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 $\varphi$.
- 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 $\varphi$.
- 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 $\varphi$.
- 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 \( \varphi \).
- Possibly augmented with a memorization mechanism.

\[\begin{array}{c}
\text{Enforcement Monitor} \\
\text{Memory}
\end{array}\]

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 \( E : \Sigma^* \rightarrow \Sigma^* \).
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)

\[ \sigma \xrightarrow{EM} S \quad EM(\sigma) \]

Output sanitiser (OS)

\[ S \xrightarrow{\sigma} EM \quad EM(\sigma) \]

Reference monitor (RM)

\[ S \xrightarrow{\sigma} EM \]

Sys 1

Sys 2

file system

input sanitiser

reference monitor

output sanitiser
Outline - On the Runtime Enforcement of Timed Properties

On the Runtime Enforcement of Untimed Properties

Specifying Timed Properties

Runtime Enforcement of Timed Properties

Extensions

Conclusions and Future Work
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

→ the memorization mechanism should be designed wrt. those constraints

Definition (Property enforcement)

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

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

$\forall \sigma \in \Sigma^*: \exists o \in \Sigma^* : E(\sigma) = o \land \varphi(o)$ (1)

$(1) \land \varphi(\sigma) \implies \sigma = o \land \forall i < |\sigma| : E(\sigma_{\cdot\cdot\cdot}i) = \sigma_{\cdot\cdot\cdot}i$ (2)

$(1) \land \varphi(\sigma) \implies \sigma = o \land \forall i < |\sigma|, \exists j \leq i : E(\sigma_{\cdot\cdot\cdot}i) = \sigma_{\cdot\cdot\cdot}j$ (3)

$(1) \land \varphi(\sigma) \implies \sigma \approx o$ (4)
Property Enforcement [Schneider00, LigattiBW05, BielovaM08, FalconeFM09a]

Relation between the input and output sequences should adhere:

- **soundness**: the output sequences should be correct with respect to the property
- **transparency**: the correct input sequences should not be modified

→ the memorization mechanism should be designed with respect to those constraints

**Definition (Property enforcement)**

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

- conservatively (1)
- precisely (2)
- delayed-precisely (3)
- effectively with respect to the equivalence relation $\approx$ (4)

\[ \forall \sigma \in \Sigma^* : \\
(1) \land \varphi(\sigma) \implies \sigma = o \land \forall i < |\sigma| : E(\sigma_{i..}) = \sigma_{i..} \] (2)

\[ (1) \land \varphi(\sigma) \implies \sigma = o \land \forall i < |\sigma|, \exists j \leq i : E(\sigma_{i..}) = \sigma_{j..} \] (3)

\[ (1) \land \varphi(\sigma) \implies \sigma \approx o \] (4)
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
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 $\text{Pref}(a \cdot b \cdot c \cdot d) \cup \text{Pref}(b \cdot a \cdot d \cdot c)$
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 $\text{Pref}(a \cdot b \cdot c \cdot d) \cup \text{Pref}(b \cdot a \cdot d \cdot c)$
Security Automata (SA) and Decidable Safety Properties

SA cannot take decisions based on possible future executions \( \Leftarrow \) 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 → 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
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
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 (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:

$$\forall \sigma \in \Sigma^\infty : \ (\varphi(\sigma) \Leftrightarrow \forall \sigma' \in \Sigma^* : \sigma' < \sigma \Rightarrow \exists \sigma'' : \sigma' \leq \sigma'' < \sigma \land \varphi(\sigma''))$$
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:

\[ \forall \sigma \in \Sigma^{\infty} : (\varphi(\sigma) \iff \forall \sigma' \in \Sigma^* : \sigma' \prec \sigma \Rightarrow \exists \sigma'' : \sigma' \preceq \sigma'' \prec \sigma \land \varphi(\sigma'')) \]
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 $\Sigma^\infty$ if:

$$\forall \sigma \in \Sigma^\infty : \ (\varphi(\sigma) \iff \forall \sigma' \in \Sigma^*: \sigma' < \sigma \Rightarrow \exists \sigma'' : \sigma' \preceq \sigma'' < \sigma \land \varphi(\sigma''))$$
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 $A\downarrow$ is a 4-tuple $(Q^{A\downarrow}, q_{init}^{A\downarrow}, \rightarrow^{A\downarrow}, Ops)$ wrt. $\Sigma$ parameterized by $Ops$

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

Advantages:

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

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

A GEM $\mathcal{A}_↓$ is a 4-tuple $(Q^{\mathcal{A}_↓}, q^{\mathcal{A}_↓}_{\text{init}}, \rightarrow^{\mathcal{A}_↓}, \text{Ops})$ wrt. $\Sigma$ parameterized by $\text{Ops}$

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

Advantages:

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

Example (Enforcement of a finitary property)

\[ \text{Pref}(a \cdot b \cdot c \cdot d) \cup \text{Pref}(b \cdot a \cdot d \cdot c) \]

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)

\[ \text{Pref}(a \cdot b \cdot c \cdot d) \cup \text{Pref}(b \cdot a \cdot d \cdot c) \]

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 \textit{r_auth} should be:

1. written in a log file
2. answered
   - either with a \textit{g_auth} or a \textit{d_auth}
   - without any \textit{ops} or \textit{r_auth} meanwhile
Instantiated GEMs: some examples

Example (Enforcement of a finitary property)

\[ \text{Pref}(a \cdot b \cdot c \cdot d) \cup \text{Pref}(b \cdot a \cdot d \cdot c) \]

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 \( r\text{-auth} \) should be:

1. written in a log file
2. answered
   - either with a \( g\text{-auth} \) or a \( d\text{-auth} \)
   - without any \( op_s \) or \( r\text{-auth} \) meanwhile
Theorem (Enf. ability of instantiated GEMs [FalconeFM09a])

GEMs instantiated by \{halt, store, dump, 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:
  \[ \sigma = (\delta_1, a_1) \cdot (\delta_2, a_2) \cdot \ldots \cdot (\delta_n, a_n), \delta_i \in \mathbb{R}_{\geq 0}, a_i \in \Sigma. \]
- Property:
  - defined by a regular timed language \( \varphi \subseteq (\mathbb{R}_{\geq 0} \times \Sigma)^* \),
  - specified by a timed automaton (TA) \( A_{\varphi} \).
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) \( A_\varphi \).
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) \( A_\varphi \).
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) \( A_\varphi \).
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) \( A_{\varphi} \).
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).

- **Σ ⊇ {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.

- $\Sigma \supseteq \{\text{req}, \text{gr}\}$
- “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

\[
\Sigma \setminus \{req, gr\} \quad req, \quad x := 0 \\
\Sigma \setminus \{req, gr\} \quad \Sigma \setminus \{req, gr\} \quad \Sigma \setminus \{gr\}; \\
gr, x < 15 \lor x > 20 \\
gr, 15 \leq x \leq 20; \\
x := 0 \\
\Sigma
\]

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

\(\epsilon \models \varphi\).

\((3, req) \nvdash \varphi\).

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

\((3, req) \cdot (15, gr) \cdot (5, req) \nvdash \varphi\).

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

Remark: response properties are neither prefix nor extension closed.
Example: response property

Let \( \Sigma \) be the alphabet \( \{\text{req, gr}\} \) and let \( \Sigma \) be the set of input time words.

\[
\begin{align*}
\Sigma & = \{\text{req, gr}\} \\
\text{Input time word:} & \quad (3, \text{req}) \cdot (15, \text{gr}) \cdot (5, \text{req}) \cdot (19, \text{gr})
\end{align*}
\]

- \( \epsilon \models \varphi \).
- \( (3, \text{req}) \not\models \varphi \).
- \( (3, \text{req}) \cdot (15, \text{gr}) \models \varphi \).
- \( (3, \text{req}) \cdot (15, \text{gr}) \cdot (5, \text{req}) \not\models \varphi \).
- \( (3, \text{req}) \cdot (15, \text{gr}) \cdot (5, \text{req}) \cdot (19, \text{gr}) \models \varphi \).

