism (2)

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



Requirements on an Enforcement Mechanism (2)

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

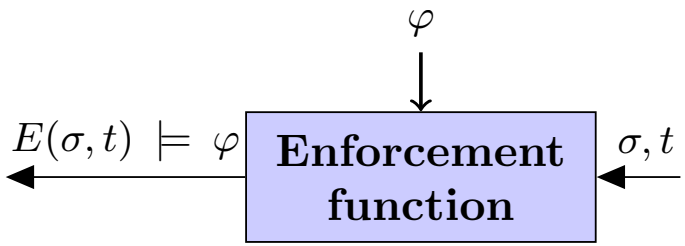


Outline - Runt. Enforcement of Regular Timed Properties

Functional definition (1)

The functional definition describes the mechanism as a function

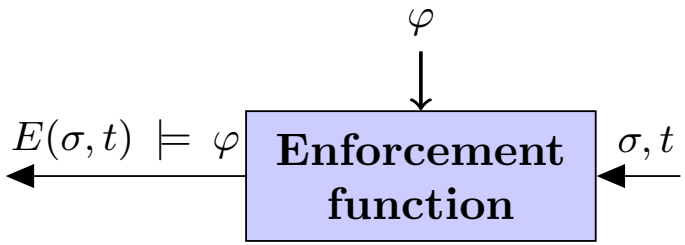
$$E_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \to (\mathbb{R}_{\geq 0} \times \Sigma)^*.$$

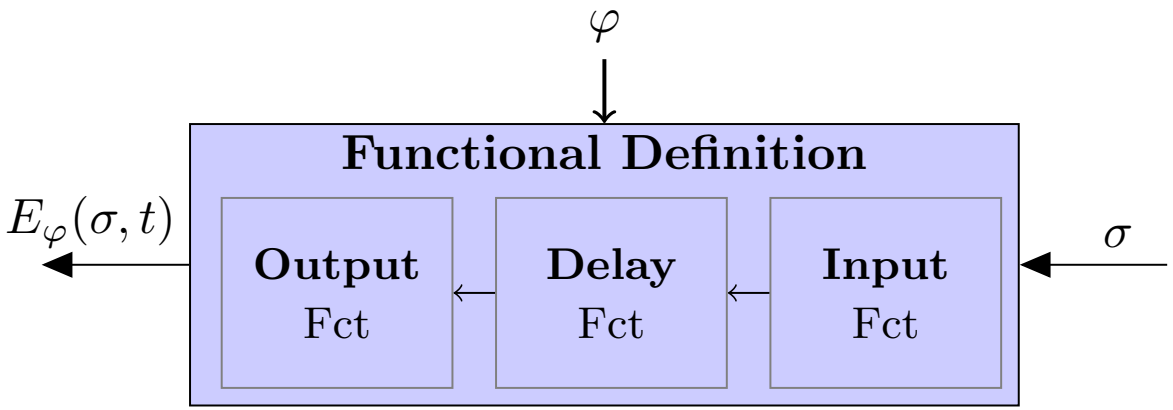


Functional definition (1)

The functional definition describes the mechanism as a function

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

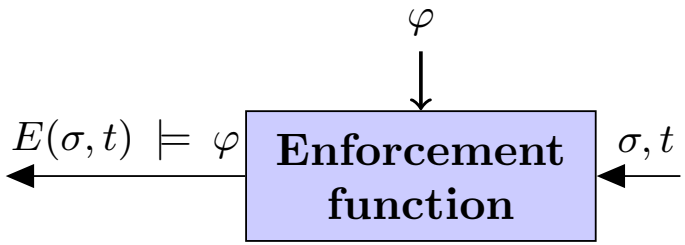


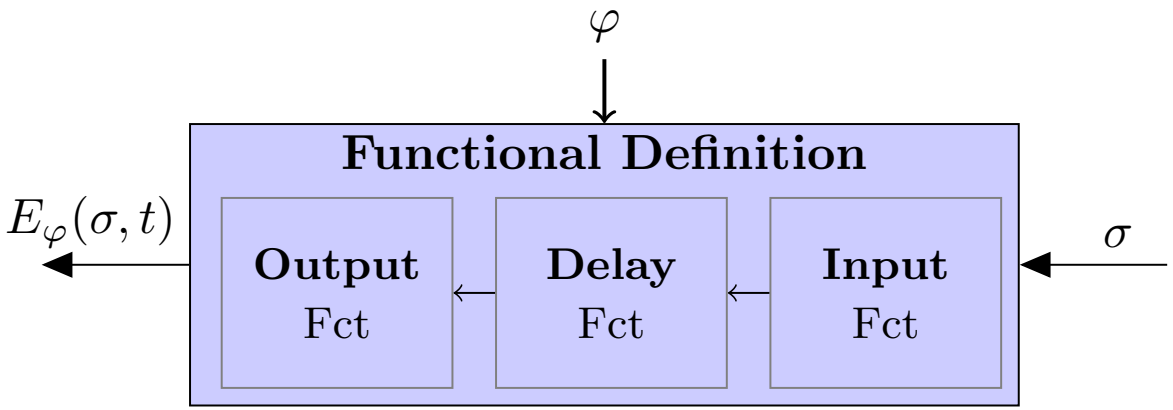


Functional definition (1)

The functional definition describes the mechanism as a function

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





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 (2)

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

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

Functional definition (2)

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

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

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

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

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

Functional definition (2)

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

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

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

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

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

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

Functional definition (2)

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

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

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

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

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

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

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

Functional definition (2)

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

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

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

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

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

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

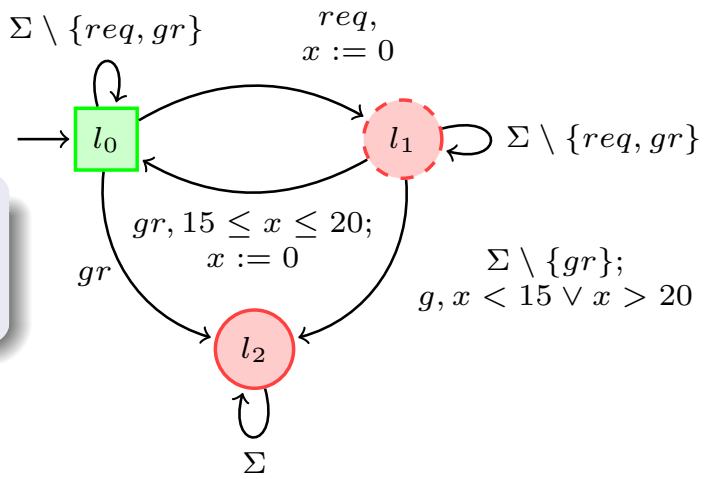
Suppose $(\sigma_s, \sigma_c) = \mathrm{store}_\varphi(\sigma)$

