On the Runtime Enforcement of (Timed) Properties

Yliès Falcone
based on joint work with T. Jéron, H. Marchand, S. Pinisetty, M. Renard, A. Rollet

Lecture at the ETR summer school

Univ. Grenoble Alpes, Inria CORSE Team, Laboratoire d'Informatique de Grenoble, France

Runtime verification and enforcement (monitors)

Runtime verification and enforcement:

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

Runtime verification

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

Runtime enforcement

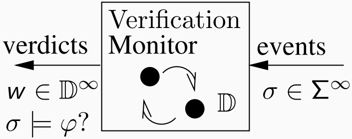
- The run should satisfy the property.
- Input: stream of events.
- Output: stream of **events** (should satisfy the property).

Runtime verification and enforcement (monitors)

Runtime verification and enforcement:

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

Runtime verification



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

Runtime enforcement

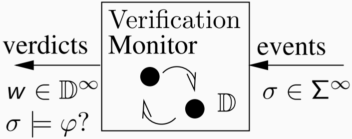
- The run should satisfy the property.
- Input: stream of events.
- Output: stream of **events** (should satisfy the property).

Runtime verification and enforcement (monitors)

Runtime verification and enforcement:

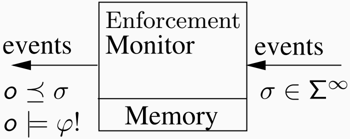
- No system model.
- A correctness property φ.
- A monitor observes the execution of a system (e.g., trace, log, messages).

Runtime verification



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

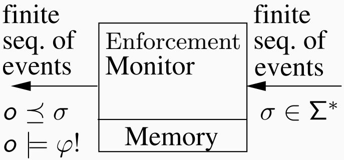
Runtime enforcement



- 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 (intuitively like a "filter").

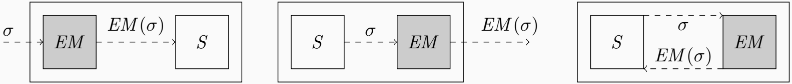
- **reads** an input sequence $\sigma \in \Sigma^*$.
- **outputs** a new sequence $o \in \Sigma^*$.
- endowed with a set of *enforcement primitives*:
  - operate on the memorization mechanism,
  - delete or insert events using the memory content and the current input.

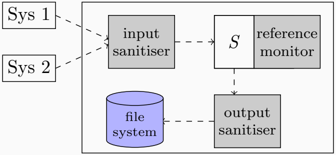
An EM behaves as a function $E : \Sigma^* \to \Sigma^*$.

Application domains and usage scenarios

- Domains: real-time embedded systems, monitor hardware failures, communication protocols, web services and many more.
- Examples of monitor usage:
  - IS: firewall to prevent DOS attack ensuring minimal delay;
  - OS: suppress sensitive information when logging, ensuring a log format;
  - RM: forbid incorrect system change, ensure proper usage of resources.

Input sanitiser (IS) Output sanitiser (OS) Reference monitor (RM)





Outline - On the Runtime Enforcement of Timed Properties

RE of Untimed Properties

Property Enforcement [Schneider00, LigattiBW05, BielovaM08, FalconeFM09a]

Relation between the input and output sequences should adhere:

- **soundness**: the output sequences should be correct wrt. the property
- **transparency**: the correct input sequences should not be modified

$\hookrightarrow$ the memorization mechanism should be designed wrt. those constraints

Definition (Property enforcement)

An EM $E : \Sigma^* \rightarrow \Sigma^*$ for φ is said to enforce

- conservatively (1)
- precisely (2)
- delayed-precisely (3)
- effectively wrt. the equivalence relation $\approx$ (4)

$$\forall \sigma \in \Sigma^* : \quad \begin{array}{rr} \exists o \in \Sigma^* : E(\sigma) = o \wedge \varphi(o) & (1) \\ (1) \wedge \varphi(\sigma) \implies \sigma = o \wedge \forall i < |\sigma| : \quad E(\sigma_{\ldots i}) = \sigma_{\ldots i} & (2) \\ (1) \wedge \varphi(\sigma) \implies \sigma = o \wedge \forall i < |\sigma|, \exists j \leq i : E(\sigma_{\ldots i}) = \sigma_{\ldots j} & (3) \\ (1) \wedge \varphi(\sigma) \implies \sigma \approx o & (4) \end{array}$$

Property Enforcement [Schneider00, LigattiBW05, BielovaM08, FalconeFM09a]

Relation between the input and output sequences should adhere:

- **soundness**: the output sequences should be correct wrt. the property
- **transparency**: the correct input sequences should not be modified

$\hookrightarrow$ the memorization mechanism should be designed wrt. those constraints

Definition (Property enforcement)

An EM $E : \Sigma^* \rightarrow \Sigma^*$ for φ is said to enforce

- conservatively (1)
- precisely (2)
- delayed-precisely (3)
- effectively wrt. the equivalence relation $\approx$ (4)

$$\forall \sigma \in \Sigma^* : \quad \begin{aligned} \exists o \in \Sigma^* : E(\sigma) = o \wedge \varphi(o) \quad & (1) \\ (1) \wedge \varphi(\sigma) \implies \sigma = o \wedge \forall i < |\sigma| : \quad E(\sigma_{\ldots i}) = \sigma_{\ldots i} \quad & (2) \\ (1) \wedge \varphi(\sigma) \implies \sigma = o \wedge \forall i < |\sigma|, \exists j \leq i : E(\sigma_{\ldots i}) = \sigma_{\ldots j} \quad & (3) \\ (1) \wedge \varphi(\sigma) \implies \sigma \approx o \quad & (4) \end{aligned}$$

Outline - On the Runtime Enforcement of Timed Properties

Security Automata (SA) [Schneider00]

First runtime mechanisms dedicated to property enforcement.

- Variant of non-deterministic Büchi automata executing in parallel with the system.
- Mechanisms able to stop the system as soon as a violation of the property is detected: *execution truncation*.

Example (Security Automata (SA))

- Prohibiting "Send" after "FileRead"
  Atomic propositions: $\{FileRead, Sent\}$
- Enforcement of a finitary property $Pref(a \cdot b \cdot c \cdot d) \cup Pref(b \cdot a \cdot d \cdot c)$

Security Automata (SA) [Schneider00]

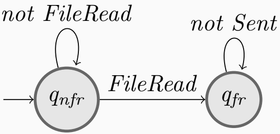
First runtime mechanisms dedicated to property enforcement.

- Variant of non-deterministic Büchi automata executing in parallel with the system.
- Mechanisms able to stop the system as soon as a violation of the property is detected: *execution truncation*.

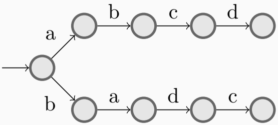
Example (Security Automata (SA))

- Prohibiting "Send" after "FileRead"

  Atomic propositions: $\{FileRead, Sent\}$



- Enforcement of a finitary property $Pref(a \cdot b \cdot c \cdot d) \cup Pref(b \cdot a \cdot d \cdot c)$



Security Automata (SA) and Decidable Safety Properties

SA cannot take decisions based on possible future executions
↪ decisions of SA are irremediable.

SA can enforce properties s.t.:

- "good" sequences are prefix-closed,
- "bad" sequences are rejected after a finite number of steps.

Theorem (Enforcement ability of SA)
SA can enforce conservatively and precisely safety properties

Hypotheses:

- The SA can halt the target system.
- The target system cannot corrupt the SA's transitions.

Security Automata (SA) and Decidable Safety Properties

SA cannot take decisions based on possible future executions
$\hookrightarrow$ decisions of SA are irremediable.

SA can enforce properties s.t.:

- "good" sequences are prefix-closed,
- "bad" sequences are rejected after a finite number of steps.

Theorem (Enforcement ability of SA)
SA can enforce conservatively and precisely safety properties

Hypotheses:

- The SA can halt the target system.
- The target system cannot corrupt the SA's transitions.

Outline - On the Runtime Enforcement of Timed Properties

On the Runtime Enforcement of Untimed Properties

- Security Automata [Schneider00]
- Edit-Automata [LigattiBW05, LigattiBW09]
- Generic Enforcement Monitors [FalconeFM09a]

Specifying Timed Properties

Runtime Enforcement of Timed Properties

Extensions

Conclusions and Future Work

Edit-Automata (EA) [LigattiBW05, LigattiBW09]

Motivated by the limitation of SA that only halt the target system

- SA are "sequence recognizers"
- EA are "sequence transformers"

EAs can

- **insert** an action (by either replacing the current input or inserting it)
- **suppress** an action (possibly *memorized in the control state* for later)

Variants of EA:

- Insertion Automata (only inserting actions)
- Suppression Automata (only suppressing actions)

Hypotheses: actions are *asynchronous*

- next action is available even if some previous actions have been suppressed
- no data-dependency between actions

Memorization of events is realized using control states

Edit-Automata (EA) [LigattiBW05, LigattiBW09]

Motivated by the limitation of SA that only halt the target system

- SA are "sequence recognizers"
- EA are "sequence transformers"

EAs can

- **insert** an action (by either replacing the current input or inserting it)
- **suppress** an action (possibly *memorized in the control state* for later)

Variants of EA:

- Insertion Automata (only inserting actions)
- Suppression Automata (only suppressing actions)

Hypotheses: actions are *asynchronous*

- next action is available even if some previous actions have been suppressed
- no data-dependency between actions

Memorization of events is realized using control states

Edit-Automata (EA) [LigattiBW05, LigattiBW09]

Motivated by the limitation of SA that only halt the target system

- SA are "sequence recognizers"
- EA are "sequence transformers"

EAs can

- **insert** an action (by either replacing the current input or inserting it)
- **suppress** an action (possibly *memorized in the control state* for later)

Variants of EA:

- Insertion Automata (only inserting actions)
- Suppression Automata (only suppressing actions)

Hypotheses: actions are *asynchronous*

- next action is available even if some previous actions have been suppressed
- no data-dependency between actions

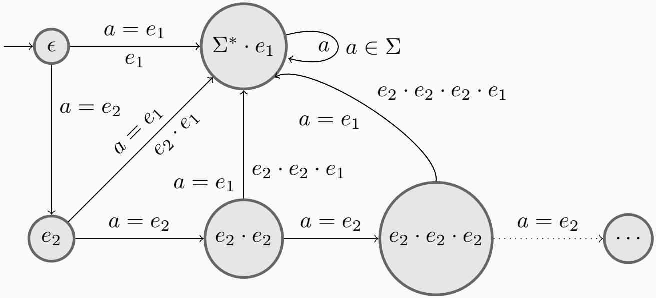
Memorization of events is realized using control states

Example of Edit Automaton

Example (Edit Automaton)

Delayed-precise enforcement of a simple co-safety/guarantee property

- alphabet=$\{e_1, e_2\}$
- property= "eventually, event e_1 occurs"



Remark: Automaton's size also depends on the alphabet's size.

Enforcement abilities of EA-like enforcement mechanisms

EA-like mechanisms form a hierarchy wrt. their enforcement ability.

Theorem (Enf. ability of Ligatti Automata [LigattiThesis, BielovaM08])
Edit Automata can delayed-precisely enforce the set of infinite renewal properties.

φ is an infinite renewal property over Σ^∞ if:

$$\forall \sigma \in \Sigma^\infty : \quad (\varphi(\sigma) \Leftrightarrow \forall \sigma' \in \Sigma^* : \sigma' \prec \sigma \Rightarrow \exists \sigma'' : \sigma' \preceq \sigma'' \prec \sigma \wedge \varphi(\sigma''))$$

Enforcement abilities of EA-like enforcement mechanisms

EA-like mechanisms form a hierarchy wrt. their enforcement ability.

Theorem (Enf. ability of Ligatti Automata [LigattiThesis, BielovaM08])
Edit Automata can delayed-precisely enforce the set of infinite renewal properties.

φ is an infinite renewal property over Σ^∞ if:

$$\forall \sigma \in \Sigma^\infty : \quad (\varphi(\sigma) \Leftrightarrow \forall \sigma' \in \Sigma^* : \sigma' \prec \sigma \Rightarrow \exists \sigma'' : \sigma' \preceq \sigma'' \prec \sigma \wedge \varphi(\sigma''))$$

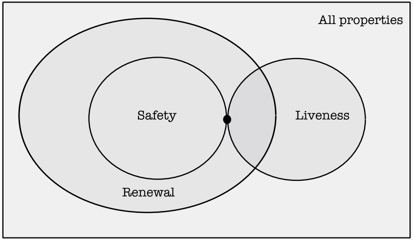
Enforcement abilities of EA-like enforcement mechanisms

EA-like mechanisms form a hierarchy wrt. their enforcement ability.

Theorem (Enf. ability of Ligatti Automata [LigattiThesis, BielovaM08])
Edit Automata can delayed-precisely enforce the set of infinite renewal properties.