Remark: response properties are neither prefix nor extension closed.
Example: response property

$\Sigma \setminus \{req, gr\}$

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

- 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

\[ \Sigma \setminus \{req, gr\} \]

\[ l_0 \xrightarrow{req, x := 0} l_1 \]

\[ l_1 \xrightarrow{\Sigma \setminus \{req, gr\}} l_2 \]

\[ l_2 \xrightarrow{\Sigma \setminus \{gr\}; gr, x < 15 \lor x > 20} \]

\[ \Sigma \]

\[ \Sigma = \{req, gr\} \]

\[ \bullet \text{ Input time word:} \]

\[ (3, req) \cdot (15, gr) \cdot (5, req) \cdot (19, gr) \]

\[ \epsilon \models \varphi. \]

\[ (3, req) \nvdash \varphi. \]

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

\[ (3, req) \cdot (15, gr) \cdot (5, req) \nvdash \varphi. \]

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

Remark: response properties are neither prefix nor extension closed.
Example: response property

\[ \Sigma \setminus \{\text{req, gr}\} \]

- Input time word:
  \[(3, \text{req}) \cdot (15, \text{gr}) \cdot (5, \text{req}) \cdot (19, \text{gr})\]

- \(\epsilon \models \varphi\).
- \((3, \text{req}) \not\models \varphi\).
- \((3, \text{req}) \cdot (15, \text{gr}) \models \varphi\).
- \((3, \text{req}) \cdot (15, \text{gr}) \cdot (5, \text{req}) \not\models \varphi\).
- \((3, \text{req}) \cdot (15, \text{gr}) \cdot (5, \text{req}) \cdot (19, \text{gr}) \models \varphi\).

Remark: response properties are neither prefix nor extension closed.
RE of Timed Properties
Outline - On the Runtime Enforcement of Timed Properties

On the Runtime Enforcement of Untimed Properties

Specifying Timed Properties

Runtime Enforcement of Timed Properties
  Requirements on an Enforcement Mechanism
  Functional Definition of an Enforcement Mechanism
  Operational Description of an Enforcement Mechanism
  Algorithmic Description of an Enforcement Mechanism
  A note on Non-enforceable Properties

Extensions

Conclusions and Future Work
Problem statement

Given some (regular) timed property $\varphi$:

$$o \models \varphi$$

What can an enforcement mechanism do?

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

How can we obtain an enforcement mechanism as a “delayer” with suppression for $\varphi$. 
Problem tackled and Contributions

\[ \varphi \text{ is a timed property} \]

A formal framework for runtime enforcement of timed properties

- Any regular timed property \( \varphi \) as input.
- Enforcement mechanisms i) add additional delays between actions to satisfy \( \varphi \), 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

\[
\sigma \preceq_d \sigma \\
\sigma \models \varphi!
\]

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

\[ o \preceq_d \sigma \]
\[ o \models \varphi ! \]

A formal framework for runtime enforcement of timed properties

- Any regular timed property \( \varphi \) as input.
- Enforcement mechanisms i) add additional delays between actions to satisfy \( \varphi \), 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

\( \varphi \) is a timed property

A formal framework for runtime enforcement of timed properties

- Any regular timed property \( \varphi \) as input.
- Enforcement mechanisms i) add additional delays between actions to satisfy \( \varphi \), 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

\( \varphi \) is a timed property

\[ o \preceq_d \sigma \]

\[ o \models \varphi ! \]

A formal framework for runtime enforcement of timed properties

- Any regular timed property \( \varphi \) as input.
- Enforcement mechanisms i) add additional delays between actions to satisfy \( \varphi \), 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

$\varphi$ is a timed property

A formal framework for runtime enforcement of timed properties

- Any regular timed property $\varphi$ as input.
- Enforcement mechanisms i) add additional delays between actions to satisfy $\varphi$, 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

\( \varphi \) is a timed property

\[
\begin{array}{c}
\text{timed events} \\
\sigma \preceq_d \sigma \\
\sigma \models \varphi!
\end{array}
\]

\[
\begin{array}{c}
\text{Enforcement Mechanism} \\
\text{timed Memory}
\end{array}
\]

\[
\sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*
\]

A formal framework for runtime enforcement of timed properties

- Any regular timed property \( \varphi \) as input.
- Enforcement mechanisms i) add additional delays between actions to satisfy \( \varphi \), 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.

\[
\Sigma \setminus \{\text{req}\} \xrightarrow{\text{req}, \ x := 0} \Sigma \setminus \{\text{req}\} \xrightarrow{\text{req}, \ x \geq 5, \ x := 0} \Sigma \setminus \{\text{req}\} \xrightarrow{\text{req}, \ x < 5} \Sigma
\]
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 $\varphi$.
- **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 $\varphi$.

- **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 $\varphi$.
- **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 $\varphi$.
- **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

On the Runtime Enforcement of Untimed Properties

Specifying Timed Properties

Runtime Enforcement of Timed Properties

Requirements on an Enforcement Mechanism

Functional Definition of an Enforcement Mechanism

Operational Description of an Enforcement Mechanism

Algorithmic Description of an Enforcement Mechanism

A note on Non-enforceable Properties

Extensions

Conclusions and Future Work
Requirements on an Enforcement Mechanism

Specified on an enforcement function for $\varphi$

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

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 $\sigma$ is the sequence read as input.
Physical constraints: The input and output are timed words. The output is produced in a “streaming fashion”\(^1\)

\[
\begin{align*}
\text{Phy1} & \quad t \leq t' \implies E_\varphi(\sigma, t) \preceq E_\varphi(\sigma, t'). \\
\text{Phy2} & \quad \sigma \preceq \sigma' \implies E_\varphi(\sigma, t) \preceq E_\varphi(\sigma', t).
\end{align*}
\]

\(^1\)Implicit universal quantification over \(\sigma\) and \(t\).
**Soundness:** The output is (eventually) correct

\[ \text{Snd } E_\varphi(\sigma, t) \neq \epsilon \implies \exists t' \geq t : E_\varphi(\sigma, t') \models \varphi. \]
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

$$\text{obs}(\sigma, t) = \max_\preceq \{ \sigma' \mid \sigma' \preceq \sigma \land \text{time}(\sigma') \leq t \}$$

$\rightarrow$ the (max) prefix of $\sigma$ that can be observed with $t$ t.u.
Requirements on an Enforcement Mechanism – transparency

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

\[ \text{Tr } E_\varphi(\sigma, t) \preceq_d \text{obs}(\sigma, t), \text{ where } \sigma' \preceq_d \sigma \text{ means:} \]
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
**Optimality:** output is produced ASAP … but not too soon
**Optimality:** output is produced ASAP ... but not too soon
Requirements on an Enforcement Mechanism – optimality

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

- $E_{\varphi}(\sigma, t)$
- Ideal sequence
- Actual delay
- Ideal delay
- Time needed to read the chunk $\leq$ actual delay
Outline - On the Runtime Enforcement of Timed Properties

On the Runtime Enforcement of Untimed Properties

Specifying Timed Properties

Runtime Enforcement of Timed Properties
  Requirements on an Enforcement Mechanism
  Functional Definition of an Enforcement Mechanism
  Operational Description of an Enforcement Mechanism
  Algorithmic Description of an Enforcement Mechanism
  A note on Non-enforceable Properties