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

where K is the set of possible corrected factors of σ between positions $|\sigma_s|$ and $|\sigma_c| + 1$ with a delay for the first event greater than $\mathrm{time}(\sigma_c \cdot (\delta, a))$.
(cf. details in the paper)

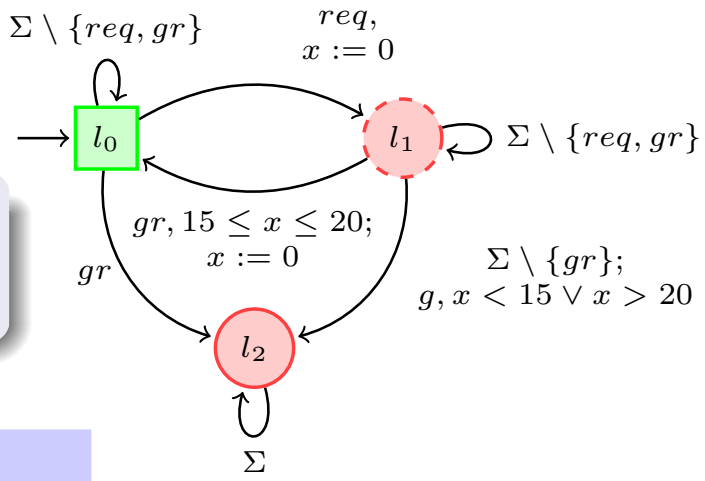
Functional definition: Example

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



Functional definition: Example

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

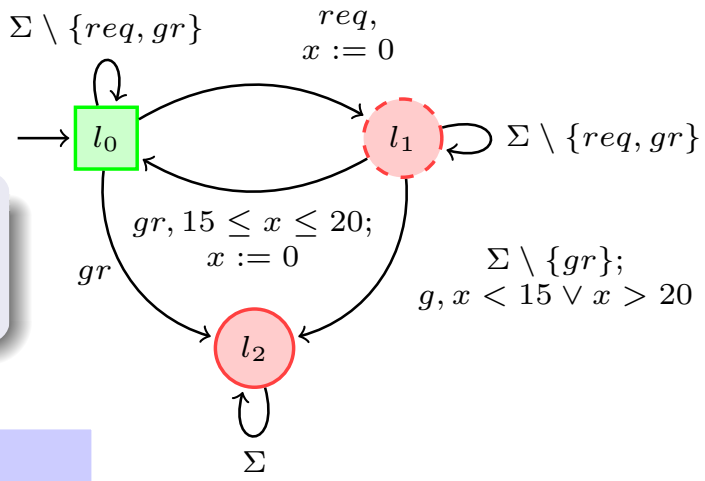


$t \in [0, 3[$

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

Functional definition: Example

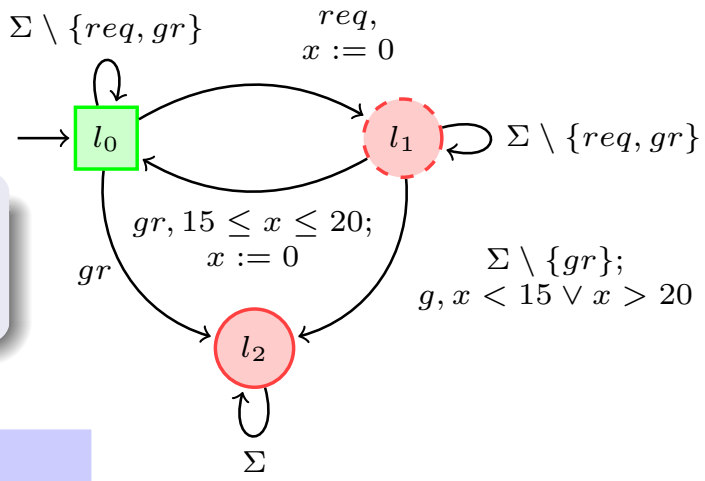
$\Sigma = \{r, g\}$.
$\sigma = (3, req) \cdot (10, gr) \cdot (3, req) \cdot (5, req)$.



$t \in [0, 3[$	$\text{obs}(\sigma, t) = \epsilon$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon)$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$
$t \in [3, 13[$	$\text{obs}(\sigma, t) = (3, req)$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req))$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$

Functional definition: Example

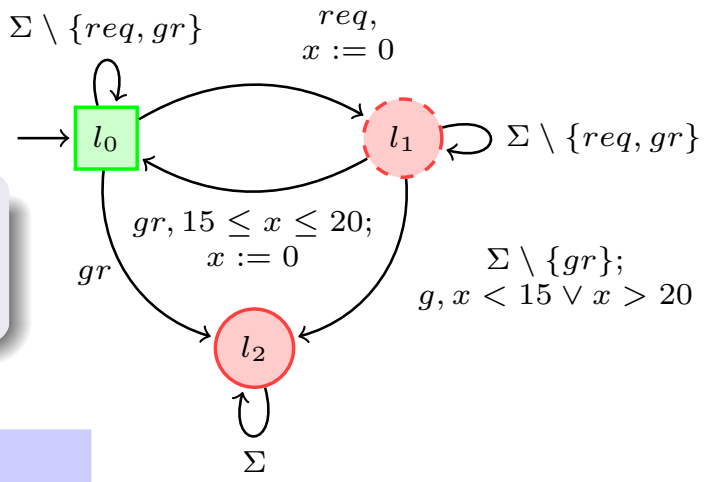
$\Sigma = \{r, g\}$.
$\sigma = (3, req) \cdot (10, gr) \cdot (3, req) \cdot (5, req)$.