φ is an infinite renewal property over Σ^∞ if:

$$\forall \sigma \in \Sigma^\infty : \quad (\varphi(\sigma) \Leftrightarrow \forall \sigma' \in \Sigma^* : \sigma' \prec \sigma \Rightarrow \exists \sigma'' : \sigma' \preceq \sigma'' \prec \sigma \wedge \varphi(\sigma''))$$



Outline - On the Runtime Enforcement of Timed Properties

On the Runtime Enforcement of Untimed Properties

- Security Automata [Schneider00]
- Edit-Automata [LigattiBW05, LigattiBW09]
- Generic Enforcement Monitors [FalconeFM09a]

Specifying Timed Properties

Runtime Enforcement of Timed Properties

Extensions

Conclusions and Future Work

Generic Enforcement Monitors (GEMs)

Definition (Generic enforcement monitor (EM(Ops)))

A *GEM* $\mathcal{A}_\downarrow$ is a 4-tuple $(Q^{\mathcal{A}_\downarrow}, q_{\text{init}}^{\mathcal{A}_\downarrow}, \rightarrow_{\mathcal{A}_\downarrow}, \mathit{Ops})$ wrt. Σ parameterized by *Ops*

- $\mathit{Ops} : \Sigma \times \mathit{Memory} \rightarrow \Sigma^* \times \mathit{memory}$
- complete transition function $\rightarrow_{\mathcal{A}_\downarrow} : Q^{\mathcal{A}_\downarrow} \times \Sigma \rightarrow Q^{\mathcal{A}_\downarrow} \times \mathit{Ops}$
- Enforcement operations $\{\mathrm{halt}, \mathrm{store}, \mathrm{dump}, \mathrm{off}\}$

Advantages:

- Instantiated GEMs encompass SA and Edit-Automata
  - for SA: use $\mathrm{dump}, \mathrm{halt}$
  - for Edit Automata: use $\mathrm{dump}, \mathrm{halt}, \mathrm{store}$
- closer to implementation (finite-state mechanisms)
- their composition is easy to define:
  - ordering enforcement operations: $\mathrm{halt} \sqsubset \mathrm{store} \sqsubset \mathrm{dump} \sqsubset \mathrm{off}$
  - define $\sqcup, \sqcap$ on Ops

Generic Enforcement Monitors (GEMs)

Definition (Generic enforcement monitor (EM(Ops)))

A *GEM* $\mathcal{A}_\downarrow$ is a 4-tuple $(Q^{\mathcal{A}_\downarrow}, q_{\text{init}}^{\mathcal{A}_\downarrow}, \rightarrow_{\mathcal{A}_\downarrow}, \mathit{Ops})$ wrt. Σ parameterized by *Ops*

- $\mathit{Ops} : \Sigma \times \mathit{Memory} \rightarrow \Sigma^* \times \mathit{memory}$
- complete transition function $\rightarrow_{\mathcal{A}_\downarrow} : Q^{\mathcal{A}_\downarrow} \times \Sigma \rightarrow Q^{\mathcal{A}_\downarrow} \times \mathit{Ops}$
- Enforcement operations $\{\text{halt}, \text{store}, \text{dump}, \text{off}\}$

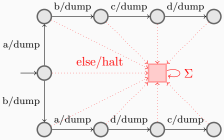
Advantages:

- Instantiated GEMs encompass SA and Edit-Automata
  - for SA: use dump, halt
  - for Edit Automata: use $\text{dump}, \text{halt}, \text{store}$
- closer to implementation (finite-state mechanisms)
- their composition is easy to define:
  - ordering enforcement operations: $\text{halt} \sqsubset \text{store} \sqsubset \text{dump} \sqsubset \text{off}$
  - define $\sqcup, \sqcap$ on Ops

Instantiated GEMs: some examples

Example (Enforcement of a finitary property)

$Pref(a \cdot b \cdot c \cdot d) \cup Pref(b \cdot a \cdot d \cdot c)$



Example (Delayed-precise enforcement of a guarantee property)

- alphabet= $\{e_1, e_2\}$
- property= "eventually, event e_1 occurs"

Example (Logging authentication requests)

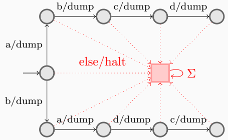
Each occurrence of *r_auth* should be:

1. written in a log file
2. answered
   - either with a *g_auth* or a *d_auth*
   - without any op_s or *r_auth* meanwhile

Instantiated GEMs: some examples

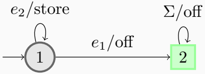
Example (Enforcement of a finitary property)

$Pref(a \cdot b \cdot c \cdot d) \cup Pref(b \cdot a \cdot d \cdot c)$



Example (Delayed-precise enforcement of a guarantee property)

- alphabet= $\{e_1, e_2\}$
- property= "eventually, event e_1 occurs"



Example (Logging authentication requests)

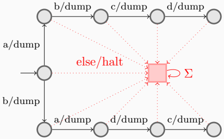
Each occurrence of *r_auth* should be:

1. written in a log file
2. answered
   - either with a *g_auth* or a *d_auth*
   - without any ops or *r_auth* meanwhile

Instantiated GEMs: some examples

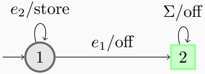
Example (Enforcement of a finitary property)

$$Pref(a \cdot b \cdot c \cdot d) \cup Pref(b \cdot a \cdot d \cdot c)$$



Example (Delayed-precise enforcement of a guarantee property)

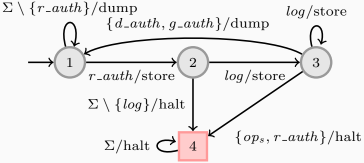
- alphabet= $\{e_1, e_2\}$
- property= "eventually, event e_1 occurs"



Example (Logging authentication requests)

Each occurrence of *r_auth* should be:

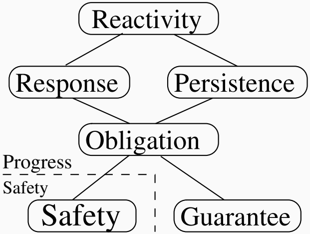
1. written in a log file
2. answered
   - either with a *g_auth* or a *d_auth*
   - without any ops or *r_auth* meanwhile



Enf. ability of instantiated GEMs in the Safety-Progress classification

Theorem (Enf. ability of instantiated GEMs [FalconeFM09a])

GEMs instantiated by $\{\mathrm{halt}, \mathrm{store}, \mathrm{dump}, \mathrm{off}\}$ *can delayed-precisely enforce the set of response properties within the Safety-Progress classification.*



Specifying Timed Properties

Specifying timed properties

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.

System Abstraction

- 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 timed automaton (TA) $\mathcal{A}_\varphi$.

Specifying timed properties

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.

System Abstraction

- 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 timed automaton (TA) $\mathcal{A}_\varphi$.

Specifying timed properties

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.

System Abstraction

- 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 timed automaton (TA) $\mathcal{A}_\varphi$.

Specifying timed properties

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.

System Abstraction

- 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 timed automaton (TA) $\mathcal{A}_\varphi$.

Specifying timed properties

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.

System Abstraction

- 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 timed automaton (TA) $\mathcal{A}_\varphi$.

Specifying timed properties

Safety, co-safety and response properties specified by TAs

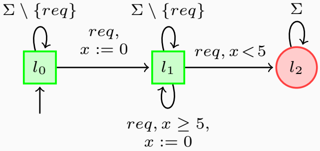
Specifying timed properties

Safety, co-safety and response properties specified by TAs

Specifying timed properties

Safety, co-safety and response properties specified by TAs

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

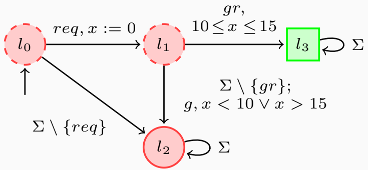


- $\Sigma \supseteq \{req\}$
- "A delay of 5 t.u. between any two requests."

Specifying timed properties

Safety, co-safety and response properties specified by TAs

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

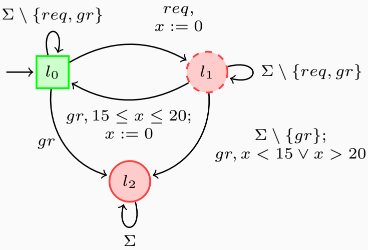


- $\Sigma \supseteq \{req, gr\}$
- "A request, and then a grant should arrive between 10 and 15 t.u."

Specifying timed properties

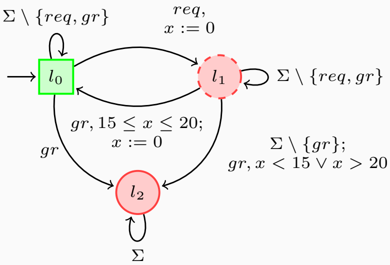
Safety, co-safety and response properties specified by TAs

Response: any property.



- $\Sigma \supseteq \{req, gr\}$
- "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



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

$\epsilon \models \varphi$.

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

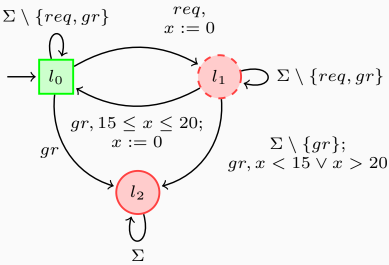
$(3, req) \cdot (15, gr) \models \varphi$.

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

$(3, req) \cdot (15, gr) \cdot (5, req) \cdot (19, gr) \models \varphi$.

Remark: response properties are neither prefix nor extension closed.

Example: response property



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

$\epsilon \models \varphi$.

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

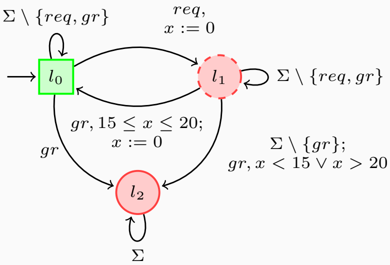
$(3, req) \cdot (15, gr) \models \varphi$.

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

$(3, req) \cdot (15, gr) \cdot (5, req) \cdot (19, gr) \models \varphi$.

Remark: response properties are neither prefix nor extension closed.

Example: response property



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

$\epsilon \models \varphi$.

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

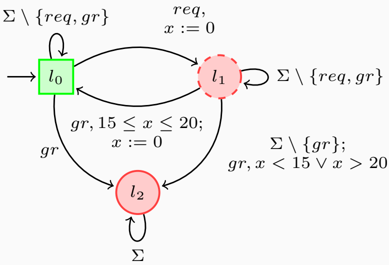
$(3, req) \cdot (15, gr) \models \varphi$.

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

$(3, req) \cdot (15, gr) \cdot (5, req) \cdot (19, gr) \models \varphi$.

Remark: response properties are neither prefix nor extension closed.

Example: response property



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

$\epsilon \models \varphi$.

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

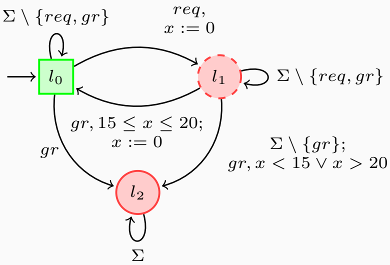
$(3, req) \cdot (15, gr) \models \varphi$.

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

$(3, req) \cdot (15, gr) \cdot (5, req) \cdot (19, gr) \models \varphi$.

Remark: response properties are neither prefix nor extension closed.

Example: response property



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

$\epsilon \models \varphi$.

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

$(3, req) \cdot (15, gr) \models \varphi$.

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

$(3, req) \cdot (15, gr) \cdot (5, req) \cdot (19, gr) \models \varphi$.

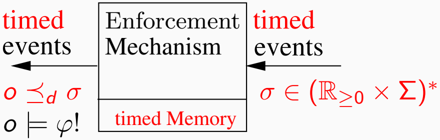
Remark: response properties are neither prefix nor extension closed.

RE of Timed Properties

Outline - On the Runtime Enforcement of Timed Properties

Problem statement

Given some (regular) timed property φ:



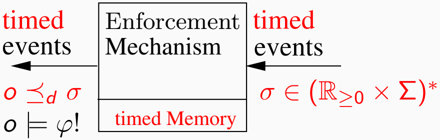
What can an enforcement mechanism do?

- CANNOT insert events.
- CANNOT change the order of events.
- CAN increase the delay between actions.
- CAN delete events.

$\hookrightarrow$ How can we obtain an enforcement mechanism as a "delayer" with suppression for φ.

Problem tackled and Contributions

φ is a timed property

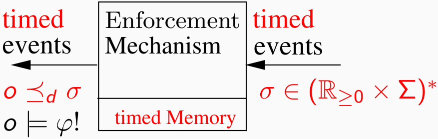


A formal framework for runtime enforcement of timed properties

- Any regular timed property φ as input.
- Enforcement mechanisms i) add additional delays between actions to satisfy φ, ii) suppress actions. – work as "delayers" with suppression.
- 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*.

Problem tackled and Contributions

φ is a timed property

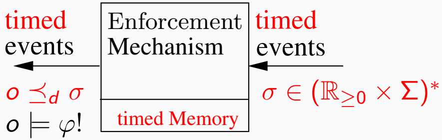


A formal framework for runtime enforcement of timed properties

- Any regular timed property φ as input.
- Enforcement mechanisms i) add additional delays between actions to satisfy φ, ii) suppress actions. – work as "delayers" with suppression.
- 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*.

Problem tackled and Contributions

φ is a timed property

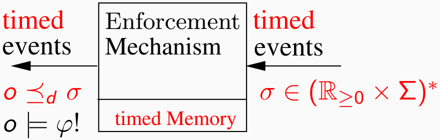


A formal framework for runtime enforcement of timed properties

- Any regular timed property φ as input.
- Enforcement mechanisms i) add additional delays between actions to satisfy φ, ii) suppress actions. – work as “delayers” with suppression.
- 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*.

Problem tackled and Contributions

φ is a timed property

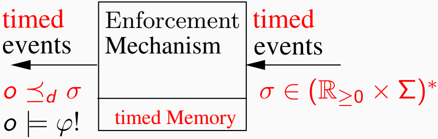


A formal framework for runtime enforcement of timed properties

- Any regular timed property φ as input.
- Enforcement mechanisms i) add additional delays between actions to satisfy φ, ii) suppress actions. – work as “delayers” with suppression.
- 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*.