Extensions

Conclusions and Future Work
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:

\[ \text{obs}(\sigma, t) = \max_\preceq \{ \sigma' | \sigma' \preceq \sigma \land \text{time}(\sigma') \leq t \}. \]
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:

\[ \text{obs}(\sigma, t) = \max \{ \sigma' \mid \sigma' \preceq \sigma \land \text{time}(\sigma') \leq t \}. \]
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:

\[ \text{obs}(\sigma, t) = \max_{\preceq} \{ \sigma' | \sigma' \preceq \sigma \land \text{time}(\sigma') \leq t \}. \]
(Simplified) Functional definition (2) – no suppression

\[ E_\phi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \rightarrow (\mathbb{R}_{\geq 0} \times \Sigma)^* \]

\[ E_\phi(\sigma, t) = \text{obs}\left( \Pi_1(\text{store}_\phi(\text{obs}(\sigma, t))), t \right). \]

\[ \text{store}_\phi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \rightarrow (\mathbb{R}_{\geq 0} \times \Sigma)^* \times (\mathbb{R}_{\geq 0} \times \Sigma)^* \]

\( \text{store}_\phi(\sigma) \) is a pair:

1. delayed correct prefix of \( \sigma \),
2. suffix of \( \sigma \) for which delays still have to be computed.
(Simplified) Functional definition (2) – no suppression

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

\[ E_\varphi(\sigma, t) = \text{obs} \left( \Pi_1(\text{store}_\varphi(\text{obs}(\sigma, t))), t \right). \]

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

\[ \text{store}_\varphi(\sigma) \text{ is a pair:} \]

1. delayed correct prefix of \( \sigma \),

2. suffix of \( \sigma \) for which delays still have to be computed.
(Simplified) Functional definition (3) – no suppression

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

\[ E_\varphi(\sigma, t) = \text{obs}\left( \prod_1(\text{store}_\varphi(\text{obs}(\sigma, t))), t \right). \]

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

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

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

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

with

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

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

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**

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

\[ E_\varphi(\sigma, t) = \text{obs}\left(\Pi_1(\text{store}_\varphi(\text{obs}(\sigma, t))), t\right). \]

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

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

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

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

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

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

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

\[ E_\varphi(\sigma, t) = \text{obs}\left( \Pi_1(\text{store}_\varphi(\text{obs}(\sigma, t))), t \right). \]

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

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

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

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

with

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

\[ \kappa_\varphi(T, \sigma_s, \sigma_c) = \{ w \in (\mathbb{R}_{\geq 0} \times \Sigma)^* \mid w \preceq_d \sigma_c \land |w| = |\sigma_c| \]

\[ \land \sigma_s \cdot w \models \varphi \land \text{delay}(w(1)) \geq T - \text{time}(\sigma_s) \} \]

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

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

\[ E_\varphi(\sigma, t) = \text{obs}\left( \prod_1(\text{store}_\varphi(\text{obs}(\sigma, t))), t \right). \]

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

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

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

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

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

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

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

\[ E_\varphi(\sigma, t) = \text{obs}\left(\Pi_1(\text{store}_\varphi(\text{obs}(\sigma, t))), t\right). \]

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

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

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

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

with

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

\[ \kappa_\varphi(T, \sigma_s, \sigma_c) = \{ w \in (\mathbb{R}_{\geq 0} \times \Sigma)^* \mid w \prec_d \sigma_c \land |w| = |\sigma_c| \]

\[ \land \sigma_s \cdot w \models \varphi \land \text{delay}(w(1)) \geq T - \text{time}(\sigma_s) \}\]

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 \(\text{time}(\sigma_c \cdot (\delta, a))\).
### Functional definition: Example

\[ \Sigma = \{req, gr\}. \]

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

\[
\begin{array}{|c|}
\hline
\text{Time Interval} & \text{Transition Specification} & \text{Observation Function} & \text{Store Function} & \text{Environment} \\
\hline
[0, 3] & & \text{obs}(\sigma, t) = \epsilon & \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon) & E(\varphi, \sigma, t) = \text{obs}(\epsilon, t) \\
\hline
[3, 13] & & \text{obs}(\sigma, t) = (3, req) & \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req)) & E(\varphi, \sigma, t) = \text{obs}(\epsilon, t) \\
\hline
[13, 16] & & \text{obs}(\sigma, t) = (3, req) \cdot (10, gr) & \text{store}_\varphi(\text{obs}(\sigma, t)) = ((13, req) \cdot (15, gr), \epsilon) & E(\varphi, \sigma, t) = \text{obs}((13, req) \cdot (15, gr), t) \\
\hline
[16, 21] & & \text{obs}(\sigma, t) = (3, req) \cdot (10, gr) \cdot (3, req) & \text{store}_\varphi(\text{obs}(\sigma, t)) = ((13, req) \cdot (15, gr), (3, req)) & E(\varphi, \sigma, t) = \text{obs}((13, req) \cdot (15, gr), t) \\
\hline
[21, \infty] & & \text{obs}(\sigma, t) = (3, req) \cdot (10, gr) \cdot (3, req) \cdot (5, req) & \text{store}_\varphi(\text{obs}(\sigma, t)) = ((13, req) \cdot (15, gr), (3, req) \cdot (5, req)) & E(\varphi, \sigma, t) = \text{obs}((13, req) \cdot (15, gr), t) \\
\hline
\end{array}
\]
**Functional definition: Example**

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Observation ((\text{obs}(\sigma, t)))</th>
<th>Store ((\text{store}_\phi(\text{obs}(\sigma, t))))</th>
<th>Environment ((E_\phi(\sigma, t)))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t \in [0, 3])</td>
<td>(\epsilon)</td>
<td>((\epsilon, \epsilon))</td>
<td>(\text{obs}(\epsilon, t))</td>
</tr>
<tr>
<td>(t \in [3, 13])</td>
<td>((3, \text{req}))</td>
<td>((\epsilon, (3, \text{req})))</td>
<td>(\text{obs}(3, \text{req}, t))</td>
</tr>
<tr>
<td>(t \in [13, 16])</td>
<td>((3, \text{req}) \cdot (10, \text{gr}))</td>
<td>(((13, \text{req}) \cdot (15, \text{gr}), \epsilon))</td>
<td>(\text{obs}((13, \text{req}) \cdot (15, \text{gr}), t))</td>
</tr>
<tr>
<td>(t \in [16, 21])</td>
<td>((3, \text{req}) \cdot (10, \text{gr}) \cdot (3, \text{req}))</td>
<td>(((13, \text{req}) \cdot (15, \text{gr}), (3, \text{req})))</td>
<td>(\text{obs}((13, \text{req}) \cdot (15, \text{gr}), t))</td>
</tr>
<tr>
<td>(t \in [21, \infty])</td>
<td>((3, \text{req}) \cdot (10, \text{gr}) \cdot (3, \text{req}) \cdot (5, \text{req}))</td>
<td>(((13, \text{req}) \cdot (15, \text{gr}), (3, \text{req}) \cdot (5, \text{req})))</td>
<td>(\text{obs}((13, \text{req}) \cdot (15, \text{gr}), t))</td>
</tr>
</tbody>
</table>
Functional definition: Example