$t \in [0, 3[$	$\text{obs}(\sigma, t) = \epsilon$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon)$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$
$t \in [3, 13[$	$\text{obs}(\sigma, t) = (3, req)$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req))$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$
$t \in [13, 16[$	$\text{obs}(\sigma, t) = (3, req) \cdot (10, gr)$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = ((13, req) \cdot (15, gr), \epsilon)$ $E_\varphi(\sigma, t) = \text{obs}((13, req) \cdot (15, gr), t)$

Functional definition: Example

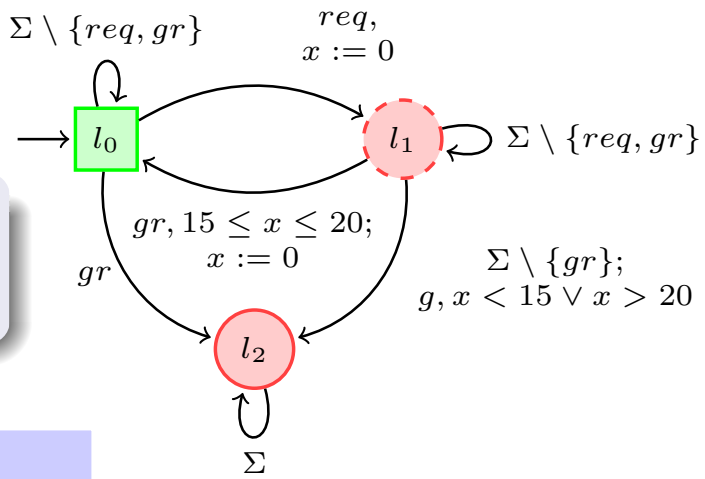
$\Sigma = \{r, g\}$.
$\sigma = (3, req) \cdot (10, gr) \cdot (3, req) \cdot (5, req)$.



$t \in [0, 3[$	$\text{obs}(\sigma, t) = \epsilon$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon)$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$
$t \in [3, 13[$	$\text{obs}(\sigma, t) = (3, req)$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req))$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$
$t \in [13, 16[$	$\text{obs}(\sigma, t) = (3, req) \cdot (10, gr)$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = ((13, req) \cdot (15, gr), \epsilon)$ $E_\varphi(\sigma, t) = \text{obs}((13, req) \cdot (15, gr), t)$
$t \in [16, 21[$	$\text{obs}(\sigma, t) = (3, req) \cdot (10, gr) \cdot (3, req)$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = ((13, req) \cdot (15, gr), (3, req))$ $E_\varphi(\sigma, t) = \text{obs}((13, req) \cdot (15, gr), t)$

Functional definition: Example

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



$t \in [0, 3[$	$\text{obs}(\sigma, t) = \epsilon$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon)$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$
$t \in [3, 13[$	$\text{obs}(\sigma, t) = (3, req)$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req))$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$
$t \in [13, 16[$	$\text{obs}(\sigma, t) = (3, req) \cdot (10, gr)$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = ((13, req) \cdot (15, gr), \epsilon)$ $E_\varphi(\sigma, t) = \text{obs}((13, req) \cdot (15, gr), t)$
$t \in [16, 21[$	$\text{obs}(\sigma, t) = (3, req) \cdot (10, gr) \cdot (3, req)$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = ((13, req) \cdot (15, gr), (3, req))$ $E_\varphi(\sigma, t) = \text{obs}((13, req) \cdot (15, gr), t)$
$t \in [21, \infty]$	$\text{obs}(\sigma, t) = (3, req) \cdot (10, gr) \cdot (3, req) \cdot (5, req)$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = ((13, req) \cdot (15, gr), (3, req) \cdot (5, req))$ $E_\varphi(\sigma, t) = \text{obs}((13, req) \cdot (15, gr), t)$

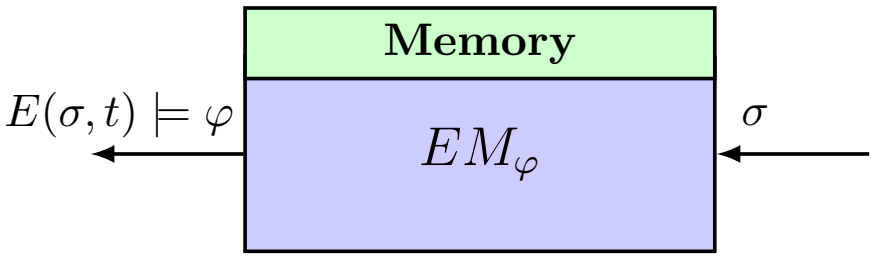
The enforcement function satisfies the requirements

Proposition: Enforcement function vs requirements

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

Outline - Runt. Enforcement of Regular Timed Properties

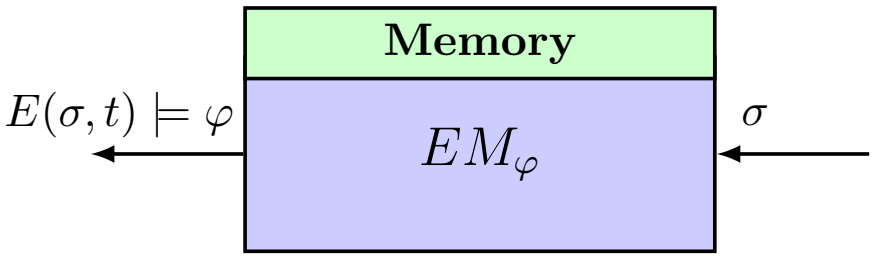
Enforcement monitor



A *rule-based transition system*:

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

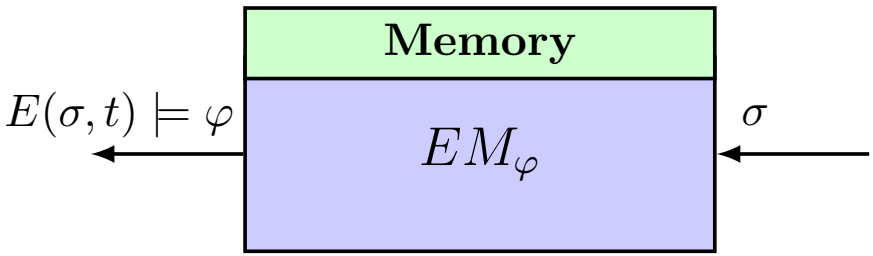
Enforcement monitor



A *rule-based transition system*:

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

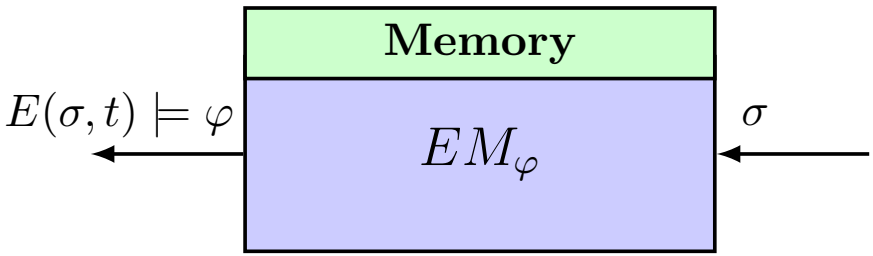
Enforcement monitor



A *rule-based transition system*:

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

Enforcement monitor



A *rule-based transition system*:

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

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

Enforcement monitor: operations

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

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

Enforcement monitor: operations

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

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

2. *store-φ*

- when a new event is received and the new event *can make the property satisfied* by delaying
- updates "good" memory and dump clock

Enforcement monitor: operations

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

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

2. *store-φ*

- when a new event is received and the new event *can make the property satisfied* by delaying
- updates "good" memory and dump clock

3. *dump*

- when an event in the good memory can be released
- updates "good" memory and dump clock

Enforcement monitor: operations

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

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

2. *store-φ*

- when a new event is received and the new event *can make the property satisfied* by delaying
- updates "good" memory and dump clock

3. *dump*

- when an event in the good memory can be released
- updates "good" memory and dump clock

4. *idle*

- when no other rule can
- updates dump and store clocks

Enforcement Monitor: correctness

Implementation relation between Enforcement Monitor and Enforcement Function

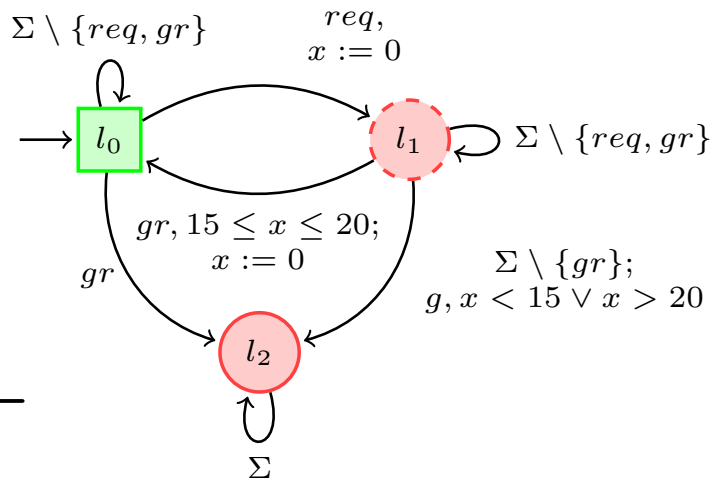
Given some property φ, at any time t, the input/output behavior of the synthesized enforcement monitor is the same as one of the corresponding enforcement function.

Corollary

Enforcement Monitors respect soundness, transparency, and optimality.

Enforcement Monitor: example

$t = 0$ - Executed operation: none

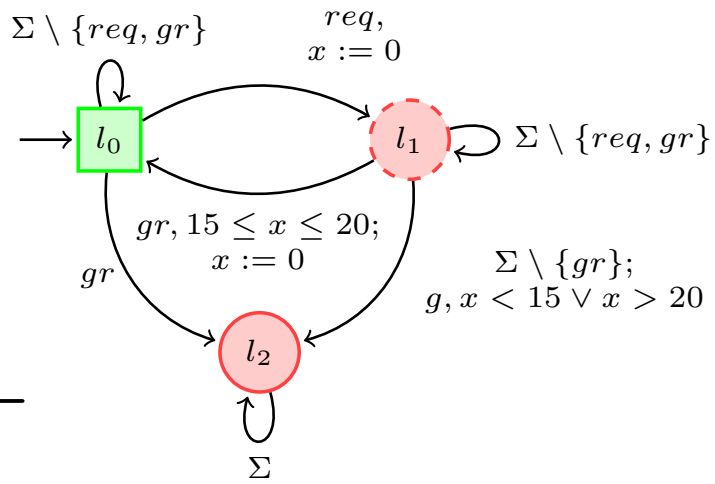


$\epsilon \longleftarrow$ $\longleftarrow (3, r) \cdot (10, g) \cdot (3, r) \cdot (5, r)$

- Good Memory: ϵ
- Bad Memory: ϵ
- State: $(l_0, 0)$

Enforcement Monitor: example

$t = 3$ - Executed operation: $\mathrm{idle}(3)$

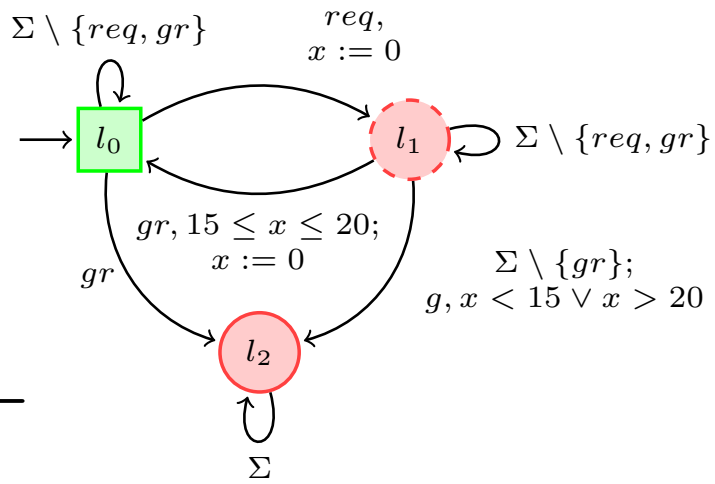


$\epsilon \leftarrow \quad \leftarrow (3, r) \cdot (10, g) \cdot (3, r) \cdot (5, r)$