Problem tackled and Contributions

φ is a timed property

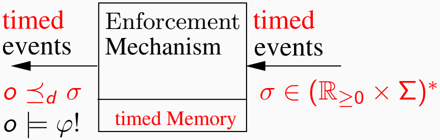


A formal framework for runtime enforcement of timed properties

- Any regular timed property φ as input.
- Enforcement mechanisms i) add additional delays between actions to satisfy φ, ii) suppress actions. – work as "delayers" with suppression.
- 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.*

Problem tackled and Contributions

φ is a timed property

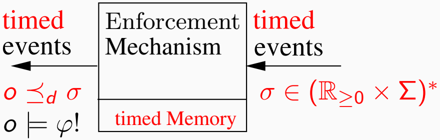


A formal framework for runtime enforcement of timed properties

- Any regular timed property φ as input.
- Enforcement mechanisms i) add additional delays between actions to satisfy φ, ii) suppress actions. – work as "delayers" with suppression.
- 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.*

Problem tackled and Contributions

φ is a timed property



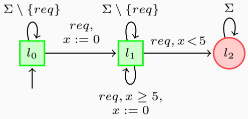
A formal framework for runtime enforcement of timed properties

- Any regular timed property φ as input.
- Enforcement mechanisms i) add additional delays between actions to satisfy φ, ii) suppress actions. – work as "delayers" with suppression.
- 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*.

Main challenges when enforcing timed properties

Main challenges when (possibly) correcting an input sequence:

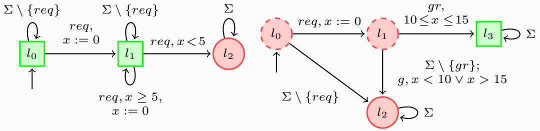
- safety properties: after each event, the decision is made (i.e., whether it can be corrected or not).
- co-safety properties: after each event, we check starting from the first event, whether the sequence read so far can be corrected or not.
- response properties:
  - we cannot decide for each event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



Main challenges when enforcing timed properties

Main challenges when (possibly) correcting an input sequence:

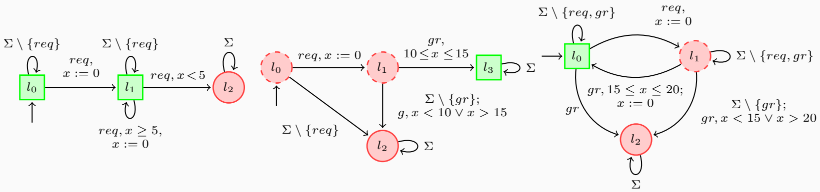
- safety properties: after each event, the decision is made (i.e., whether it can be corrected or not).
- co-safety properties: after each event, we check starting from the first event, whether the sequence read so far can be corrected or not.
- response properties:
  - we cannot decide for each event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



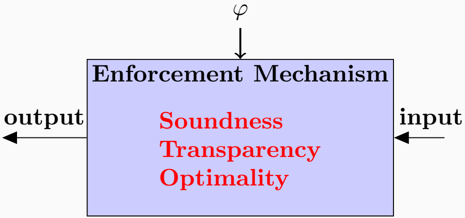
Main challenges when enforcing timed properties

Main challenges when (possibly) correcting an input sequence:

- safety properties: after each event, the decision is made (i.e., whether it can be corrected or not).
- co-safety properties: after each event, we check starting from the first event, whether the sequence read so far can be corrected or not.
- 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 chunks of sequences as soon as possible.

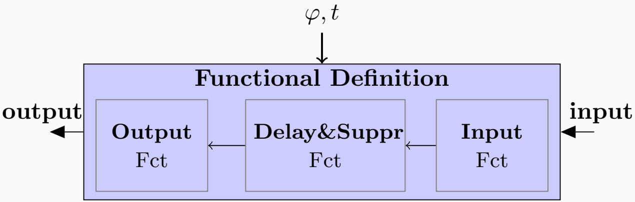


Summary of the approach



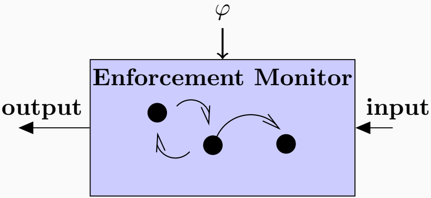
- **Requirements** for any enforcement mechanism for φ.
- **Functional definition** (satisfies the requirements):
  - description of the input/output behavior ;
  - composition of 3 functions: process input, computing the delayed timed word, and process output.
- **Enforcement monitor**:
  - description 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



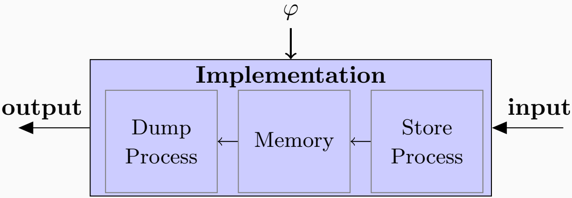
- **Requirements** for any enforcement mechanism for φ.
- **Functional definition** (satisfies the requirements):
  - description of the input/output behavior ;
  - composition of 3 functions: process input, computing the delayed timed word, and process output.
- **Enforcement monitor**:
  - description 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 - On the Runtime Enforcement of Timed Properties

Requirements on an Enforcement Mechanism

Specified on an enforcement function for φ

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

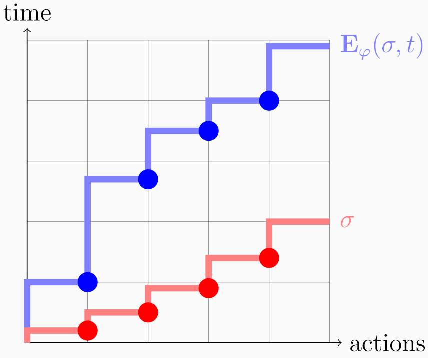
where for $\sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$ and $t \in \mathbb{R}_{\geq 0}$:

$E_\varphi(\sigma, t)$ is the sequence produced by the enforcement mechanism

- at time t,
- if σ is the sequence read as input.

Requirements on an Enforcement Mechanism – physical constraints

Physical constraints: The input and output are timed words. The output is produced in a "streaming fashion"[1]



Phy1 $t \leq t' \implies E_\varphi(\sigma, t) \preccurlyeq E_\varphi(\sigma, t')$.

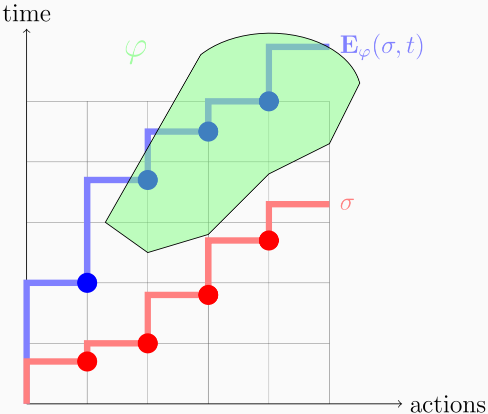
Phy2 $\sigma \preccurlyeq \sigma' \implies E_\varphi(\sigma, t) \preccurlyeq E_\varphi(\sigma', t)$.

[1] Implicit universal quantification over σ and t.

Requirements on an Enforcement Mechanism – soundness

Soundness: The output is (eventually) correct

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



Requirements on an Enforcement Mechanism – transparency

Transparency: (prefix) relation between the input and output sequences

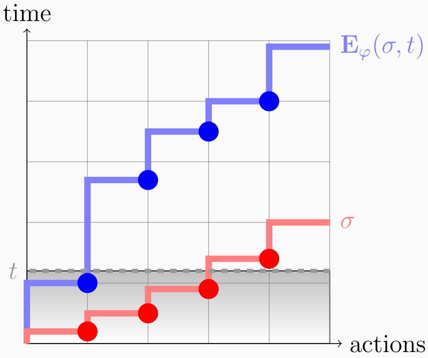
Requirements on an Enforcement Mechanism – transparency

Transparency: (prefix) relation between the input and output sequences

What the enforcement mechanism observes at time t is

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

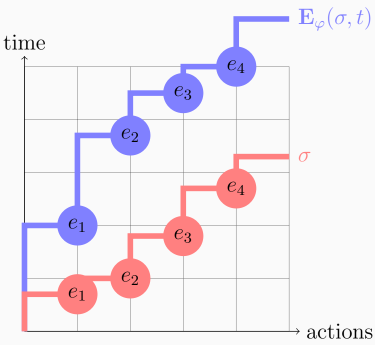
$\hookrightarrow$ the (max) prefix of σ that can be observed with t t.u.

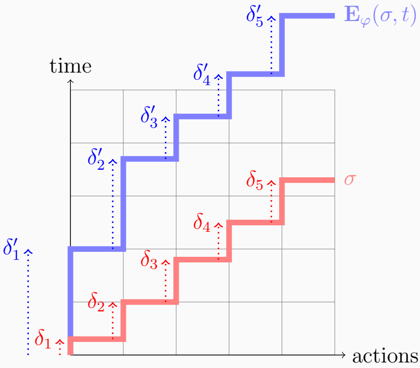


Requirements on an Enforcement Mechanism – transparency

Transparency: (prefix) relation between the input and output sequences

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



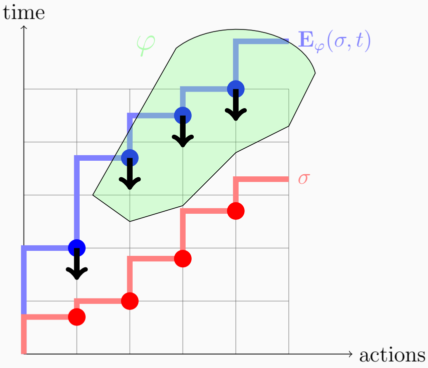


Requirements on an Enforcement Mechanism – optimality

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

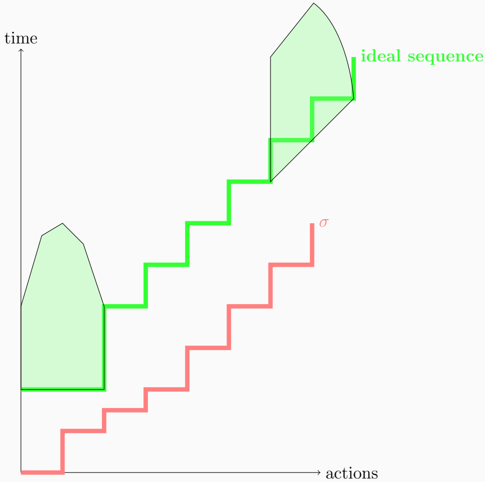
Requirements on an Enforcement Mechanism – optimality

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



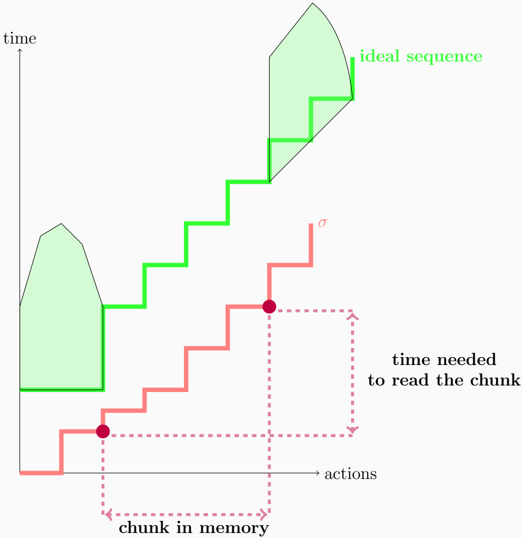
Requirements on an Enforcement Mechanism – optimality

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



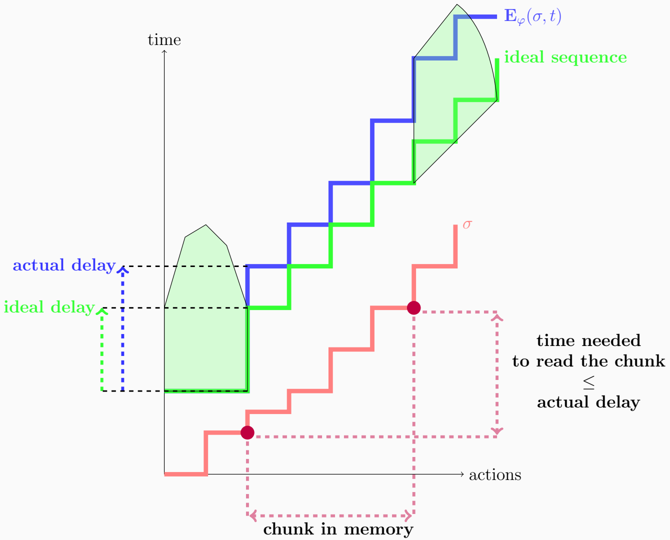
Requirements on an Enforcement Mechanism – optimality

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



Requirements on an Enforcement Mechanism – optimality

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

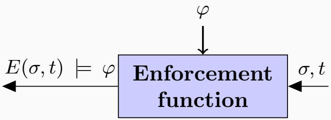


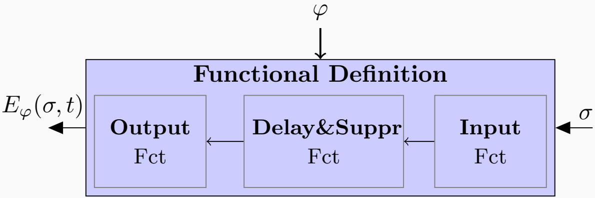
Outline - On the Runtime Enforcement of Timed Properties