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

\[
\begin{align*}
\text{\(t \in [0, 3]\)} & \quad \begin{aligned}
\text{obs}(\sigma, t) &= \epsilon \\
\text{store}_\varphi(\text{obs}(\sigma, t)) &= (\epsilon, \epsilon) \\
E_\varphi(\sigma, t) &= \text{obs}(\epsilon, t)
\end{aligned} \\
\text{\(t \in [3, 13]\)} & \quad \begin{aligned}
\text{obs}(\sigma, t) &= (3, req) \\
\text{store}_\varphi(\text{obs}(\sigma, t)) &= (\epsilon, (3, req)) \\
E_\varphi(\sigma, t) &= \text{obs}(\epsilon, t)
\end{aligned} \\
\text{\(t \in [13, 16]\)} & \quad \begin{aligned}
\text{obs}(\sigma, t) &= (3, req) \cdot (10, gr) \\
\text{store}_\varphi(\text{obs}(\sigma, t)) &= ((13, req) \cdot (15, gr), \epsilon) \\
E_\varphi(\sigma, t) &= \text{obs}((13, req) \cdot (15, gr), t)
\end{aligned} \\
\text{\(t \in [16, 21]\)} & \quad \begin{aligned}
\text{obs}(\sigma, t) &= (3, req) \cdot (10, gr) \cdot (3, req) \\
\text{store}_\varphi(\text{obs}(\sigma, t)) &= ((13, req) \cdot (15, gr), (3, req)) \\
E_\varphi(\sigma, t) &= \text{obs}((13, req) \cdot (15, gr), t)
\end{aligned} \\
\text{\(t \in [21, \infty]\)} & \quad \begin{aligned}
\text{obs}(\sigma, t) &= (3, req) \cdot (10, gr) \cdot (3, req) \cdot (5, req) \\
\text{store}_\varphi(\text{obs}(\sigma, t)) &= ((13, req) \cdot (15, gr), (3, req) \cdot (5, req)) \\
E_\varphi(\sigma, t) &= \text{obs}((13, req) \cdot (15, gr), t)
\end{aligned}
\end{align*}
\]
Functional definition: Example

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

<table>
<thead>
<tr>
<th>( t \in [0, 3] )</th>
<th>( \text{obs}(\sigma, t) = \epsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon) )</td>
</tr>
<tr>
<td></td>
<td>( E_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( t \in [3, 13] )</th>
<th>( \text{obs}(\sigma, t) = (3, \text{req}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, \text{req})) )</td>
</tr>
<tr>
<td></td>
<td>( E_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( t \in [13, 16] )</th>
<th>( \text{obs}(\sigma, t) = (3, \text{req}) \cdot (10, \text{gr}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = ((13, \text{req}) \cdot (15, \text{gr}), \epsilon) )</td>
</tr>
<tr>
<td></td>
<td>( E_\varphi(\sigma, t) = \text{obs}((13, \text{req}) \cdot (15, \text{gr}), t) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( t \in [16, 21] )</th>
<th>( \text{obs}(\sigma, t) = (3, \text{req}) \cdot (10, \text{gr}) \cdot (3, \text{req}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = ((13, \text{req}) \cdot (15, \text{gr}), (3, \text{req})) )</td>
</tr>
<tr>
<td></td>
<td>( E_\varphi(\sigma, t) = \text{obs}((13, \text{req}) \cdot (15, \text{gr}), t) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( t \in [21, \infty) )</th>
<th>( \text{obs}(\sigma, t) = (3, \text{req}) \cdot (10, \text{gr}) \cdot (3, \text{req}) \cdot (5, \text{req}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = ((13, \text{req}) \cdot (15, \text{gr}), (3, \text{req}) \cdot (5, \text{req})) )</td>
</tr>
<tr>
<td></td>
<td>( E_\varphi(\sigma, t) = \text{obs}((13, \text{req}) \cdot (15, \text{gr}), t) )</td>
</tr>
</tbody>
</table>
Functional definition: Example

\[ \Sigma = \{ req, gr \} \]

\[ \sigma = (3, req) \cdot (10, gr) \cdot (3, req) \cdot (5, req) \]

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

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

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

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

\[ t \in [21, \infty] \]
\[ \text{obs}(\sigma, t) = (3, req) \cdot (10, gr) \cdot (3, req) \cdot (5, req) \]
\[ \text{store}_\varphi(\text{obs}(\sigma, t)) = ((13, req) \cdot (15, gr), (3, req) \cdot (5, req)) \]
\[ E_\varphi(\sigma, t) = \text{obs}((13, req) \cdot (15, gr), t) \]
Functional definition: Example

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

On the Runtime Enforcement of Untimed Properties

Specifying Timed Properties

Runtime Enforcement of Timed Properties
  Requirements on an Enforcement Mechanism
  Functional Definition of an Enforcement Mechanism
  Operational Description of an Enforcement Mechanism
  Algorithmic Description of an Enforcement Mechanism
  A note on Non-enforceable Properties

Extensions

Conclusions and Future Work
Enforcement monitor

\[ E(\sigma, t) \models \varphi \]

A rule-based transition system:

- configurations keep track of
  - the prefix of \( \sigma \) that has been corrected but yet to be output ("good memory"),
  - the suffix of \( \sigma \) 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 \( \sigma \) that has been corrected but yet to be output ("good memory"),
  - the suffix of \( \sigma \) 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 $\sigma$ that has been corrected but yet to be output ("good memory"),
  - the suffix of $\sigma$ 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 $\sigma$ that has been corrected but yet to be output ("good memory"),
  - the suffix of $\sigma$ 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-\(\bar{\varphi}\)**
   - when a new event is received and it *cannot make* \(\varphi\) *satisfied* by delaying.
   - updates “bad” memory and store clock

2. **store-\(\varphi\)**
   - when a new event is received and it *can make* \(\varphi\) *satisfied* by delaying
   - updates “good” memory and store clock

3. **suppress**
   - when an event is received and prevents \(\varphi\)’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-$\overline{\phi}$
   - when a new event is received and it \textit{cannot make $\phi$ satisfied} by delaying.
   - updates “bad” memory and store clock

2. store-$\phi$
   - when a new event is received and it \textit{can make $\phi$ satisfied} by delaying.
   - updates “good” memory and store clock

3. suppress
   - when an event is received and prevents $\phi$'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-ϕ**
   - 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 $\varphi$ satisfied by delaying.
   - updates “bad” memory and store clock

2. store-$\varphi$
   - when a new event is received and it can make $\varphi$ satisfied by delaying
   - updates “good” memory and store clock

3. suppress
   - when an event is received and prevents $\varphi$’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-$$\varphi$$**
   - when a new event is received and it cannot make $$\varphi$$ satisfied by delaying.
   - updates “bad” memory and store clock

2. **store-$$\varphi$$**
   - when a new event is received and it can make $$\varphi$$ satisfied by delaying
   - updates “good” memory and store clock

3. **suppress**
   - when an event is received and prevents $$\varphi$$'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 $\varphi$, 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

\[
\begin{align*}
\sum \setminus \{ \text{req, gr} \} & \quad \text{req, } x := 0 \\
\Sigma & \quad \Sigma \setminus \{ \text{req, gr} \} \\
\Sigma & \quad \Sigma \setminus \{ \text{gr} \}; \\
\text{gr, } 15 \leq x \leq 20; & \quad x := 0 \\
\text{gr} & \quad \Sigma \setminus \{ \text{gr} \}; \\
\text{gr, } x < 15 \lor x > 20 & \quad (3, r) \cdot (10, g) \cdot (3, r) \cdot (5, r)
\end{align*}
\]

- Good Memory: \( \epsilon \)
- Bad Memory: \( \epsilon \)
- State: \((l_0, 0)\)
Enforcement Monitor: example

\[ t = 3 \]

- Executed operation: \texttt{idle}(3)

\[
\begin{align*}
\Sigma \setminus \{\text{req}, \text{gr}\} & \quad \text{req,} \\
& \quad x := 0 \\
\Sigma \setminus \{\text{req}, \text{gr}\} & \quad \text{gr,} \\
& \quad 15 \leq x \leq 20; \quad x := 0 \\
\Sigma \setminus \{\text{gr}\} & \quad \text{gr,} \\
& \quad x < 15 \lor x > 20
\end{align*}
\]

\[
(3, r) \cdot (10, g) \cdot (3, r) \cdot (5, r)
\]

- Good Memory: \( \epsilon \)
- Bad Memory: \( \epsilon \)
- State: \((l_0, 3)\)
Enforcement Monitor: example