- Good Memory: ϵ
- Bad Memory: ϵ
- State: $(l_0, 3)$

Enforcement Monitor: example

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

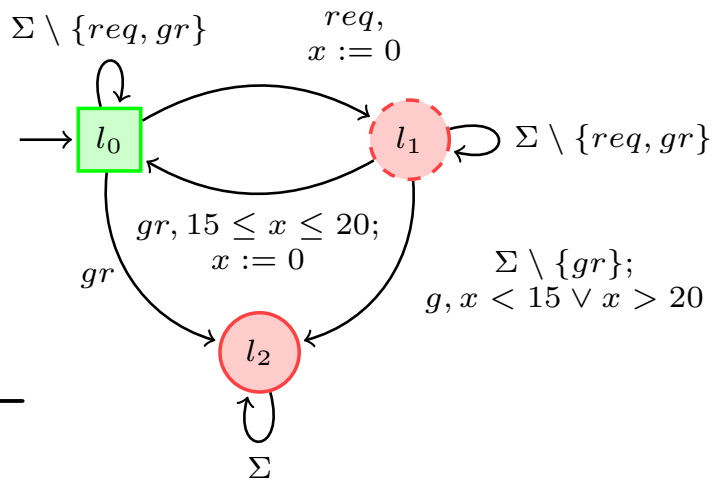


$\epsilon \longleftarrow \longleftarrow (10, g) \cdot (3, r) \cdot (5, r)$

- Good Memory: ϵ
- Bad Memory: $(3, r)$
- State: $(l_0, 0)$

Enforcement Monitor: example

$t = 13$ - Executed operation: $\mathrm{idle}(10)$

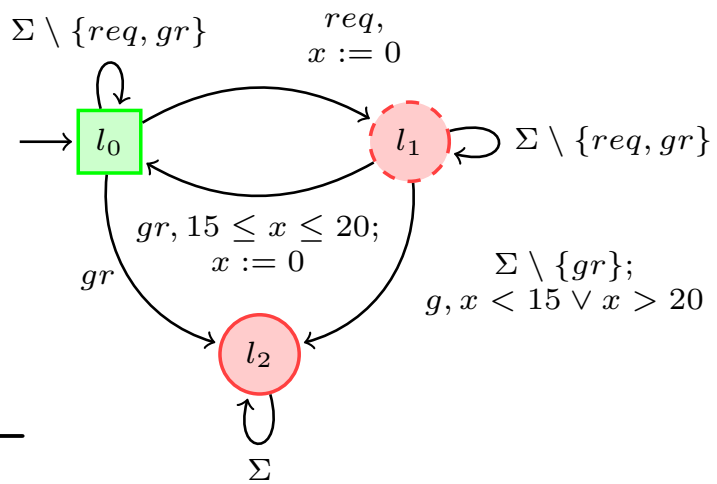


$\epsilon \longleftarrow \longleftarrow (10, g) \cdot (3, r) \cdot (5, r)$

- Good Memory: ϵ
- Bad Memory: $(3, r)$
- State: $(l_0, 0)$

Enforcement Monitor: example

$t = 13$ - Executed operation: *store-*φ

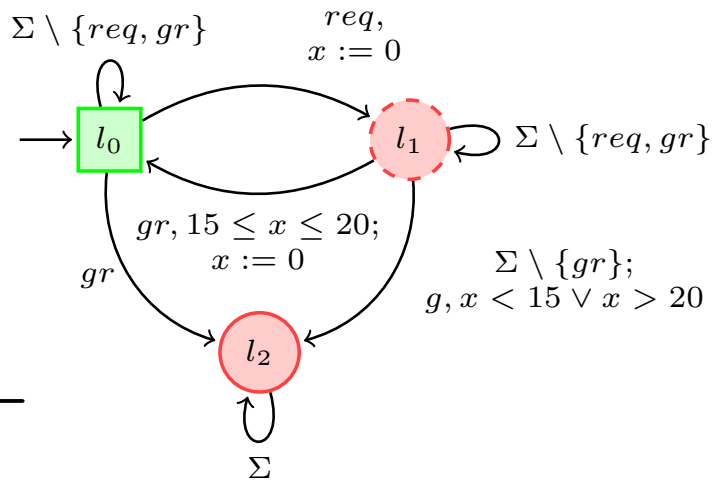


$\epsilon \leftarrow \leftarrow (3, r) \cdot (5, r)$

- Good Memory: $(13, r) \cdot (15, g)$
- Bad Memory: ϵ
- State: $(l_0, 0)$

Enforcement Monitor: example

$t = 13$ - Executed operation: *dump*

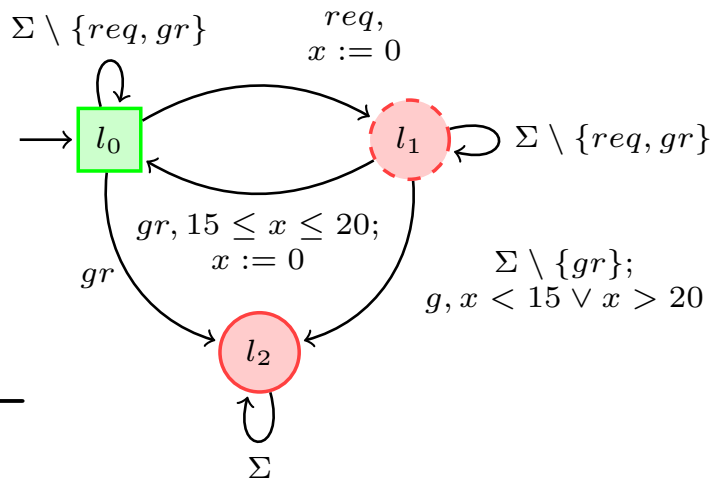


$(13, r) \leftarrow \leftarrow (3, r) \cdot (5, r)$