Functional definition (1)

The functional definition describes the mechanism as a function

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





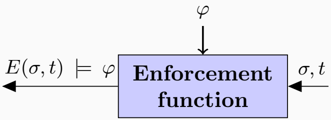
Input and output functions are realized by the *observation function*:

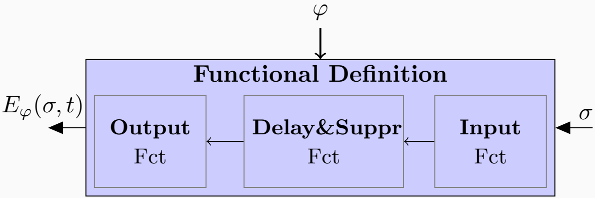
$$\text{obs}(\sigma, t) = \max_{\preccurlyeq}\{\sigma' \mid \sigma' \preccurlyeq \sigma \wedge \text{time}(\sigma') \leq t\}.$$

Functional definition (1)

The functional definition describes the mechanism as a function

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





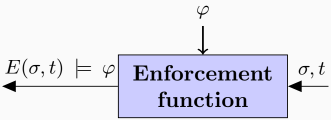
Input and output functions are realized by the *observation function*:

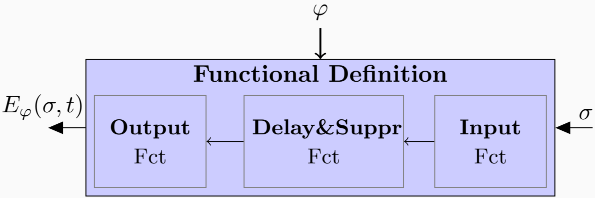
$$\text{obs}(\sigma, t) = \max_{\preccurlyeq}\{\sigma' \mid \sigma' \preccurlyeq \sigma \wedge \text{time}(\sigma') \leq t\}.$$

Functional definition (1)

The functional definition describes the mechanism as a function

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





Input and output functions are realized by the *observation function*:

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

(Simplified) Functional definition (2) – no suppression

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

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

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

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

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

(Simplified) Functional definition (2) – no suppression

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

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

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

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

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

(Simplified) Functional definition (3) – no suppression

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

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

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

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

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

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

with

$$K = \kappa_\varphi(\mathrm{time}(\sigma) + \delta, \sigma_s, \sigma_c \cdot (\delta, a))$$

$$\kappa_\varphi(T, \sigma_s, \sigma_c) = \{w \in (\mathbb{R}_{\geq 0} \times \Sigma)^* \mid w \preccurlyeq_d \sigma_c \wedge |w| = |\sigma_c| \wedge \sigma_s \cdot w \models \varphi \wedge \mathrm{delay}(w(1)) \geq T - \mathrm{time}(\sigma_s)\}$$

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

(Simplified) Functional definition (3) – no suppression

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

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

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

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

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

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

with

$$K = \kappa_\varphi(\mathrm{time}(\sigma) + \delta, \sigma_s, \sigma_c \cdot (\delta, a))$$

$$\kappa_\varphi(T, \sigma_s, \sigma_c) = \{ w \in (\mathbb{R}_{\geq 0} \times \Sigma)^* \mid w \preccurlyeq_d \sigma_c \wedge |w| = |\sigma_c| \wedge \sigma_s \cdot w \models \varphi \wedge \mathrm{delay}(w(1)) \geq T - \mathrm{time}(\sigma_s) \}$$

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

(Simplified) Functional definition (3) – no suppression

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

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

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

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

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

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

with

$$K = \kappa_\varphi(\text{time}(\sigma) + \delta, \sigma_s, \sigma_c \cdot (\delta, a))$$

$$\kappa_\varphi(T, \sigma_s, \sigma_c) = \{w \in (\mathbb{R}_{\geq 0} \times \Sigma)^* \mid w \preccurlyeq_d \sigma_c \wedge |w| = |\sigma_c| \wedge \sigma_s \cdot w \models \varphi \wedge \text{delay}(w(1)) \geq T - \text{time}(\sigma_s)\}$$

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

(Simplified) Functional definition (3) – no suppression

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

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

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

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

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

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

with

$$K = \kappa_\varphi(\mathrm{time}(\sigma) + \delta, \sigma_s, \sigma_c \cdot (\delta, a))$$

$$\kappa_\varphi(T, \sigma_s, \sigma_c) = \{w \in (\mathbb{R}_{\geq 0} \times \Sigma)^* \mid w \preccurlyeq_d \sigma_c \wedge |w| = |\sigma_c| \wedge \sigma_s \cdot w \models \varphi \wedge \mathrm{delay}(w(1)) \geq T - \mathrm{time}(\sigma_s)\}$$

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

(Simplified) Functional definition (3) – no suppression

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

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

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

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

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

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

with

$$K = \kappa_\varphi(\mathrm{time}(\sigma) + \delta, \sigma_s, \sigma_c \cdot (\delta, a))$$

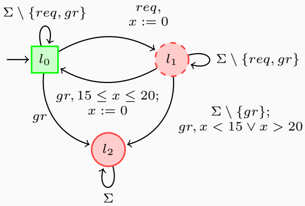
$$\kappa_\varphi(T, \sigma_s, \sigma_c) = \{w \in (\mathbb{R}_{\geq 0} \times \Sigma)^* \mid w \preccurlyeq_d \sigma_c \wedge |w| = |\sigma_c| \wedge \sigma_s \cdot w \models \varphi \wedge \mathrm{delay}(w(1)) \geq T - \mathrm{time}(\sigma_s)\}$$

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

Functional definition: Example

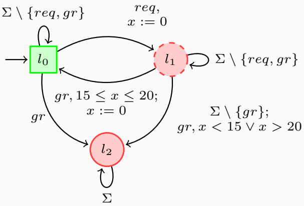
$\Sigma = \{req, gr\}$.

$\sigma = (3, req) \cdot (10, gr) \cdot (3, req) \cdot (5, req)$.



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

Functional definition: Example

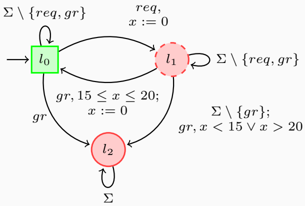


$\Sigma = \{req, gr\}$.

$\sigma = (3, req) \cdot (10, gr) \cdot (3, req) \cdot (5, req)$.

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

Functional definition: Example



$\Sigma = \{req, gr\}$.

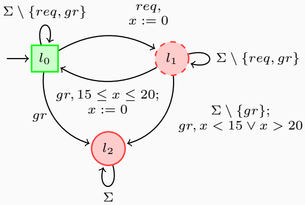
$\sigma = (3, req) \cdot (10, gr) \cdot (3, req) \cdot (5, req)$.

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

Functional definition: Example

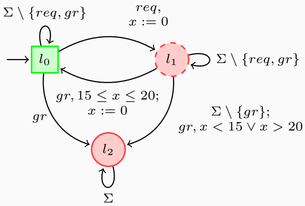
$\Sigma = \{req, gr\}$.

$\sigma = (3, req) \cdot (10, gr) \cdot (3, req) \cdot (5, req)$.



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

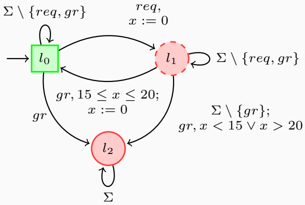
Functional definition: Example



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

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

Functional definition: Example



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

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

The enforcement function satisfies the requirements

Proposition: Enforcement function vs requirements

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

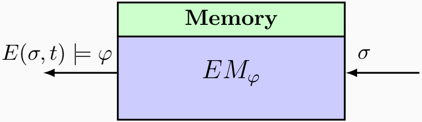
Proof

By induction on the length of the input sequence.

See papers.

Outline - On the Runtime Enforcement of Timed Properties

Enforcement monitor



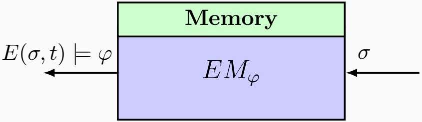
A *rule-based transition system*:

- 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;
- an initial configuration;
- rule-based transitions executing enforcement operations (cf. next slide).

Remark 1: for safety and co-safety, some memories can be discarded.

Remark 2: formal definition in papers.

Enforcement monitor



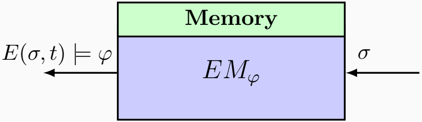
A *rule-based transition system*:

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

Remark 2: formal definition in papers.

Enforcement monitor



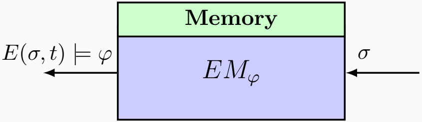
A *rule-based transition system*:

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

Remark 2: formal definition in papers.

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

Remark 2: formal definition in papers.

Enforcement monitor: operations

1. **store-$\overline{\varphi}$**
   - when a new event is received and it *cannot make* φ *satisfied* by delaying.
   - updates "bad" memory and store clock
2. **store-φ**
   - when a new event is received and it *can make* φ *satisfied* by delaying
   - updates "good" memory and store clock
3. **suppress**
   - when an event is received and prevents φ's satisfaction
4. **dump**
   - when an event in the good memory can be released
   - updates "good" memory and dump clock
5. **idle**
   - when no other rule applies (i.e., when time elapses and nothing happens)
   - updates dump and store clocks

Enforcement monitor: operations

1. **store-$\overline{\varphi}$**
   - when a new event is received and it *cannot make* φ *satisfied* by delaying.
   - updates “bad” memory and store clock
2. **store-φ**
   - when a new event is received and it *can make* φ *satisfied* by delaying
   - updates “good” memory and store clock
3. **suppress**
   - when an event is received and prevents φ's satisfaction
4. **dump**
   - when an event in the good memory can be released
   - updates “good” memory and dump clock
5. **idle**
   - when no other rule applies (i.e., when time elapses and nothing happens)
   - updates dump and store clocks

Enforcement monitor: operations

1. **store-$\overline{\varphi}$**
   - when a new event is received and it *cannot make* φ *satisfied* by delaying.
   - updates "bad" memory and store clock
2. **store-φ**
   - when a new event is received and it *can make* φ *satisfied* by delaying
   - updates "good" memory and store clock
3. **suppress**
   - when an event is received and prevents φ's satisfaction
4. **dump**
   - when an event in the good memory can be released
   - updates "good" memory and dump clock
5. **idle**
   - when no other rule applies (i.e., when time elapses and nothing happens)
   - updates dump and store clocks

Enforcement monitor: operations

1. **store-$\overline{\varphi}$**
   - when a new event is received and it *cannot make* φ *satisfied* by delaying.
   - updates "bad" memory and store clock
2. **store-φ**
   - when a new event is received and it *can make* φ *satisfied* by delaying
   - updates "good" memory and store clock
3. **suppress**
   - when an event is received and prevents φ's satisfaction
4. **dump**
   - when an event in the good memory can be released
   - updates "good" memory and dump clock
5. **idle**
   - when no other rule applies (i.e., when time elapses and nothing happens)
   - updates dump and store clocks

Enforcement monitor: operations

1. **store-$\overline{\varphi}$**
   - when a new event is received and it *cannot make* φ *satisfied* by delaying.
   - updates "bad" memory and store clock
2. **store-φ**
   - when a new event is received and it *can make* φ *satisfied* by delaying
   - updates "good" memory and store clock
3. **suppress**
   - when an event is received and prevents φ's satisfaction
4. **dump**
   - when an event in the good memory can be released
   - updates "good" memory and dump clock
5. **idle**
   - when no other rule applies (i.e., when time elapses and nothing happens)
   - updates dump and store clocks

Enforcement Monitor: correctness

Implementation relation between Enforcement Monitor and Enforcement Function

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

Proof

By induction on the length of the input sequence and "integrating the behavior of enforcement monitors over time".

See papers.

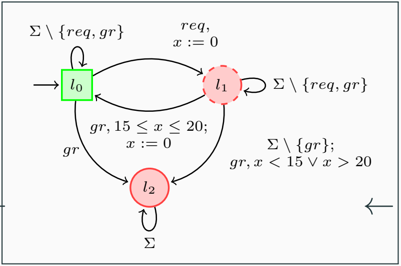
Corollary

Enforcement Monitors respect soundness, transparency, and optimality.