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

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

\[\begin{align*}
\Sigma \setminus \{req, gr\} & \quad \{req, \} \leftarrow l_0 \\
\{gr, x := 0 \mid 15 \leq x \leq 20\} & \quad \{gr\} \leftarrow l_1 \\
\{gr, x < 15 \lor x > 20\} & \quad \{gr\} \leftarrow l_2
\end{align*}\]
Enforcement Monitor: example

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

- **Good Memory:** $\epsilon$
- **Bad Memory:** $(3, r)$
- **State:** $(l_0, 0)$
Enforcement Monitor: example

\[ t = 13 \]

- Executed operation: \( \text{store-} \varphi \)

\[
\begin{align*}
\Sigma \setminus \{\text{req, gr}\} & \quad \text{req,} \\
\quad \quad x := 0 & \quad \Sigma \setminus \{\text{req, gr}\} \\
\text{gr, } 15 \leq x \leq 20; \\
\quad \quad x := 0 & \quad \Sigma \setminus \{\text{gr}\}; \\
\text{gr, } x < 15 \lor x > 20 & \quad \Sigma \setminus \{\text{gr}\}; \\
\quad \quad \epsilon & \quad (3, r) \cdot (5, r)
\end{align*}
\]

- Good Memory: 
  \( (13, r) \cdot (15, g) \)
- Bad Memory: \( \epsilon \)
- State: \( (l_0, 0) \)
Enforcement Monitor: example

$t = 13$ - Executed operation: dump

- Good Memory: (15, g)
- Bad Memory: ε
- State: (l₀, 15)
Enforcement Monitor: example

\[ t = 16 \]

- Executed operation: \( \text{idle}(3) \)

\[
\begin{align*}
\Sigma \setminus \{\text{req, gr}\} & \\
\rightarrow l_0 & \xrightarrow{\text{req}, x := 0} l_1 & \xrightarrow{\Sigma \setminus \{\text{req, gr}\}} l_1 \\
gr, 15 \leq x \leq 20; x := 0 & \\
\rightarrow l_2 & \xrightarrow{\Sigma \setminus \{\text{gr}\}} l_2 & \xrightarrow{\Sigma \setminus \{\text{gr}\}} l_2 \\
gr, x < 15 \lor x > 20 & \\
\rightarrow (13, r) & \leftarrow (3, r) \cdot (5, r) & \\
\end{align*}
\]

- Good Memory: \( (15, g) \)
- Bad Memory: \( \epsilon \)
- State: \( (l_0, 15) \)
Enforcement Monitor: example

- Executed operation: \( \text{store-} \overline{\varphi} \)

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

\[ t = 16 \]
Enforcement Monitor: example

\[ t = 21 \]

- Executed operation: \( \text{idle}(5) \)

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

(13, r) \leftarrow \text{(5, r)}

- Good Memory: (15, g)
- Bad Memory: (3, r)
- State: (l_0, 15)
Enforcement Monitor: example

\[ t = 21 \]

- Executed operation: suppress

\[ (13, r) \]

\[ \Sigma \setminus \{req, gr\} \]

\[ l_0 \]

\[ req, x := 0 \]

\[ l_1 \]

\[ \Sigma \setminus \{req, gr\} \]

\[ gr, 15 \leq x \leq 20; x := 0 \]

\[ l_2 \]

\[ \Sigma \setminus \{gr\}; gr, x < 15 \lor x > 20 \]

\[ \Sigma \]

- Good Memory: \((15, g)\)
- Bad Memory: \((3, r)\)
- State: \((l_0, 15)\)
Enforcement Monitor: example

\[ t = 28 \] - Executed operation: \text{idle}(7)

- Good Memory: (15, g)
- Bad Memory: (3, r)
- State: (l_0, 15)
Enforcement Monitor: example

\( t = 28 \)

- Executed operation: \( \text{dump} \)

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

\( \Sigma \setminus \{\text{req, gr}\} \)

\( \text{req, } x := 0 \)

\( \Sigma \setminus \{\text{req, gr}\} \)

\( \text{gr, } 15 \leq x \leq 20; \quad x := 0 \)

\( \Sigma \setminus \{\text{gr}\}; \quad \text{gr, } x < 15 \lor x > 20 \)

\( \Sigma \)

- Good Memory: \( \epsilon \)
- Bad Memory: \( (3, r) \)
- State: \( (l_0, 15) \)
Outline - On the Runtime Enforcement of Timed Properties

On the Runtime Enforcement of Untimed Properties

Specifying Timed Properties

Runtime Enforcement of Timed Properties
  Requirements on an Enforcement Mechanism
  Functional Definition of an Enforcement Mechanism
  Operational Description of an Enforcement Mechanism
  Algorithmic Description of an Enforcement Mechanism
  A note on Non-enforceable Properties

Extensions

Conclusions and Future Work
Implementation – simplified algorithms

*Used primitives:*
- `await(condition)`, `wait(time)`
- `post(loc, valuation, tw)`
- `update(loc, valuation, tw)`

*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, \nu, \sigma_c)`

if `isPath = tt` then

\[
\sigma_s \leftarrow \sigma_s \cdot \sigma'_c
\]
\[
(l, \nu) \leftarrow `post(l, \nu, \sigma'_c)`
\]
\[
\sigma_c \leftarrow \epsilon
\]

end if

end while

Algorithm: DumpProcess

\[
d \leftarrow 0
\]

while `tt` do

\[
\text{await } (\sigma_s \neq \epsilon)
\]
(\delta, a) \leftarrow `dequeue (\sigma_s)`
\[
\text{wait } (\delta - d)
\]
dump (a)
\[
d \leftarrow 0
\]
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
    wait (\(\sigma_s \neq \epsilon\))
    (\(\delta, a\)) \leftarrow \text{dequeue} (\(\sigma_s\))
    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 \text{await} (\text{event})
    \(\sigma_c \leftarrow \sigma_c \cdot (\delta, a)\)
    (\(\sigma'_c, \text{isPath}\)) \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

On the Runtime Enforcement of Untimed Properties

Specifying Timed Properties

Runtime Enforcement of Timed Properties
  Requirements on an Enforcement Mechanism
  Functional Definition of an Enforcement Mechanism
  Operational Description of an Enforcement Mechanism
  Algorithmic Description of an Enforcement Mechanism
  A note on Non-enforceable Properties

Extensions

Conclusions and Future Work
Non-enforceable response properties

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

<table>
<thead>
<tr>
<th>Interval</th>
<th>( t ) \in [0, 3]</th>
<th>( t ) \in [3, 7]</th>
<th>( t ) \in [7, 9]</th>
<th>( t ) \in [9, 15]</th>
<th>( t ) \in [15, \infty]</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{obs}(\sigma, t) = \epsilon )</td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon) )</td>
<td>( \text{obs}(\sigma, t) = (3, req) )</td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req)) )</td>
<td>( \text{obs}(\sigma, t) = (3, req) \cdot (4, gr) )</td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req) \cdot (4, gr)) )</td>
</tr>
<tr>
<td>( \text{E}_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
<td>( \text{E}_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
<td>( \text{E}_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
<td>( \text{E}_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
<td>( \text{E}_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
<td></td>
</tr>
</tbody>
</table>
Non-enforceable response properties

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

- **\( t \in [0, 3] \)**
  - \( \text{obs}(\sigma, t) = \epsilon \)
  - \( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon) \)
  - \( E_\varphi(\sigma, t) = \text{obs}(\epsilon, t) \)

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

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

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

- **\( t \in [15, \infty] \)**
  - \( \text{obs}(\sigma, t) = (3, \text{req}) \cdot (4, \text{gr}) \cdot (2, \text{req}) \cdot (6, \text{gr}) \)
  - \( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, \text{req}) \cdot (4, \text{gr}) \cdot (2, \text{req}) \cdot (6, \text{gr})) \)
  - \( E_\varphi(\sigma, t) = \text{obs}(\epsilon, t) \)
Non-enforceable response properties