- Good Memory: $(15, g)$
- Bad Memory: ϵ
- State: $(l_0, 15)$

Enforcement Monitor: example

$t = 16$ - Executed operation: $\mathrm{idle}(3)$

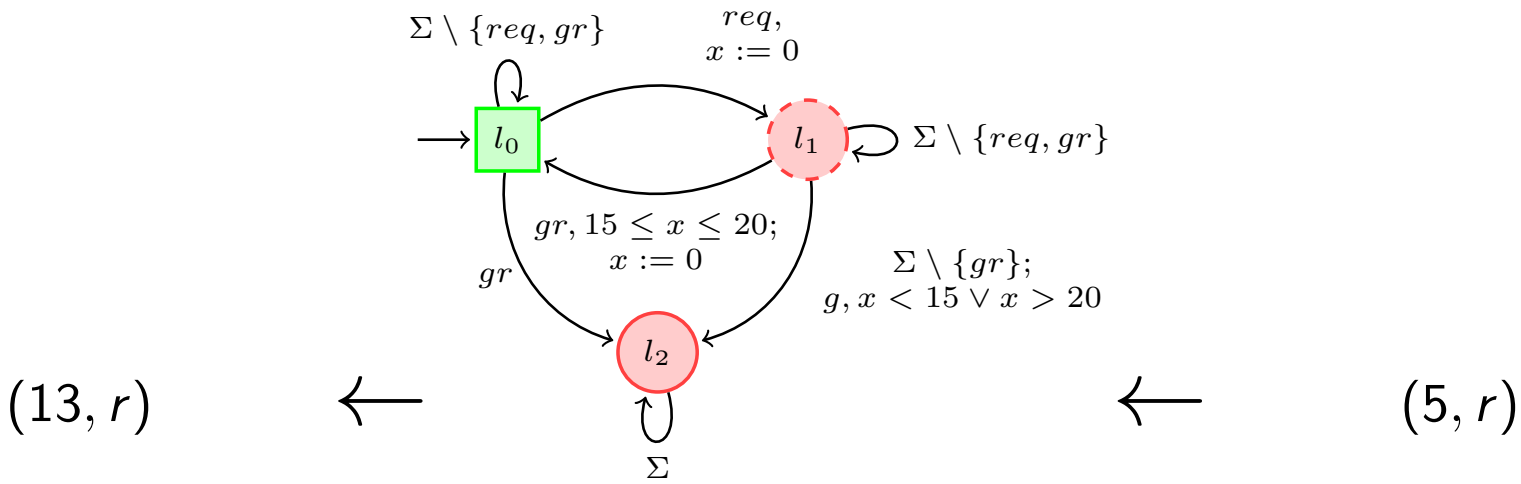


$(13, r) \leftarrow \leftarrow (3, r) \cdot (5, r)$

- Good Memory: $(15, g)$
- Bad Memory: ϵ
- State: $(l_0, 15)$

Enforcement Monitor: example

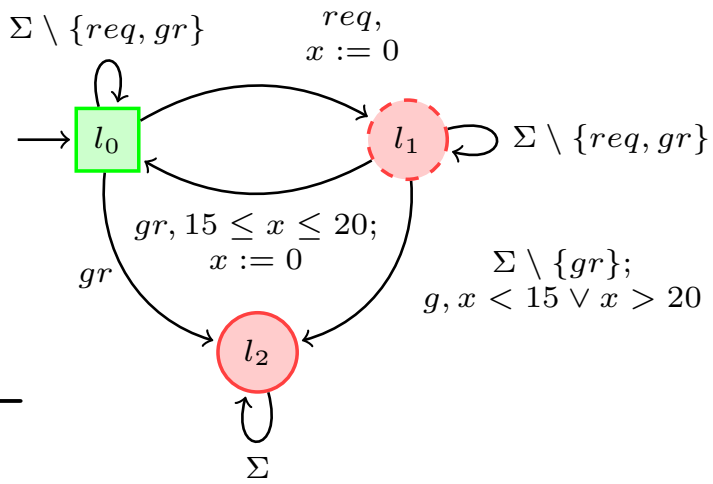
$t = 16$ - Executed operation: *store-*$\overline{\varphi}$



- Good Memory: $(15, g)$
- Bad Memory: $(3, r)$
- State: $(l_0, 15)$

Enforcement Monitor: example

$t = 21$ - Executed operation: $\mathrm{idle}(5)$

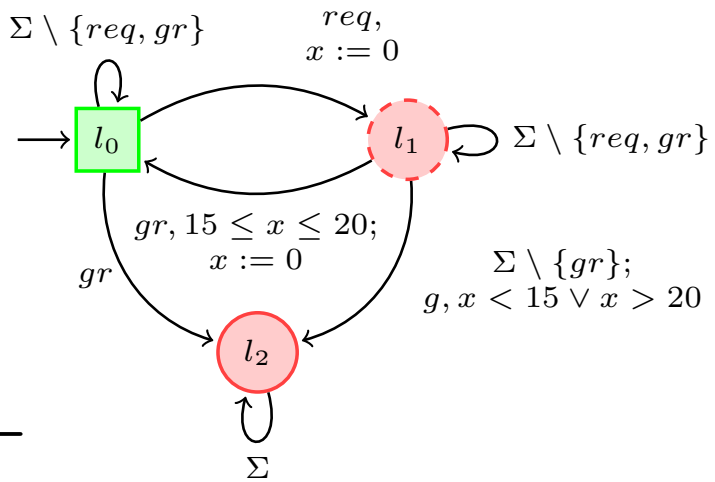


$(13, r)$ ← ← $(5, r)$

- Good Memory: $(15, g)$
- Bad Memory: $(3, r)$
- State: $(l_0, 15)$

Enforcement Monitor: example

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

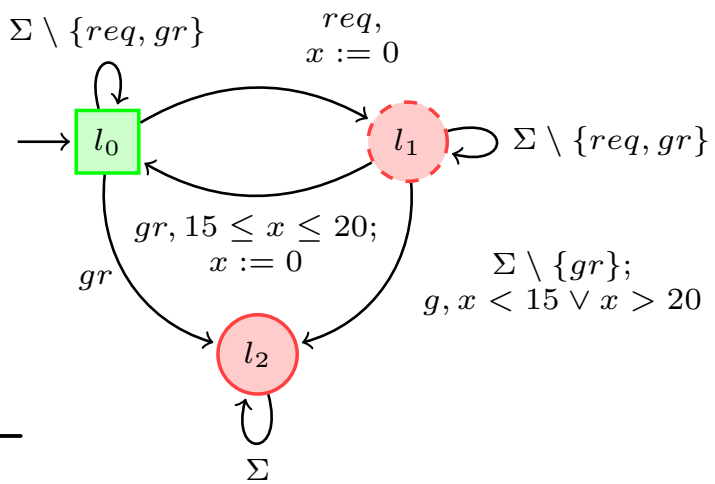


$(13, r)$ ← ← ϵ

- Good Memory: $(15, g)$
- Bad Memory: $(3, r) \cdot (5, r)$
- State: $(l_0, 15)$