Enforcement Monitor: example

$t = 0$ - Executed operation: none

$\epsilon \longleftarrow$



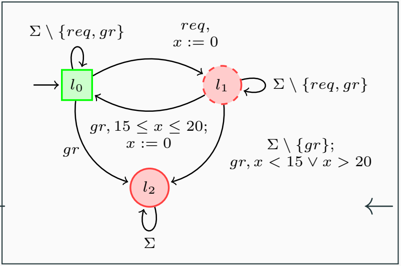
$\longleftarrow (3, r) \cdot (10, g) \cdot (3, r) \cdot (5, r)$

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

Enforcement Monitor: example

$t = 3$ - Executed operation: idle(3)

ϵ $\longleftarrow$



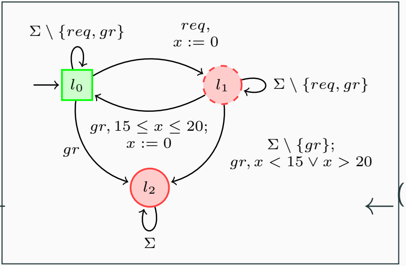
$\longleftarrow$ $(3, r) \cdot (10, g) \cdot (3, r) \cdot (5, r)$

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

Enforcement Monitor: example

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

ϵ $\longleftarrow$



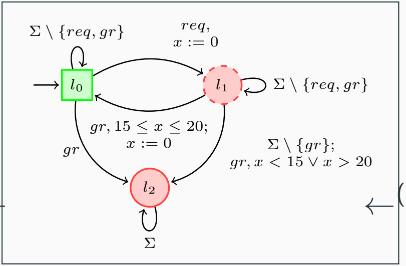
$\longleftarrow (10, g) \cdot (3, r) \cdot (5, r)$

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

Enforcement Monitor: example

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

ϵ $\longleftarrow$



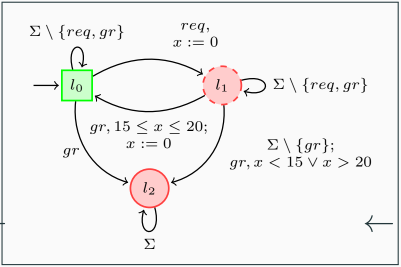
$\longleftarrow (10, g) \cdot (3, r) \cdot (5, r)$

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

Enforcement Monitor: example

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

ϵ $\longleftarrow$



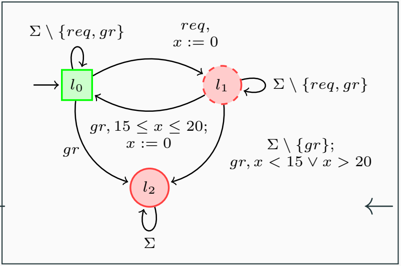
$\longleftarrow$ $(3, r) \cdot (5, r)$

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

Enforcement Monitor: example

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

$(13, r)$ $\longleftarrow$



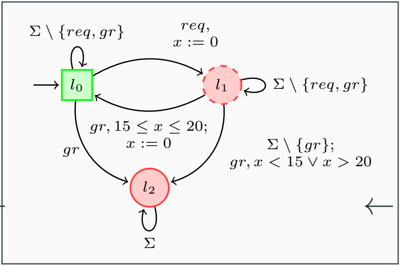
$\longleftarrow$ $(3, r) \cdot (5, r)$

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

Enforcement Monitor: example

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

$(13, r)$ $\leftarrow$

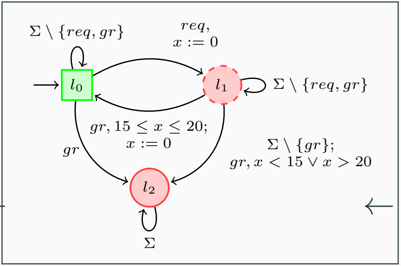


$\leftarrow$ $(3, r) \cdot (5, r)$

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

Enforcement Monitor: example

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

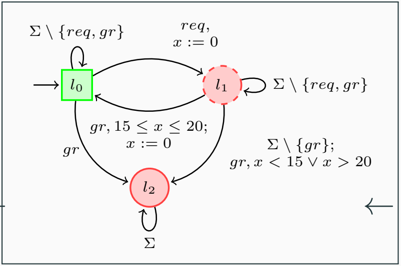
$(13, r)$ $\leftarrow$  $\leftarrow$ $(5, r)$

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

Enforcement Monitor: example

$t = 21$ - Executed operation: idle(5)

$(13, r)$ $\leftarrow$



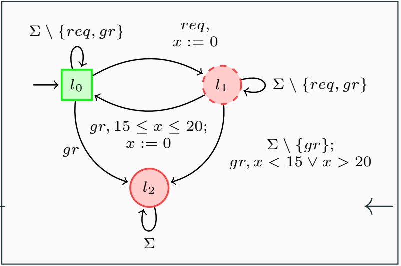
$\leftarrow$ $(5, r)$

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

Enforcement Monitor: example

$t = 21$ - Executed operation: *suppress*

$(13, r)$ $\leftarrow$



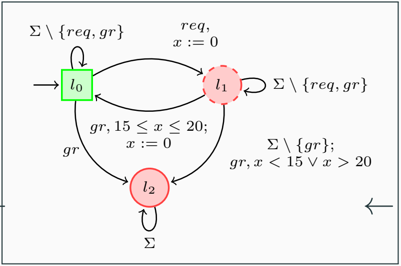
$\leftarrow$ ϵ

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

Enforcement Monitor: example

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

$(13, r)$ $\leftarrow$



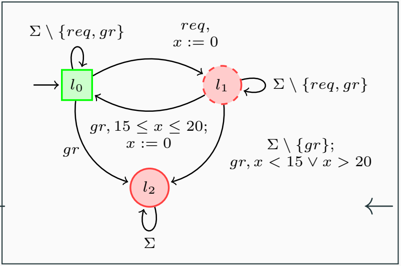
$\leftarrow$ ϵ

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

Enforcement Monitor: example

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

$(13, r) \cdot (15, g)$ $\leftarrow$

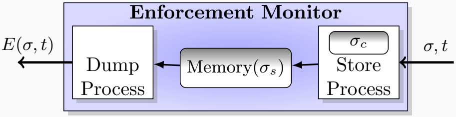


$\leftarrow$ ϵ

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

Outline - On the Runtime Enforcement of Timed Properties

Implementation – simplified algorithms



Used primitives:

- await(*condition*), wait(*time*)
- post(*loc*, *valuation*, *tw*)
- update(*loc*, *valuation*, *tw*)

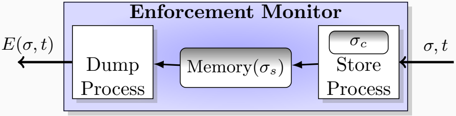
Algorithm: DumpProcess

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

Algorithm: StoreProcess

$(l, \nu) \leftarrow (l_0, [X \leftarrow 0])$
$\sigma_s, \sigma_c \leftarrow \epsilon$
while tt **do**
  $(\delta, a) \leftarrow$ await ($event$)
  $\sigma_c \leftarrow \sigma_c \cdot (\delta, a)$
  $(\sigma'_c, isPath) \leftarrow$ update(l, ν, σ_c)
  if $isPath$ = tt **then**
    $\sigma_s \leftarrow \sigma_s \cdot \sigma'_c$
    $(l, \nu) \leftarrow$ post(l, ν, σ'_c)
    $\sigma_c \leftarrow \epsilon$
  end if
end while

Implementation – simplified algorithms



Used primitives:

- await(*condition*), wait(*time*)
- post(*loc*, *valuation*, *tw*)
- update(*loc*, *valuation*, *tw*)

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

Algorithm: StoreProcess

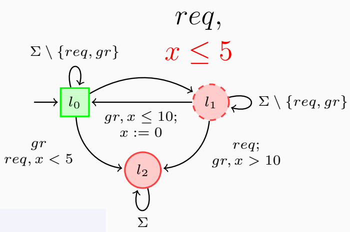
$(l, \nu) \leftarrow (l_0, [X \leftarrow 0])$
$\sigma_s, \sigma_c \leftarrow \epsilon$
while tt **do**
  $(\delta, a) \leftarrow$ await $(event)$
  $\sigma_c \leftarrow \sigma_c \cdot (\delta, a)$
  $(\sigma'_c, isPath) \leftarrow \text{update}(l, \nu, \sigma_c)$
  if $isPath =$ tt **then**
    $\sigma_s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

Outline - On the Runtime Enforcement of Timed Properties

Non-enforceable response properties

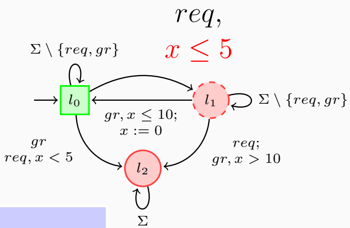
$\Sigma = \{gr, req\}$.

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



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

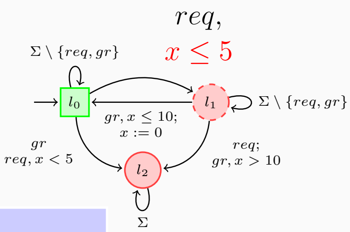
Non-enforceable response properties



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

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

Non-enforceable response properties

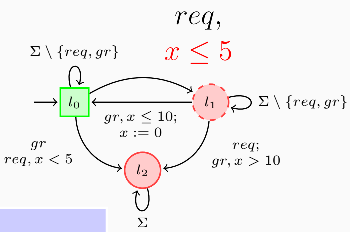


$\Sigma = \{gr, req\}$.

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

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

Non-enforceable response properties

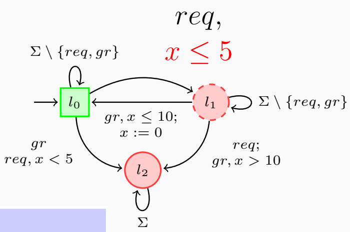


$\Sigma = \{gr, req\}$.

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

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

Non-enforceable response properties

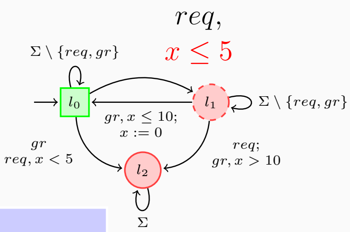


$\Sigma = \{gr, req\}$.

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

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

Non-enforceable response properties



$\Sigma = \{gr, req\}$.

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

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

Extensions

Outline - On the Runtime Enforcement of Timed Properties

Outline - On the Runtime Enforcement of Timed Properties

Uncontrollable events

Two sets of events : controllable (Σ_c) and uncontrollable (Σ_u).

Uncontrollable Events

Uncont. events must be emitted upon reception (i.e., only observable events).

New definitions and challenges

- Delays between stored controllable events have to be recomputed upon reception of each uncont. event
- Prevent the system of reaching a bad state upon reception of any sequence of uncont. events → uncont. events must be anticipated
- Enforceability depends on the received uncontrollable events

Contributions

- Redefining soundness, transparency, and optimality.
- Enforcement mechanisms at two levels of abstraction, *functional* and *operational*, for both untimed and timed regular properties

Uncontrollable events

Two sets of events : controllable (Σ_c) and uncontrollable (Σ_u).

Uncontrollable Events

Uncont. events must be emitted upon reception (i.e., only observable events).

New definitions and challenges

- Delays between stored controllable events have to be recomputed upon reception of each uncont. event
- Prevent the system of reaching a bad state upon reception of any sequence of uncont. events $\rightarrow$ uncont. events must be anticipated
- Enforceability depends on the received uncontrollable events

Contributions

- Redefining soundness, transparency, and optimality.
- Enforcement mechanisms at two levels of abstraction, *functional* and *operational*, for both untimed and timed regular properties

Uncontrollable events

Two sets of events : controllable (Σ_c) and uncontrollable (Σ_u).

Uncontrollable Events

Uncont. events must be emitted upon reception (i.e., only observable events).

New definitions and challenges

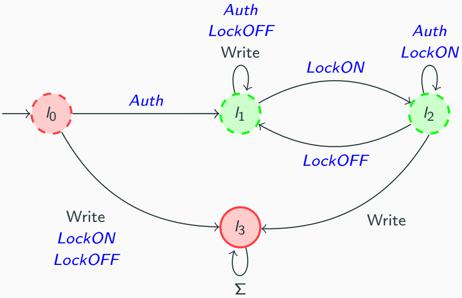
- Delays between stored controllable events have to be recomputed upon reception of each uncont. event
- Prevent the system of reaching a bad state upon reception of any sequence of uncont. events → uncont. events must be anticipated
- Enforceability depends on the received uncontrollable events

Contributions

- Redefining soundness, transparency, and optimality.
- Enforcement mechanisms at two levels of abstraction, *functional* and *operational*, for both untimed and timed regular properties

Example of Non-Enforceable Property

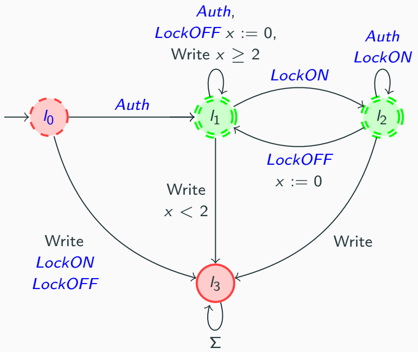
Example of a simple shared storage device example (without time).
$\Sigma_u = \{Auth, LockOFF, LockON\}, \Sigma_c = \{\text{Write}\}$



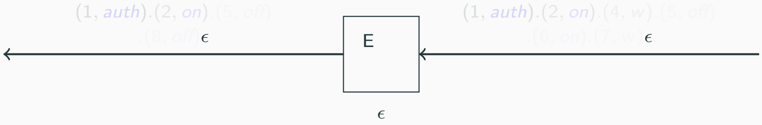
- In l_0, impossible to ensure correctness of this property
- If *Auth* is read, then this property becomes *enforceable*