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

<table>
<thead>
<tr>
<th>Interval</th>
<th>Observation ((\text{obs}(\sigma, t)))</th>
<th>Store ((\text{store}_\varphi(\text{obs}(\sigma, t))))</th>
<th>Enforcement ((\text{E}_\varphi(\sigma, t)))</th>
</tr>
</thead>
<tbody>
<tr>
<td>([0, 3])</td>
<td>(\epsilon)</td>
<td>((\epsilon, \epsilon))</td>
<td>(\text{obs}(\epsilon, t))</td>
</tr>
<tr>
<td>([3, 7])</td>
<td>((3, req))</td>
<td>((\epsilon, (3, req)))</td>
<td>(\text{obs}(\epsilon, t))</td>
</tr>
<tr>
<td>([7, 9])</td>
<td>((3, req) \cdot (4, gr))</td>
<td>((\epsilon, (3, req) \cdot (4, gr)))</td>
<td>(\text{obs}(\epsilon, t))</td>
</tr>
<tr>
<td>([9, 15])</td>
<td>((3, req) \cdot (4, gr) \cdot (2, req))</td>
<td>((\epsilon, (3, req) \cdot (4, gr) \cdot (2, req)))</td>
<td>(\text{obs}(\epsilon, t))</td>
</tr>
<tr>
<td>([15, \infty])</td>
<td>((3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr))</td>
<td>((\epsilon, (3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr)))</td>
<td>(\text{obs}(\epsilon, t))</td>
</tr>
</tbody>
</table>

\[ \Sigma \setminus \{\text{req, gr}\} \]
\[ x \leq 5 \]
**Non-enforceable response properties**

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Observation at Time</th>
<th>Store Operation</th>
<th>Evaluation of Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t \in [0, 3] )</td>
<td>( \text{obs}(\sigma, t) = \epsilon )</td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon) )</td>
<td>( \text{E}_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
</tr>
<tr>
<td>( t \in [3, 7] )</td>
<td>( \text{obs}(\sigma, t) = (3, req) )</td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req)) )</td>
<td>( \text{E}_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
</tr>
<tr>
<td>( t \in [7, 9] )</td>
<td>( \text{obs}(\sigma, t) = (3, req) \cdot (4, gr) )</td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req) \cdot (4, gr)) )</td>
<td>( \text{E}_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
</tr>
<tr>
<td>( t \in [9, 15] )</td>
<td>( \text{obs}(\sigma, t) = (3, req) \cdot (4, gr) \cdot (2, req) )</td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req) \cdot (4, gr) \cdot (2, req)) )</td>
<td>( \text{E}_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
</tr>
<tr>
<td>( t \in [15, \infty) )</td>
<td>( \text{obs}(\sigma, t) = (3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr) )</td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr)) )</td>
<td>( \text{E}_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
</tr>
</tbody>
</table>

Diagram:

- \( l_0 \rightarrow l_1 \): \( gr, x \leq 10; \ x := 0 \)
- \( l_1 \rightarrow l_2 \): \( gr, x > 10 \)
- \( l_2 \): \( req; \)
- \( l_2 \rightarrow l_0 \): \( req, x < 5 \)
- \( l_0 \rightarrow l_2 \): \( req, x \geq 5 \)
- \( l_0 \rightarrow l_0 \): \( \Sigma \setminus \{req, gr\} \)
- \( l_1 \rightarrow l_1 \): \( \Sigma \setminus \{req, gr\} \)

Equation:

\[ t \in [0, 3] \]: \( \text{obs}(\sigma, t) = \epsilon \), \( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon) \), \( \text{E}_\varphi(\sigma, t) = \text{obs}(\epsilon, t) \)

\[ t \in [3, 7] \]: \( \text{obs}(\sigma, t) = (3, req) \), \( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req)) \), \( \text{E}_\varphi(\sigma, t) = \text{obs}(\epsilon, t) \)

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

\[ t \in [9, 15] \]: \( \text{obs}(\sigma, t) = (3, req) \cdot (4, gr) \cdot (2, req) \), \( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req) \cdot (4, gr) \cdot (2, req)) \), \( \text{E}_\varphi(\sigma, t) = \text{obs}(\epsilon, t) \)

\[ t \in [15, \infty) \]: \( \text{obs}(\sigma, t) = (3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr) \), \( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr)) \), \( \text{E}_\varphi(\sigma, t) = \text{obs}(\epsilon, t) \)
Non-enforceable response properties

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

<table>
<thead>
<tr>
<th>Interval</th>
<th>Observation</th>
<th>Store</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t \in [0, 3[ )</td>
<td>( \text{obs}(\sigma, t) = \epsilon )</td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, \epsilon) )</td>
<td>( E_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
</tr>
<tr>
<td>( t \in [3, 7[ )</td>
<td>( \text{obs}(\sigma, t) = (3, req) )</td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req)) )</td>
<td>( E_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
</tr>
<tr>
<td>( t \in [7, 9[ )</td>
<td>( \text{obs}(\sigma, t) = (3, req) \cdot (4, gr) )</td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req) \cdot (4, gr)) )</td>
<td>( E_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
</tr>
<tr>
<td>( t \in [9, 15[ )</td>
<td>( \text{obs}(\sigma, t) = (3, req) \cdot (4, gr) \cdot (2, req) )</td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req) \cdot (4, gr) \cdot (2, req)) )</td>
<td>( E_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
</tr>
<tr>
<td>( t \in [15, \infty[ )</td>
<td>( \text{obs}(\sigma, t) = (3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr) )</td>
<td>( \text{store}_\varphi(\text{obs}(\sigma, t)) = (\epsilon, (3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr)) )</td>
<td>( E_\varphi(\sigma, t) = \text{obs}(\epsilon, t) )</td>
</tr>
</tbody>
</table>
Non-enforceable response properties

\[ \Sigma = \{gr, req\}. \]
\[ \sigma = (3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr). \]
Extensions
Outline - On the Runtime Enforcement of Timed Properties

On the Runtime Enforcement of Untimed Properties

Specifying Timed Properties

Runtime Enforcement of Timed Properties

Extensions

Considering Uncontrollable Events [ICTAC’15]

Considering Events with Data [WODES’14]

Conclusions and Future Work
Outline - On the Runtime Enforcement of Timed Properties

On the Runtime Enforcement of Untimed Properties

Specifying Timed Properties

Runtime Enforcement of Timed Properties

Extensions

Considering Uncontrollable Events [ICTAC’15]

Considering Events with Data [WODES’14]

Conclusions and Future Work
Uncontrollable events

Two sets of events: controllable ($\Sigma_c$) and uncontrollable ($\Sigma_u$).

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 ($\Sigma_c$) and uncontrollable ($\Sigma_u$).

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 ($\Sigma_c$) and uncontrollable ($\Sigma_u$).

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
Example of Non-Enforceable Property

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

- 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

\[ \Sigma_u = \{ \text{Auth, LockOFF, LockON} \}, \Sigma_c = \{ \text{Write} \} \]
Property with time: example of execution

\[ \Sigma_u = \{ \text{Auth, LockOFF, LockON} \}, \Sigma_c = \{ \text{Write} \} \]
Property with time: example of execution

\[ \Sigma_u = \{ \text{Auth}, \text{LockOFF}, \text{LockON} \}, \Sigma_c = \{ \text{Write} \} \]
Property with time: example of execution

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

Diagram:
- Initial state: \( l_0 \)
- Transitions:
  - Auth: \( l_0 \rightarrow l_1 \)
  - LockOFF: \( l_1 \rightarrow l_3 \)
  - Write: \( l_3 \rightarrow l_1 \)
  - LockON: \( l_2 \rightarrow l_3 \)
  - LockOFF: \( l_3 \rightarrow l_2 \)

States:
- \( l_0 \): Auth
- \( l_1 \): Write
- \( l_2 \): LockON
- \( l_3 \): LockOFF

Transitions:
- \( \text{Auth, LockOFF} x := 0 \)
- \( \text{Write} x \geq 2 \)
- \( \text{Write} x < 2 \)
- \( \text{Write} \)

Time:
\[ t = 4 \]

Events:
- \((1, \text{auth}).(2, \text{on}).(5, \text{off}) \)
- \((8, \text{off}) \)
- \((1, \text{auth}).(2, \text{on}).(4, \text{w}) (5, \text{off}) \)
- \((6, \text{on}).(7, \text{w}) \)

Graph:
- Graph with nodes and directed edges illustrating the transitions and states.
- Labels on edges and nodes indicating actions and values.