Enforcement Monitor: example

$t = 28$ - Executed operation: $\mathrm{idle}(7)$

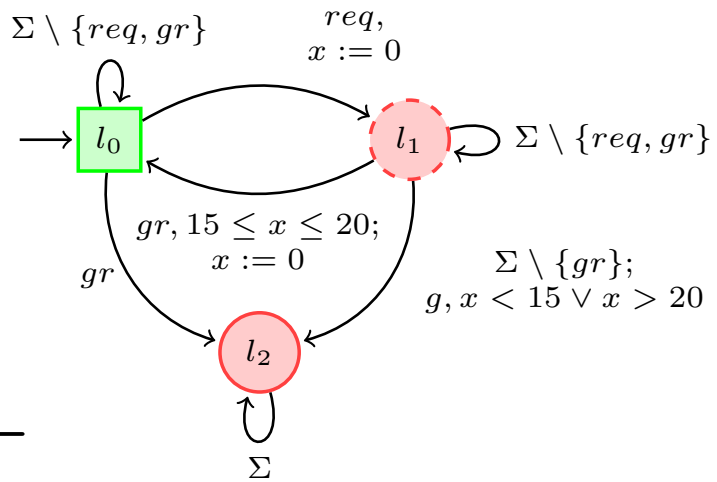


$(13, r)$ $\leftarrow$ $\leftarrow$ ϵ

- Good Memory: $(15, g)$
- Bad Memory: $(3, r) \cdot (5, r)$
- State: $(l_0, 15)$

Enforcement Monitor: example

$t = 28$ - Executed operation: *dump*

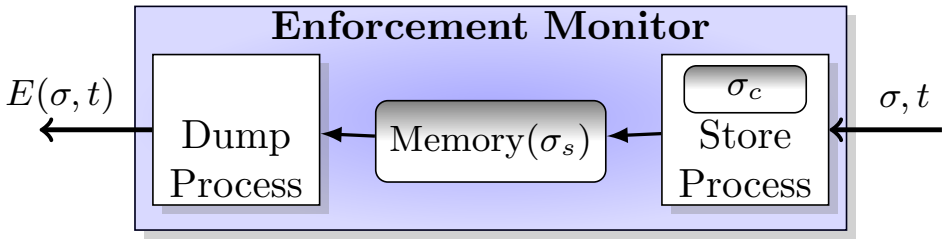


$(13, r) \cdot (15, g)$ ← ← ϵ

- Good Memory: ϵ
- Bad Memory: $(3, r) \cdot (5, r)$
- State: $(l_0, 15)$

Outline - Runt. Enforcement of Regular Timed Properties

Implementation



Algorithm: StoreProcess

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

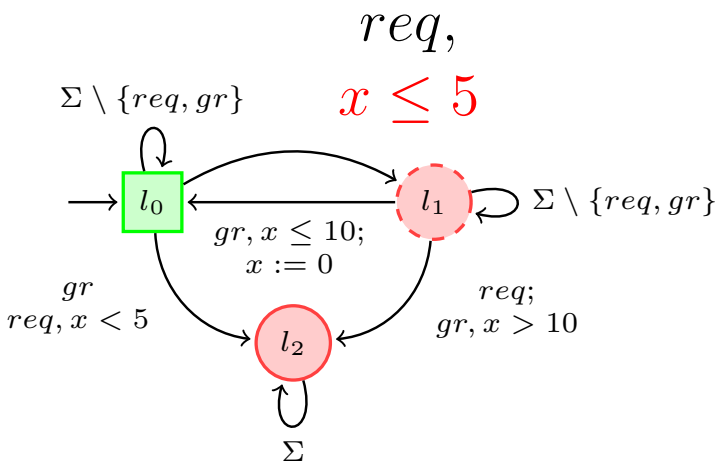
Algorithm: DumpProcess

$d \leftarrow 0$
while `tt` **do**
  await $(\sigma_s \neq \epsilon)$
  $(\delta, a) \leftarrow$ dequeue (σ_s)
  wait $(\delta - d)$
  dump (a)
  $d \leftarrow 0$
end while