Property with time: example of execution

$\Sigma_u = \{Auth, LockOFF, LockON\}, \Sigma_c = \{\text{Write}\}$

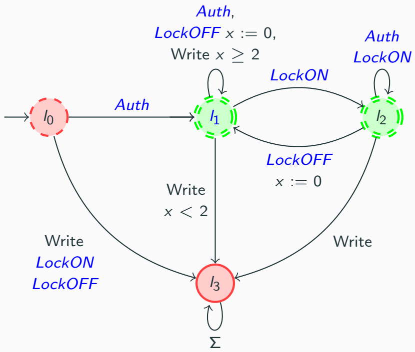


t = 0

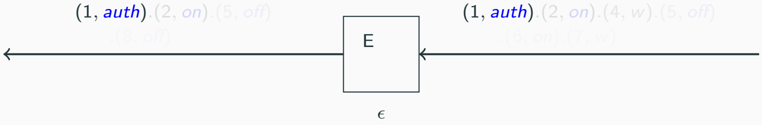


Property with time: example of execution

$\Sigma_u = \{Auth, LockOFF, LockON\}, \Sigma_c = \{\text{Write}\}$

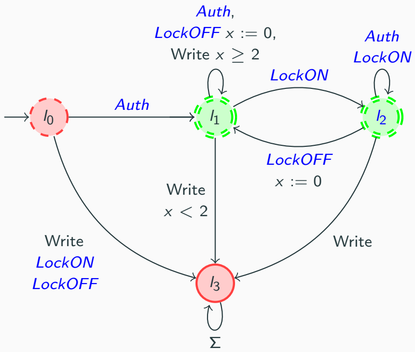


t = 1



Property with time: example of execution

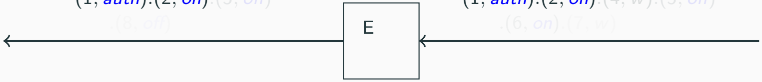
$\Sigma_u = \{Auth, LockOFF, LockON\}, \Sigma_c = \{\text{Write}\}$



t = 2

$(1, auth).(2, on).(5, off).(8, off)$

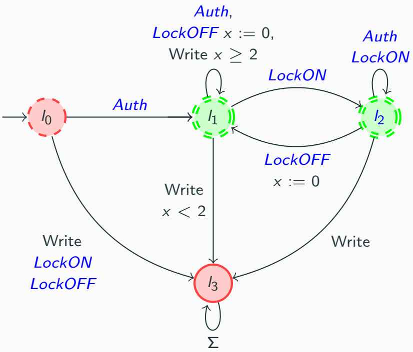
$(1, auth).(2, on).(4, w).(5, off).(6, on).(7, w)$



ϵ

Property with time: example of execution

$\Sigma_u = \{Auth, LockOFF, LockON\}, \Sigma_c = \{\text{Write}\}$



t = 4

$(1, auth).(2, on).(5, off).(8, off)$

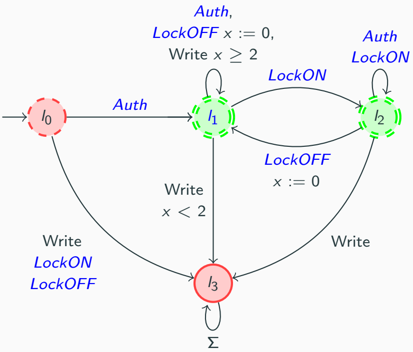
E

$(1, auth).(2, on).(4, w).(5, off).(6, on).(7, w)$

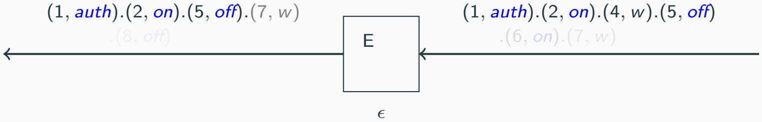
w

Property with time: example of execution

$\Sigma_u = \{Auth, LockOFF, LockON\}, \Sigma_c = \{\text{Write}\}$

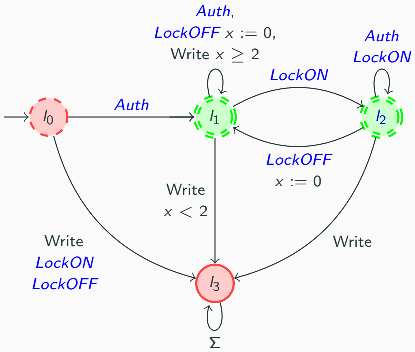


t = 5



Property with time: example of execution

$\Sigma_u = \{Auth, LockOFF, LockON\}, \Sigma_c = \{\text{Write}\}$



t = 6

(1, *auth*).(2, *on*).(5, *off*).(6, *on*) .(8, *off*)

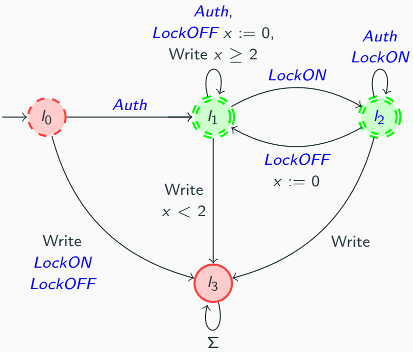
E

(1, *auth*).(2, *on*).(4, *w*).(5, *off*) .(6, *on*).(7, *w*)

w

Property with time: example of execution

$\Sigma_u = \{Auth, LockOFF, LockON\}, \Sigma_c = \{\text{Write}\}$



t = 7

(1, *auth*).(2, *on*).(5, *off*).(6, *on*) .(8, *off*)

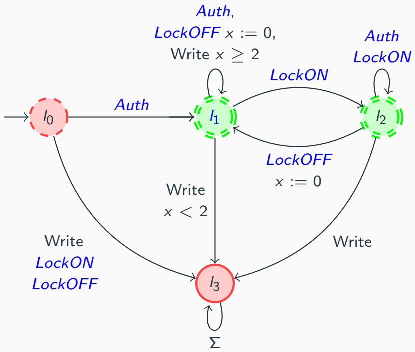
E

(1, *auth*).(2, *on*).(4, *w*).(5, *off*) .(6, *on*).(7, *w*)

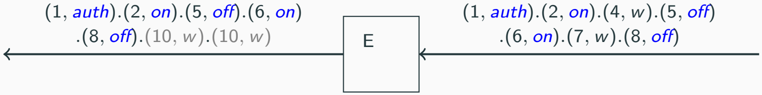
w.w

Property with time: example of execution

$\Sigma_u = \{Auth, LockOFF, LockON\}, \Sigma_c = \{\text{Write}\}$

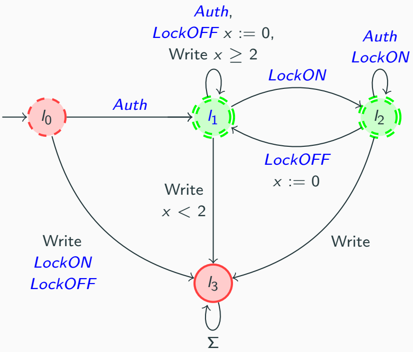


t = 8

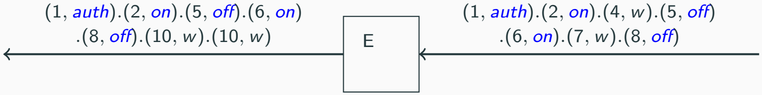


Property with time: example of execution

$\Sigma_u = \{Auth, LockOFF, LockON\}, \Sigma_c = \{\text{Write}\}$



t = 10



Outline - On the Runtime Enforcement of Timed Properties

Motivations for enforcement with time and data

Allow specifying desired behavior of a system more precisely (time constraints between events, allowing events to carry data, expressing constraints).

Specifying constraints over time and data

- *For each client with* **special** *id, 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 with* **normal** *id, after a request, there should be a response after a delay of X t.u., where X is the number of request messages..*

Many application domains

- Communication protocols.
- Managing resource allocation.
- Real-time embedded systems.
- Monitor hardware failures.
- Web services.
- Several other domains.

Enforcement Monitors

- Firewall (to prevent DOS attacks).
- Scheduler for resource allocation.

Motivations for enforcement with time and data

Allow specifying desired behavior of a system more precisely (time constraints between events, allowing events to carry data, expressing constraints).

Specifying constraints over time and data

- *For each client with* **special** *id, 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 with* **normal** *id, after a request, there should be a response after a delay of X t.u., where X is the number of request messages..*

Many application domains

- Communication protocols.
- Managing resource allocation.
- Real-time embedded systems.
- Monitor hardware failures.
- Web services.
- Several other domains.

Enforcement Monitors

- Firewall (to prevent DOS attacks).
- Scheduler for resource allocation.

Outline - On the Runtime Enforcement of Timed Properties

Parameterized Timed Automata with Variables (PTAVs)

- Extension of timed automata.
- Describes a set of *identical* timed automata (differing only in the value of its parameter p) extended with internal and external variables.
- (Inspired from IOSTS, TIOSTS, QEA, Parametric trace slicing.)

Syntax of PTAV

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

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

Parameterized Timed Automata with Variables (PTAVs)

- Extension of timed automata.
- Describes a set of *identical* timed automata (differing only in the value of its parameter p) extended with internal and external variables.
- (Inspired from IOSTS, TIOSTS, QEA, Parametric trace slicing.)

Syntax of PTAV

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

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

Parameterized Timed Automata with Variables (PTAV)

Events, timed words

- Event: $e_i = (\delta_i, a_i(\pi_i, \eta_i))$, $a_i \in \Sigma$, $\pi_i \in \mathcal{D}_p$, $\eta_i \in \mathcal{D}_V$.
- Timed word: $\sigma = (\delta_1, a_1(\pi_1, \eta_1)) \cdots (\delta_n, a_n(\pi_n, \eta_n))$.
- Instance of PTAV $A(\pi)$ accepts σ if $\sigma \in \mathcal{L}(A(\pi))$.

Projections of σ according to the runtime values of π

- $\sigma = (0.5, a(1, \eta_1)) \cdot (0.3, a(2, \eta_2)) \cdot (0.2, a(1, \eta_3)) \cdot (0.4, a(2, \eta_4))$
- $\sigma \downarrow_1 = (0.5, a(1, \eta_1)) \cdot (0.5, a(1, \eta_3))$
- $\sigma \downarrow_2 = (0.8, a(2, \eta_2)) \cdot (0.6, a(2, \eta_4))$

Parameterized Timed Automata with Variables (PTAV)

Events, timed words

- Event: $e_i = (\delta_i, a_i(\pi_i, \eta_i))$, $a_i \in \Sigma$, $\pi_i \in \mathcal{D}_p$, $\eta_i \in \mathcal{D}_V$.
- Timed word: $\sigma = (\delta_1, a_1(\pi_1, \eta_1)) \cdots (\delta_n, a_n(\pi_n, \eta_n))$.
- Instance of PTAV $A(\pi)$ accepts σ if $\sigma \in \mathcal{L}(A(\pi))$.

Projections of σ according to the runtime values of π

- $\sigma = (0.5, a(1, \eta_1)) \cdot (0.3, a(2, \eta_2)) \cdot (0.2, a(1, \eta_3)) \cdot (0.4, a(2, \eta_4))$
- $\sigma \downarrow_1 = (0.5, a(1, \eta_1)) \cdot (0.5, a(1, \eta_3))$
- $\sigma \downarrow_2 = (0.8, a(2, \eta_2)) \cdot (0.6, a(2, \eta_4))$

Parameterized Timed Automata with Variables (PTAV)

Events, timed words

- Event: $e_i = (\delta_i, a_i(\pi_i, \eta_i))$, $a_i \in \Sigma$, $\pi_i \in \mathcal{D}_p$, $\eta_i \in \mathcal{D}_V$.
- Timed word: $\sigma = (\delta_1, a_1(\pi_1, \eta_1)) \cdots (\delta_n, a_n(\pi_n, \eta_n))$.
- Instance of PTAV $A(\pi)$ accepts σ if $\sigma \in \mathcal{L}(A(\pi))$.

Projections of σ according to the runtime values of π

- $\sigma = (0.5, a(1, \eta_1)) \cdot (0.3, a(2, \eta_2)) \cdot (0.2, a(1, \eta_3)) \cdot (0.4, a(2, \eta_4))$
- $\sigma \downarrow_1 = (0.5, a(1, \eta_1)) \cdot (0.5, a(1, \eta_3))$
- $\sigma \downarrow_2 = (0.8, a(2, \eta_2)) \cdot (0.6, a(2, \eta_4))$

Parameterized Timed Automata with Variables (PTAV)

Events, timed words

- Event: $e_i = (\delta_i, a_i(\pi_i, \eta_i))$, $a_i \in \Sigma$, $\pi_i \in \mathcal{D}_p$, $\eta_i \in \mathcal{D}_V$.
- Timed word: $\sigma = (\delta_1, a_1(\pi_1, \eta_1)) \cdots (\delta_n, a_n(\pi_n, \eta_n))$.
- Instance of PTAV $A(\pi)$ accepts σ if $\sigma \in \mathcal{L}(A(\pi))$.

Projections of σ according to the runtime values of π

- $\sigma = (0.5, a(1, \eta_1)) \cdot (0.3, a(2, \eta_2)) \cdot (0.2, a(1, \eta_3)) \cdot (0.4, a(2, \eta_4))$
- $\sigma \downarrow_1 = (0.5, a(1, \eta_1)) \cdot (0.5, a(1, \eta_3))$
- $\sigma \downarrow_2 = (0.8, a(2, \eta_2)) \cdot (0.6, a(2, \eta_4))$

Parameterized Timed Automata with Variables (PTAV)

Events, timed words

- Event: $e_i = (\delta_i, a_i(\pi_i, \eta_i))$, $a_i \in \Sigma$, $\pi_i \in \mathcal{D}_p$, $\eta_i \in \mathcal{D}_V$.
- Timed word: $\sigma = (\delta_1, a_1(\pi_1, \eta_1)) \cdots (\delta_n, a_n(\pi_n, \eta_n))$.
- Instance of PTAV $A(\pi)$ accepts σ if $\sigma \in \mathcal{L}(A(\pi))$.