Conclusion:
- Example execution of a property with time, showing transitions and state changes.

Future Work:
- Further exploration of timed properties and their specifications.

Extensions:
- Advanced techniques for specifying timed properties in system models.

Conclusions and Future Work:
Property with time: example of execution

\[ \Sigma_u = \{ \text{Auth, LockOFF, LockON} \}, \; \Sigma_c = \{ \text{Write} \} \]
Property with time: example of execution

\[ \Sigma_u = \{ \text{Auth, LockOFF, LockON} \}, \Sigma_c = \{ \text{Write} \} \]
Property with time: example of execution

\[ \Sigma_u = \{ \text{Auth, LockOFF, LockON} \}, \Sigma_c = \{ \text{Write} \} \]
Property with time: example of execution

\[ \Sigma_u = \{ \text{Auth}, \text{LockOFF}, \text{LockON} \}, \Sigma_c = \{ \text{Write} \} \]
Property with time: example of execution

\[ \Sigma_u = \{ \text{Auth, LockOFF, LockON} \}, \Sigma_c = \{ \text{Write} \} \]
Outline - On the Runtime Enforcement of Timed Properties

On the Runtime Enforcement of Untimed Properties

Specifying Timed Properties

Runtime Enforcement of Timed Properties

Extensions

Considering Uncontrollable Events [ICTAC’15]

Considering Events with Data [WODES’14]

Conclusions and Future Work
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

On the Runtime Enforcement of Untimed Properties

Specifying Timed Properties

Runtime Enforcement of Timed Properties

Extensions

Considering Uncontrollable Events [ICTAC’15]

Considering Events with Data [WODES’14]

Parameterized Timed Automata with Variables (PTAVs)

Runtime Enforcement of PTAVs

Application Domains

Conclusions and Future Work
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
\[ A(p) = \langle p, V, C, \Theta, L, l_0, L_G, X, \Sigma_p, \Delta \rangle. \]

- \( p \) - 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 \( \pi \) of \( p \) is denoted as \( A(\pi) \).
Parameterized Timed Automata with Variables (PTAVs)

- Extension of timed automata.
- Describes a set of \textit{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.)

\textbf{Syntax of PTAV}

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

- $p$ - 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 $\pi$ 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 \( \sigma \) if \( \sigma \in L(A(\pi)) \).

Projections of \( \sigma \) according to the runtime values of \( \pi \)

- \( \sigma = (0.5, a(1, \eta_1)) \cdot (0.3, a(2, \eta_2)) \cdot (0.2, a(1, \eta_3)) \cdot (0.4, a(2, \eta_4)) \)
- \( \sigma \downarrow_1 = (0.5, a(1, \eta_1)) \cdot (0.5, a(1, \eta_3)) \)
- \( \sigma \downarrow_2 = (0.8, a(2, \eta_2)) \cdot (0.6, a(2, \eta_4)) \)
Parameterized Timed Automata with Variables (PTAV)

Events, timed words

- **Event**: $e_i = (\delta_i, a_i(\pi_i, \eta_i))$, $a_i \in \Sigma$, $\pi_i \in D_p$, $\eta_i \in D_V$.
- **Timed word**: $\sigma = (\delta_1, a_1(\pi_1, \eta_1)) \cdot \ldots \cdot (\delta_n, a_n(\pi_n, \eta_n))$.
- Instance of PTAV $A(\pi)$ accepts $\sigma$ if $\sigma \in L(A(\pi))$.

Projections of $\sigma$ according to the runtime values of $\pi$

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

Events, timed words

- **Event**: $e_i = (\delta_i, a_i(\pi_i, \eta_i)), a_i \in \Sigma, \pi_i \in D_p, \eta_i \in D_V$.
- **Timed word**: $\sigma = (\delta_1, a_1(\pi_1, \eta_1)) \cdot \cdots \cdot (\delta_n, a_n(\pi_n, \eta_n))$.
- Instance of PTAV $A(\pi)$ accepts $\sigma$ if $\sigma \in L(A(\pi))$.

Projections of $\sigma$ according to the runtime values of $\pi$

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

Projections of $\sigma$ according to the runtime values of $\pi$

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

Projections of $\sigma$ according to the runtime values of $\pi$

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

**Events, timed words**

- **Event**: $e_i = (\delta_i, a_i(\pi_i, \eta_i))$, $a_i \in \Sigma$, $\pi_i \in D_p$, $\eta_i \in D_V$.
- **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 $\sigma$ if $\sigma \in L(A(\pi))$.**

**Projections of $\sigma$ according to the runtime values of $\pi$**

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

“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.”

\[ \Sigma_{s\text{Id}} \setminus \{R1.\text{alloc}(s\text{Id})\} \]

\[ R1.\text{alloc}(s\text{Id}), \]  
\[ x > \text{reset}, \]  
\[ \text{counter} := 1, x := 0, y := 0 \]

\[ \Sigma_{s\text{Id}} \setminus \{R1.\text{alloc}(s\text{Id})\} \]

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

\[ \Sigma_{s\text{Id}} \setminus \{R1.\text{alloc}(s\text{Id})\} \]

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

2Squares 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 \(\text{maxreq}\), 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

On the Runtime Enforcement of Untimed Properties

Specifying Timed Properties

Runtime Enforcement of Timed Properties

Extensions

Considering Uncontrollable Events [ICTAC’15]

Considering Events with Data [WODES’14]

Parameterized Timed Automata with Variables (PTAVs)

Runtime Enforcement of PTAVs

Application Domains

Conclusions and Future Work
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 $\varphi$ 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**.
- **$o = \text{merge}(o_1, o_2)$ 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 $\varphi$ 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.
- $o = \text{merge}(o_1, o_2)$ 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 \( \varphi \) 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.**
- **\( o = \text{merge}(o_1, o_2) \) 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 $\varphi$ 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.
- $o = \text{merge}(o_1, o_2)$ 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 \( \varphi \) 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.
- \( o = \text{merge}(o_1, o_2) \) is only sound (order of events may not be preserved globally).
Requirements: Enforcement of parametric timed properties

\[ E_{\varphi_\rho}(\sigma, t) : (\mathbb{R}_{\geq 0} \times \Lambda)^* \times \mathbb{R}_{\geq 0} \rightarrow (\mathbb{R}_{\geq 0} \times \Lambda)^* \]

Parametric Soundness, Transparency and Optimality

- **Soundness**: \( \forall \pi \in \text{Dom}(p), \forall \sigma \in (\mathbb{R}_{\geq 0} \times \Lambda)^* : \text{sound}(E_{\varphi_\rho(\pi)}, \sigma_{\downarrow \pi}) \)
- **Transparency**: \( \forall \pi \in \text{Dom}(p), \forall \sigma \in (\mathbb{R}_{\geq 0} \times \Lambda)^* : \text{transparent}(E_{\varphi_\rho(\pi)}, \sigma_{\downarrow \pi}) \)
- **Optimality**: \( \forall \pi \in \text{Dom}(p), \forall \sigma \in (\mathbb{R}_{\geq 0} \times \Lambda)^* : \text{optimal}(E_{\varphi_\rho(\pi)}, \sigma_{\downarrow \pi}) \)