Outline - Runt. Enforcement of Regular Timed Properties

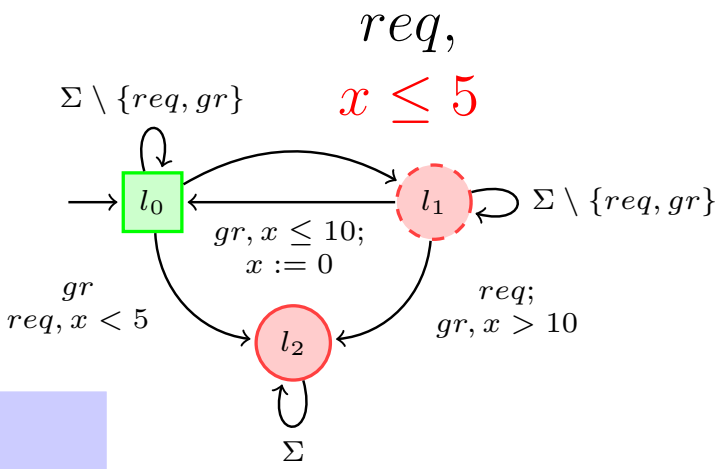
Non-enforceable response properties

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



Non-enforceable response properties

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

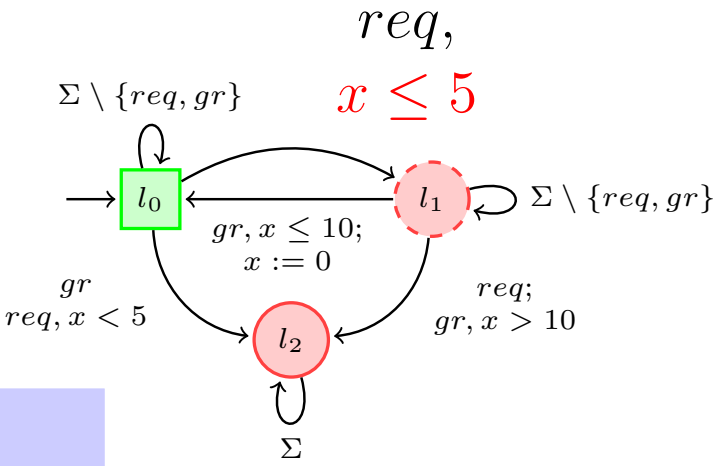


$t \in [0, 3[$

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

Non-enforceable response properties

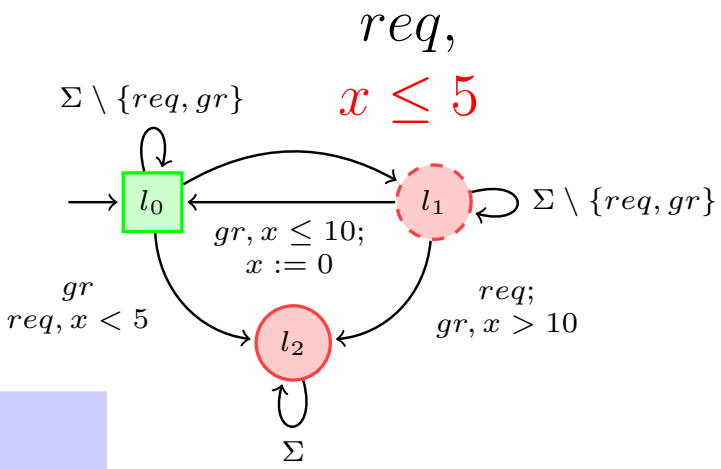
$\Sigma = \{gr, req\}$.
$\sigma = (3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr)$.



$t \in [0, 3[$	$\text{obs}(\sigma, t) = \epsilon$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon)$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$
$t \in [3, 7[$	$\text{obs}(\sigma, t) = (3, req)$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req))$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$

Non-enforceable response properties

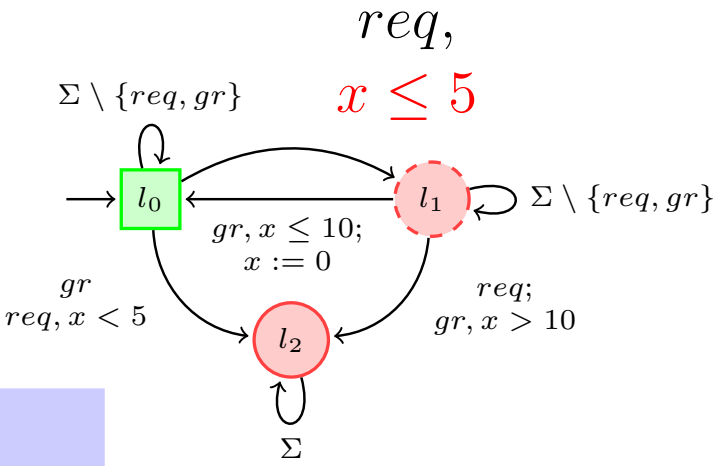
$\Sigma = \{gr, req\}$.
$\sigma = (3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr)$.



$t \in [0, 3[$	$\text{obs}(\sigma, t) = \epsilon$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon)$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$
$t \in [3, 7[$	$\text{obs}(\sigma, t) = (3, req)$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req))$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$
$t \in [7, 9[$	$\text{obs}(\sigma, t) = (3, req) \cdot (4, gr)$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req) \cdot (4, gr))$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$

Non-enforceable response properties

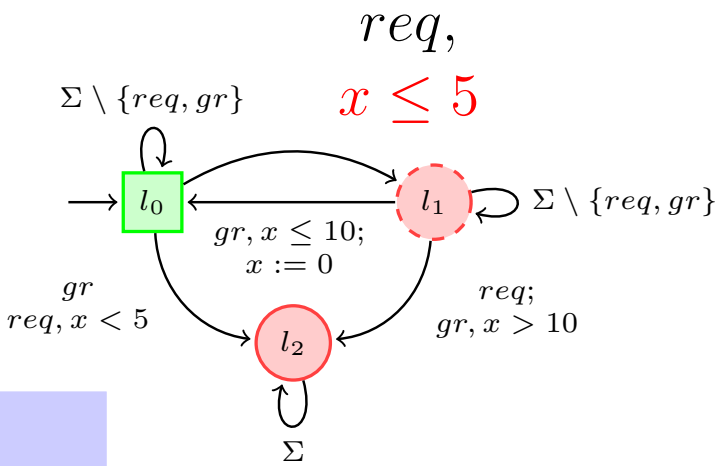
$\Sigma = \{gr, req\}$.
$\sigma = (3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr)$.



$t \in [0, 3[$	$\text{obs}(\sigma, t) = \epsilon$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon)$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$
$t \in [3, 7[$	$\text{obs}(\sigma, t) = (3, req)$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req))$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$
$t \in [7, 9[$	$\text{obs}(\sigma, t) = (3, req) \cdot (4, gr)$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req) \cdot (4, gr))$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$
$t \in [9, 15[$	$\text{obs}(\sigma, t) = (3, req) \cdot (4, gr) \cdot (2, req)$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req) \cdot (4, gr) \cdot (2, req))$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$

Non-enforceable response properties

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



$t \in [0, 3[$	$\text{obs}(\sigma, t) = \epsilon$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon)$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$
$t \in [3, 7[$	$\text{obs}(\sigma, t) = (3, req)$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req))$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$
$t \in [7, 9[$	$\text{obs}(\sigma, t) = (3, req) \cdot (4, gr)$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req) \cdot (4, gr))$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$
$t \in [9, 15[$	$\text{obs}(\sigma, t) = (3, req) \cdot (4, gr) \cdot (2, req)$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req) \cdot (4, gr) \cdot (2, req))$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$
$t \in [15, \infty]$	$\text{obs}(\sigma, t) = (3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr)$ $\text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr))$ $E_\varphi(\sigma, t) = \text{obs}(\epsilon, t)$

Outline - Runt. Enforcement of Regular Timed Properties

Conclusions and Future Work

Enforcement monitoring for systems with timing requirements.

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

Future Work

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