Projections of σ according to the runtime values of π

- $\sigma = (0.5, a(1, \eta_1)) \cdot (0.3, a(2, \eta_2)) \cdot (0.2, a(1, \eta_3)) \cdot (0.4, a(2, \eta_4))$
- $\sigma \downarrow_1 = (0.5, a(1, \eta_1)) \cdot (0.5, a(1, \eta_3))$
- $\sigma \downarrow_2 = (0.8, a(2, \eta_2)) \cdot (0.6, a(2, \eta_4))$

Parameterized Timed Automata with Variables (PTAV)

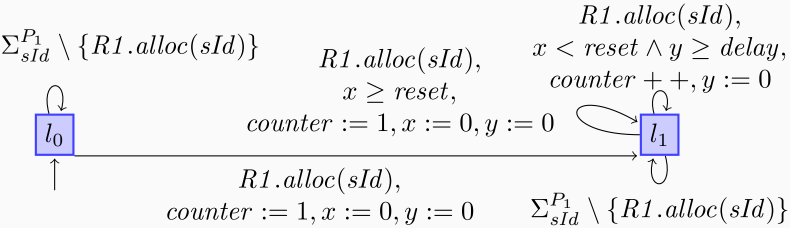
Events, timed words

- Event: $e_i = (\delta_i, a_i(\pi_i, \eta_i))$, $a_i \in \Sigma$, $\pi_i \in \mathcal{D}_p$, $\eta_i \in \mathcal{D}_V$.
- Timed word: $\sigma = (\delta_1, a_1(\pi_1, \eta_1)) \cdots (\delta_n, a_n(\pi_n, \eta_n))$.
- Instance of PTAV $A(\pi)$ accepts σ if $\sigma \in \mathcal{L}(A(\pi))$.

Projections of σ according to the runtime values of π

- $\sigma = (0.5, a(1, \eta_1)) \cdot (0.3, a(2, \eta_2)) \cdot (0.2, a(1, \eta_3)) \cdot (0.4, a(2, \eta_4))$
- $\sigma \downarrow_1 = (0.5, a(1, \eta_1)) \cdot (0.5, a(1, \eta_3))$
- $\sigma \downarrow_2 = (0.8, a(2, \eta_2)) \cdot (0.6, a(2, \eta_4))$

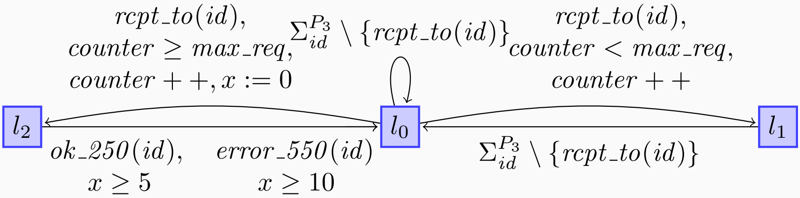
Examples of PTAVs for resource allocation [2]



"There should be a dynamic delay between two allocation requests to the same resource by a service. This delay increases as the number of allocations increases and also depends on the service id."

[2] Squares denote accepting locations. Non-accepting locations are omitted

Examples of PTAVs for resource allocation (ctd) [3]



If

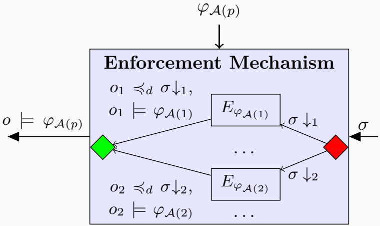
- the number of RCPT_TO messages is greater than *maxreq*, and
- the response of the server is OK_250 (resp. ERROR_550),

then there should be a delay of at least 5 (resp. 10) t.u. before sending the response.

[3] Squares denote accepting locations. Non-accepting locations are omitted

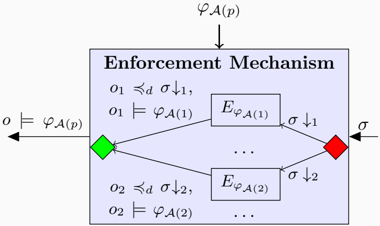
Outline - On the Runtime Enforcement of Timed Properties

Enforcement of parametric timed properties



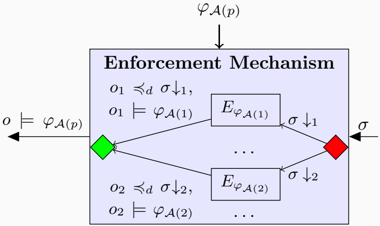
- Input/output timed words: $\sigma = (\delta_1, a_1(\pi, \eta_1)) \cdots (\delta_n, a_n(\pi, \eta_n))$.
- Property φ specified by a PTAV.
- An instance of EM per parameter value (takes as input only the events with same parameter value).
- Output of each EM instance satisfies the soundness, transparency and optimality constraints.
- $o = merge(o_1, o_2)$ is only sound (order of events may not be preserved globally).

Enforcement of parametric timed properties



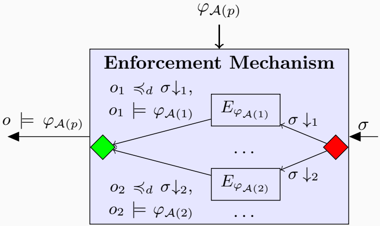
- Input/output timed words: $\sigma = (\delta_1, a_1(\pi, \eta_1)) \cdots (\delta_n, a_n(\pi, \eta_n))$.
- Property φ specified by a PTAV.
- An instance of EM per parameter value (takes as input only the events with same parameter value).
- Output of each EM instance satisfies the soundness, transparency and optimality constraints.
- $o = merge(o_1, o_2)$ is only sound (order of events may not be preserved globally).

Enforcement of parametric timed properties



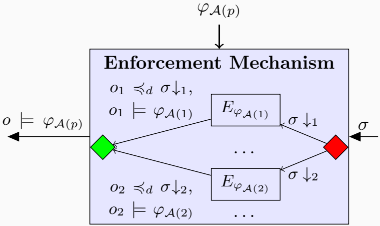
- Input/output timed words: $\sigma = (\delta_1, a_1(\pi, \eta_1)) \cdots (\delta_n, a_n(\pi, \eta_n))$.
- Property φ specified by a PTAV.
- An instance of EM per parameter value (takes as input only the events with same parameter value).
- Output of each EM instance satisfies the soundness, transparency and optimality constraints.
- $o = merge(o_1, o_2)$ is only sound (order of events may not be preserved globally).

Enforcement of parametric timed properties



- Input/output timed words: $\sigma = (\delta_1, a_1(\pi, \eta_1)) \cdots (\delta_n, a_n(\pi, \eta_n))$.
- Property φ specified by a PTAV.
- An instance of EM per parameter value (takes as input only the events with same parameter value).
- Output of each EM instance satisfies the soundness, transparency and optimality constraints.
- $o = merge(o_1, o_2)$ is only sound (order of events may not be preserved globally).

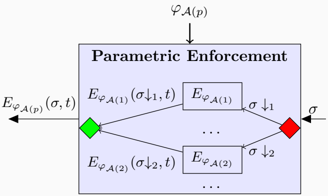
Enforcement of parametric timed properties



- Input/output timed words: $\sigma = (\delta_1, a_1(\pi, \eta_1)) \cdots (\delta_n, a_n(\pi, \eta_n))$.
- Property φ specified by a PTAV.
- An instance of EM per parameter value (takes as input only the events with same parameter value).
- Output of each EM instance satisfies the soundness, transparency and optimality constraints.
- $o = merge(o_1, o_2)$ is only sound (order of events may not be preserved globally).

Requirements: Enforcement of parametric timed properties

$$E_{\varphi_p}(\sigma, t) : (\mathbb{R}_{\geq 0} \times \Lambda)^* \times \mathbb{R}_{\geq 0} \rightarrow (\mathbb{R}_{\geq 0} \times \Lambda)^*$$



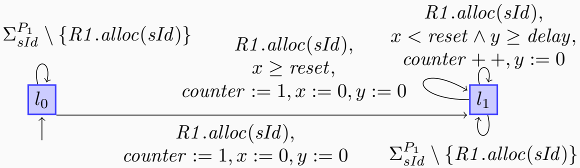
Parametric Soundness, Transparency and Optimality

- Soundness: $\forall \pi \in \mathcal{D}om(p), \forall \sigma \in (\mathbb{R}_{\geq 0} \times \Lambda)^* : sound(E_{\varphi_{\mathcal{A}(\pi)}}, \sigma\downarrow_\pi)$
- Transparency: $\forall \pi \in \mathcal{D}om(p), \forall \sigma \in (\mathbb{R}_{\geq 0} \times \Lambda)^* : transparent(E_{\varphi_{\mathcal{A}(\pi)}}, \sigma\downarrow_\pi)$
- Optimality: $\forall \pi \in \mathcal{D}om(p), \forall \sigma \in (\mathbb{R}_{\geq 0} \times \Lambda)^* : optimal(E_{\varphi_{\mathcal{A}(\pi)}}, \sigma\downarrow_\pi)$

Proposition

Given a safety PTAV $\mathcal{A}(p)$ specifying property $\varphi_{\mathcal{A}(p)}$, for all $\pi \in D_p$, the enforcement function $E_{\varphi_{\mathcal{A}(\pi)}}$, is sound, transparent, and optimal w.r.t. $\mathcal{A}(\pi)$.

Parametric case: Example



- *delay* $= 5$
- *reset* $= 100$

- Let $\sigma = (2, R1.alloc(1)) \cdot (1, R1.alloc(2)) \cdot (1, R1.alloc(1))$.
- $\mathcal{D}om(\sigma) = \{1, 2\}$, we have two monitor instances.

$\Pi = 1$

- $\sigma \downarrow_1 = (2, R1.alloc(1)) \cdot (2, R1.alloc(1))$.
- $E_{\varphi_{\mathcal{A}(1)}} = (2, R1.alloc(1)) \cdot (5, R1.alloc(1))$.

$\Pi = 2$

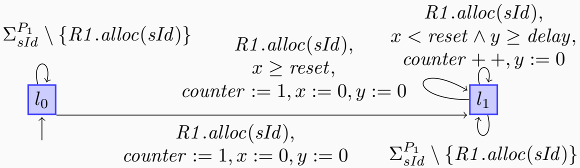
- $\sigma \downarrow_2 = (3, R1.alloc(2))$.
- $E_{\varphi_{\mathcal{A}(2)}} = (3, R1.alloc(2))$.

$E_{\varphi_{\mathcal{A}(1)}}$ and $E_{\varphi_{\mathcal{A}(2)}}$ are sound, transparent, and optimal.

Remark: properties of the global output trace

- $E_{\varphi_{\mathcal{A}(p)}}(\sigma, t) = merge(E_{\varphi_{\mathcal{A}(1)}}(\sigma\downarrow_1, t), E_{\varphi_{\mathcal{A}(2)}}(\sigma\downarrow_2, t))$
- Only soundness is preserved.

Parametric case: Example



- *delay* $= 5$
- *reset* $= 100$

- Let $\sigma = (2, R1.alloc(1)) \cdot (1, R1.alloc(2)) \cdot (1, R1.alloc(1))$.
- $\mathcal{D}om(\sigma) = \{1, 2\}$, we have two monitor instances.

$\Pi = 1$

- $\sigma \downarrow_1 = (2, R1.alloc(1)) \cdot (2, R1.alloc(1))$.
- $E_{\varphi_{\mathcal{A}(1)}} = (2, R1.alloc(1)) \cdot (5, R1.alloc(1))$.

$\Pi = 2$

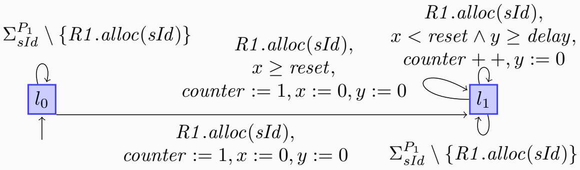
- $\sigma \downarrow_2 = (3, R1.alloc(2))$.
- $E_{\varphi_{\mathcal{A}(2)}} = (3, R1.alloc(2))$.

$E_{\varphi_{\mathcal{A}(1)}}$ and $E_{\varphi_{\mathcal{A}(2)}}$ are sound, transparent, and optimal.

Remark: properties of the global output trace

- $E_{\varphi_{\mathcal{A}(p)}}(\sigma, t) = merge(E_{\varphi_{\mathcal{A}(1)}}(\sigma\downarrow_1, t), E_{\varphi_{\mathcal{A}(2)}}(\sigma\downarrow_2, t))$
- Only soundness is preserved.

Parametric case: Example



- $delay = 5$
- $reset = 100$

- Let $\sigma = (2, R1.alloc(1)) \cdot (1, R1.alloc(2)) \cdot (1, R1.alloc(1))$.
- $\mathcal{D}om(\sigma) = \{1, 2\}$, we have two monitor instances.

$\Pi = 1$

- $\sigma \downarrow_1 = (2, R1.alloc(1)) \cdot (2, R1.alloc(1))$.
- $E_{\varphi_{\mathcal{A}(1)}} = (2, R1.alloc(1)) \cdot (5, R1.alloc(1))$.

$\Pi = 2$

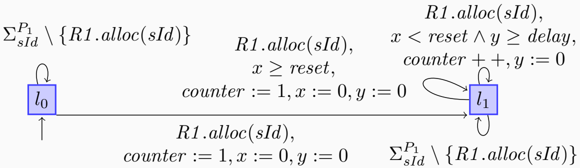
- $\sigma \downarrow_2 = (3, R1.alloc(2))$.
- $E_{\varphi_{\mathcal{A}(2)}} = (3, R1.alloc(2))$.

$E_{\varphi_{\mathcal{A}(1)}}$ and $E_{\varphi_{\mathcal{A}(2)}}$ are sound, transparent, and optimal.

Remark: properties of the global output trace

- $E_{\varphi_{\mathcal{A}(p)}}(\sigma, t) = merge(E_{\varphi_{\mathcal{A}(1)}}(\sigma\downarrow_1, t), E_{\varphi_{\mathcal{A}(2)}}(\sigma\downarrow_2, t))$
- Only soundness is preserved.

Parametric case: Example



- $delay = 5$
- $reset = 100$

- Let $\sigma = (2, R1.alloc(1)) \cdot (1, R1.alloc(2)) \cdot (1, R1.alloc(1))$.
- $\mathcal{D}om(\sigma) = \{1, 2\}$, we have two monitor instances.

$\Pi = 1$

- $\sigma \downarrow_1 = (2, R1.alloc(1)) \cdot (2, R1.alloc(1))$.
- $E_{\varphi_{\mathcal{A}(1)}} = (2, R1.alloc(1)) \cdot (5, R1.alloc(1))$.

$\Pi = 2$

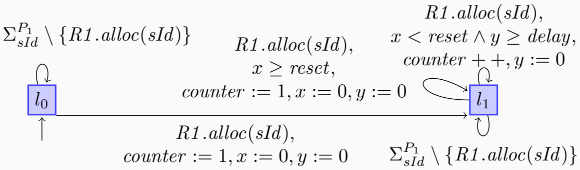
- $\sigma \downarrow_2 = (3, R1.alloc(2))$.
- $E_{\varphi_{\mathcal{A}(2)}} = (3, R1.alloc(2))$.

$E_{\varphi_{\mathcal{A}(1)}}$ and $E_{\varphi_{\mathcal{A}(2)}}$ are sound, transparent, and optimal.

Remark: properties of the global output trace

- $E_{\varphi_{\mathcal{A}(p)}}(\sigma, t) = merge(E_{\varphi_{\mathcal{A}(1)}}(\sigma\downarrow_1, t), E_{\varphi_{\mathcal{A}(2)}}(\sigma\downarrow_2, t))$
- Only soundness is preserved.

Parametric case: Example



- $delay = 5$
- $reset = 100$

- Let $\sigma = (2, R1.alloc(1)) \cdot (1, R1.alloc(2)) \cdot (1, R1.alloc(1))$.
- $\mathcal{D}om(\sigma) = \{1, 2\}$, we have two monitor instances.

$\Pi = 1$

- $\sigma \downarrow_1 = (2, R1.alloc(1)) \cdot (2, R1.alloc(1))$.
- $E_{\varphi_{\mathcal{A}(1)}} = (2, R1.alloc(1)) \cdot (5, R1.alloc(1))$.

$\Pi = 2$

- $\sigma \downarrow_2 = (3, R1.alloc(2))$.
- $E_{\varphi_{\mathcal{A}(2)}} = (3, R1.alloc(2))$.

$E_{\varphi_{\mathcal{A}(1)}}$ and $E_{\varphi_{\mathcal{A}(2)}}$ are sound, transparent, and optimal.

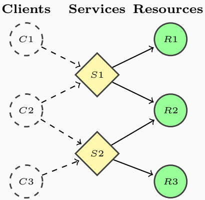
Remark: properties of the global output trace

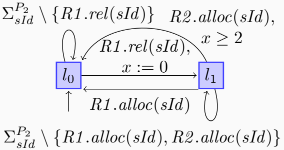
- $E_{\varphi_{\mathcal{A}(p)}}(\sigma, t) = merge(E_{\varphi_{\mathcal{A}(1)}}(\sigma\downarrow_1, t), E_{\varphi_{\mathcal{A}(2)}}(\sigma\downarrow_2, t))$
- Only soundness is preserved.

Outline - On the Runtime Enforcement of Timed Properties

Application Domains

Resource allocation in a client-server model





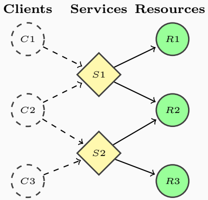
After releasing R1, there should be a delay of at least 2 t.u. before allocating R2.

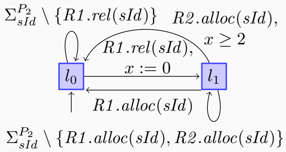
Protecting mail servers

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.

Application Domains

Resource allocation in a client-server model

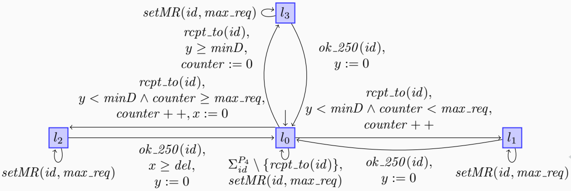




After releasing R1, there should be a delay of at least 2 t.u. before allocating R2.

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.



Conclusions and Future Work

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).
- Enforcement Mechanisms are **delayers** (with several *alternative enforcement primitives*: suppress actions, augment/reduce delays).
- Exhibiting a notion of non-enforceable properties.
- Requirements with constraints on *data and time*.

Future Work

- Delineate the set of *enforceable* response properties.
- *More expressive* formalisms such as context-free timed languages.
- Probabilistic models for events.
- Implementing efficient enforcement monitors (in application scenarios).

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).
- Enforcement Mechanisms are **delayers** (with several *alternative enforcement primitives*: suppress actions, augment/reduce delays).
- Exhibiting a notion of non-enforceable properties.
- Requirements with constraints on *data and time*.

Future Work

- Delineate the set of *enforceable* response properties.
- *More expressive* formalisms such as context-free timed languages.
- Probabilistic models for events.
- Implementing efficient enforcement monitors (in application scenarios).

Software Verification and Testing at SAC 2018

Consider submitting to the **Software Verification and Testing** at SAC 2018!

April 9 – 13, 2018 in Pau, France

http://sac-svt-2018.imag.fr

Important Dates:

- Sept 15, 2017: Submission of regular papers and SRC research abstracts
- Nov 10, 2017: Notification of paper and SRC acceptance/rejection
- April 9 – 13, 2018: SAC and SVT day

References on Runtime Enforcement (of Timed) Properties

- First definition of mechanisms for safety and co-safety properties: [PinisettyFJMRN12].
- Extension to regular properties: [PinisettyFJM14a].
- Enforcement mechanism as delayers (preserving delay between events): [PinisettyFJMRN14] (summarizing [PinisettyFJMRN12, PinisettyFJM14a])
- Events with Data: [PinisettyFJM14b].
- Uncontrollable events: [RenardFRPJM15].
- Enforcement mechanism as delayers+suppression (reducing delay between events): [FalconeJMP16].
- Using Game Theory to synthesize mechanisms

References on Runtime Enforcement (of Untimed Properties)

General Models of Enforcement Mechanisms:

- Security Automata (SAs): [Schneider00].
- Edit Automata (EAs): [LigattiBW05, LigattiBW09, LigattiThesis].
- Generic Enforcement Monitors (GEMs): [FalconeMFR11].

Models taking memory limitations into account:

- Shallow History Automata: [Fong04].
- Finite Edit Automata: [BeauquierCL09].
- Limiting the amount of Memory: [TalhiTD06].

Synthesis of Enforcement Mechanisms:

- Synthesis from Process Algebraic Descriptions: [MartinelliM07].
- Synthesizing GEMs from Streett Automata: [FalconeFM08].
- Synthesizing SAs from Rabin Automata: [ChabotKT09].
- Synthesizing GEMs from Safety-Progress Properties: [FalconeFM12].

Enforceable properties:
[Schneider00, HamelnMS06, LigattiThesis, BielovaM08, FalconeFM12].

References on Runtime Verification

Tutorials and surveys:

- [PlatnerN81]
- [SchroederB95]
- [ColinM05]
- [HavelundG08]
- [LeuckerS08]
- [FalconeHR13]

References i

Danièle Beauquier, Joëlle Cohen, and Ruggero Lanotte.
Security policies enforcement using finite edit automata.
Electr. Notes Theor. Comput. Sci., 229(3):19–35, 2009.

Nataliia Bielova and Fabio Massacci.
Do you really mean what you actually enforced?
In *FAST'08: 5th International Workshop on Formal Aspects in Security and Trust. Revised Selected Papers*, pages 287–301, 2008.

Hugues Chabot, Raphael Khoury, and Nadia Tawbi.
Generating in-line monitors for Rabin automata.
In *NordSec'09: 14th Nordic Conf. on Secure IT Systems*, pages 287–301, 2009.

References ii

Severine Colin and Leonardo Mariani.
Run-time verification.
In *Model-based Testing of Reactive Systems*, volume 3472 of *LNCS*, pages 525–556, 2005.

Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier.
Synthesizing enforcement monitors wrt. the safety-progress classification of properties.
In *ICISS'08: Proceedings of the* 4^{th} *International Conference on Information Systems Security*, pages 41–55, 2008.

Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier.
Enforcement monitoring wrt. the safety-progress classification of properties.
In *SAC '09: Proceedings of the ACM symposium on Applied Computing*, pages 593–600, 2009.

References iii

Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier.
What can you verify and enforce at runtime?
STTT, 14(3):349–382, 2012.

Yliès Falcone, Klaus Havelund, and Giles Reger.
A tutorial on runtime verification.
In Manfred Broy, Doron A. Peled, and Georg Kalus, editors, *Engineering Dependable Software Systems*, volume 34 of *NATO Science for Peace and Security Series, D: Information and Communication Security*, pages 141–175. IOS Press, 2013.

Yliès Falcone, Thierry Jéron, Hervé Marchand, and Srinivas Pinisetty.
Runtime enforcement of regular timed properties by suppressing and delaying events.
Science of Computer Programming, 123(3):2–41, 2016.

References iv

Yliès Falcone, Laurent Mounier, Jean-Claude Fernandez, and Jean-Luc Richier.
Runtime enforcement monitors: composition, synthesis, and enforcement abilities.
Formal Methods in System Design, 38(3):223–262, 2011.

Philip W. L. Fong.
Access control by tracking shallow execution history.
In *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, pages 43–55, 2004.

Kevin W. Hamlen, Greg Morrisett, and Fred B. Schneider.
Computability classes for enforcement mechanisms.
ACM Trans. Programming Lang. and Syst., 28(1):175–205, 2006.

References v

Klaus Havelund and Allen Goldberg.
Verify your runs.
Verified Software: Theories, Tools, Experiments: First IFIP TC 2/WG 2.3 Conference, VSTTE 2005. Revised Selected Papers and Discussions, pages 374–383, 2008.

Martin Leucker and Christian Schallhart.
A brief account of runtime verification.
Journal of Logic and Algebraic Programming, 78(5):293–303, may/june 2008.

Jay Ligatti, Lujo Bauer, and David Walker.
Enforcing non-safety security policies with program monitors.
In *ESORICS'05 Proceedings of the* 10^{th} *European Symposium on Research in Computer Security*, pages 355–373, 2005.

References vi

Jay Ligatti, Lujo Bauer, and David Walker.
Run-time enforcement of nonsafety policies.
ACM Transaction Information System Security., 12(3), 2009.

Jarred Adam Ligatti.
Policy Enforcement via Program Monitoring.
PhD thesis, Princeton University, June 2006.

Fabio Martinelli and Ilaria Matteucci.
Through modeling to synthesis of security automata.
Electronic Notes in Theoritical Compututer Science, 179:31–46, 2007.

Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, and Hervé Marchand.
Runtime enforcement of regular timed properties.
In Yookun Cho, Sung Y. Shin, Sang-Wook Kim, Chih-Cheng Hung, and Jiman Hong, editors, *Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014*, pages 1279–1286. ACM, 2014.

References vii

Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, and Hervé Marchand.
Runtime enforcement of parametric timed properties with practical applications.
In Jean-Jacques Lesage, Jean-Marc Faure, José E. R. Cury, and Bengt Lennartson, editors, *12th International Workshop on Discrete Event Systems, WODES 2014, Cachan, France, May 14-16, 2014.*, pages 420–427. International Federation of Automatic Control, 2014.

Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, Hervé Marchand, Antoine Rollet, and Omer Landry Nguena-Timo.
Runtime enforcement of timed properties.
In *Runtime Verification, Third International Conference, RV 2012, Istanbul, Turkey, September 25-28, 2012, Revised Selected Papers*, pages 229–244, 2012.

References viii

Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, Hervé Marchand, Antoine Rollet, and Omer Nguena-Timo.
Runtime enforcement of timed properties revisited.
Formal Methods in System Design, 45(3):381–422, 2014.

B. Plattner and J. Nievergelt.
Special feature: Monitoring program execution: A survey.
Computer, 14(11):76–93, 1981.

Matthieu Renard, Yliès Falcone, Antoine Rollet, Srinivas Pinisetty, Thierry Jéron, and Hervé Marchand.
Enforcement of (timed) properties with uncontrollable events.
In Martin Leucker, Camilo Rueda, and Frank D. Valencia, editors, *Theoretical Aspects of Computing - ICTAC 2015 - 12th International Colloquium Cali, Colombia, October 29-31, 2015, Proceedings*, volume 9399 of *Lecture Notes in Computer Science*, pages 542–560. Springer, 2015.

References ix

Fred B. Schneider.
Enforceable security policies.
ACM Transactions on Information and System Security, 3(1), 2000.

Beth A. Schroeder.
On-line monitoring: A tutorial.
Computer, 28(6):72–78, 1995.

Chamseddine Talhi, Nadia Tawbi, and Mourad Debbabi.
Execution monitoring enforcement for limited-memory systems.
In *PST'06: Proceedings of the International Conference on Privacy, Security and Trust*, pages 1–12, 2006.