**Proposition**

Given a safety PTAV \( A(\rho) \) specifying property \( \varphi_\rho(\rho) \), for all \( \pi \in D_\rho \), the enforcement function \( E_{\varphi_A(\pi)} \), is sound, transparent, and optimal w.r.t. \( A(\pi) \).
Parametric case: Example

\[
\Sigma_{sId}^1 \setminus \{R1.aloc(sId)}\]  
\[
R1.aloc(sId), \quad x < \text{reset} \land y \geq \text{delay},\quad \text{counter} \leftarrow++, \ y := 0
\]

- \(\text{delay} = 5\)
- \(\text{reset} = 100\)

- Let \(\sigma = (2, R1.aloc(1)) \cdot (1, R1.aloc(2)) \cdot (1, R1.aloc(1)).\)
- \(\text{Dom}(\sigma) = \{1, 2\}\), we have two monitor instances.

\[\Pi = 1\]

- \(\sigma \downarrow_1 = (2, R1.aloc(1)) \cdot (2, R1.aloc(1)).\)
- \(E_{\varphi,A(1)} = (2, R1.aloc(1)) \cdot (5, R1.aloc(1)).\)

\[E_{\varphi,A(1)}\] and \(E_{\varphi,A(2)}\) are sound, transparent, and optimal.

Remark: properties of the global output trace

- \(E_{\varphi,A(p)}(\sigma, t) = \text{merge}(E_{\varphi,A(1)}(\sigma \downarrow_1, t), E_{\varphi,A(2)}(\sigma \downarrow_2, t))\)
- Only soundness is preserved.
Parametric case: Example

\[ \Sigma_{sId}^{P_1} \setminus \{ R1.allocate(sId) \} \]

- \( \Sigma_{sId}^{P_1} \setminus \{ R1.allocate(sId) \} \)
- \( R1.allocate(sId), x < \text{reset} \land y \geq \text{delay}, \) counter ++, \( y := 0 \)
- \( l_0 \)
- \( l_1 \)

- \( \text{delay} = 5 \)
- \( \text{reset} = 100 \)

- Let \( \sigma = (2, R1.allocate(1)) \cdot (1, R1.allocate(2)) \cdot (1, R1.allocate(1)). \)
- \( \text{Dom}(\sigma) = \{1, 2\} \), we have two monitor instances.

\[ \Pi = 1 \]

- \( \sigma \downarrow_1 = (2, R1.allocate(1)) \cdot (2, R1.allocate(1)). \)
- \( E_{\varphi_A(1)} = (2, R1.allocate(1)) \cdot (5, R1.allocate(1)). \)

\[ \Pi = 2 \]

- \( \sigma \downarrow_2 = (3, R1.allocate(2)). \)
- \( E_{\varphi_A(2)} = (3, R1.allocate(2)). \)

\( E_{\varphi_A(1)} \) and \( E_{\varphi_A(2)} \) are sound, transparent, and optimal.

Remark: properties of the global output trace

- \( E_{\varphi_A(p)}(\sigma, t) = \text{merge}(E_{\varphi_A(1)}(\sigma \downarrow_1, t), E_{\varphi_A(2)}(\sigma \downarrow_2, t)) \)
- Only soundness is preserved.
**Parametric case: Example**

\[
\begin{align*}
\sum_{sId} P_1 \setminus \{R1.\text{alloc}(sId)\} & \quad R1.\text{alloc}(sId), \\
& \quad x < \text{reset} \land y \geq \text{delay}, \\
& \quad \text{counter} := 1, x := 0, y := 0
\end{align*}
\]

- \( \text{delay} = 5 \)
- \( \text{reset} = 100 \)

- Let \( \sigma = (2, R1.\text{alloc}(1)) \cdot (1, R1.\text{alloc}(2)) \cdot (1, R1.\text{alloc}(1)) \).
- \( \text{Dom}(\sigma) = \{1, 2\} \), we have two monitor instances.

\[\Pi = 1\]

- \( \sigma \downarrow_1 = (2, R1.\text{alloc}(1)) \cdot (2, R1.\text{alloc}(1)) \).
- \( E_{\varphi_A(1)} = (2, R1.\text{alloc}(1)) \cdot (5, R1.\text{alloc}(1)) \).

\( E_{\varphi_A(1)} \) and \( E_{\varphi_A(2)} \) are sound, transparent, and optimal.

\[\Pi = 2\]

- \( \sigma \downarrow_2 = (3, R1.\text{alloc}(2)) \).
- \( E_{\varphi_A(2)} = (3, R1.\text{alloc}(2)) \).

Remark: properties of the global output trace

- \( E_{\varphi_A(p)}(\sigma, t) = \text{merge}(E_{\varphi_A(1)}(\sigma \downarrow_1, t), E_{\varphi_A(2)}(\sigma \downarrow_2, t)) \)
- Only soundness is preserved.
Parametric case: Example

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

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

- \( \text{delay} = 5 \)
- \( \text{reset} = 100 \)

- Let \( \sigma = (2, R1.\text{alloc}(1)) \cdot (1, R1.\text{alloc}(2)) \cdot (1, R1.\text{alloc}(1)) \).
- \( \text{Dom}(\sigma) = \{1, 2\} \), we have two monitor instances.

\[ \Pi = 1 \]
- \( \sigma \downarrow_1 = (2, R1.\text{alloc}(1)) \cdot (2, R1.\text{alloc}(1)) \).
- \( E_{\varphi, A(1)} = (2, R1.\text{alloc}(1)) \cdot (5, R1.\text{alloc}(1)) \).

\[ E_{\varphi, A(1)} \text{ and } E_{\varphi, A(2)} \text{ are sound, transparent, and optimal.} \]

Remark: properties of the global output trace

- \( E_{\varphi, A(p)}(\sigma, t) = \text{merge}(E_{\varphi, A(1)}(\sigma \downarrow_1, t), E_{\varphi, A(2)}(\sigma \downarrow_2, t)) \)
- Only soundness is preserved.
Parametric case: Example

\[ \Sigma_{sId}^{P_1} \setminus \{ R1.alloc(sId) \} \]

- \( R1.alloc(sId), \ x < \text{reset} \land y \geq \text{delay}, \)
  \( \text{counter} \ := \ 1, \ x := 0, \ y := 0 \)

\[ \Sigma_{sId}^{P_1} \setminus \{ R1.alloc(sId) \} \]

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

- \( \text{delay} = 5 \)
- \( \text{reset} = 100 \)

Let \( \sigma = (2, R1.alloc(1)) \cdot (1, R1.alloc(2)) \cdot (1, R1.alloc(1)) \).

- \( \text{Dom}(\sigma) = \{1, 2\} \), we have two monitor instances.

\( \Pi = 1 \)

- \( \sigma \downarrow_1 = (2, R1.alloc(1)) \cdot (2, R1.alloc(1)) \).
- \( E_{\varphi_A(1)} = (2, R1.alloc(1)) \cdot (5, R1.alloc(1)) \).

\( E_{\varphi_A(1)} \) and \( E_{\varphi_A(2)} \) are sound, transparent, and optimal.

Remark: properties of the global output trace

- \( E_{\varphi_A(\rho)}(\sigma, t) = \text{merge}(E_{\varphi_A(1)}(\sigma\downarrow_1, t), E_{\varphi_A(2)}(\sigma\downarrow_2, t)) \)
- Only soundness is preserved.
Outline - On the Runtime Enforcement of Timed Properties

On the Runtime Enforcement of Untimed Properties

Specifying Timed Properties

Runtime Enforcement of Timed Properties

Extensions

Considering Uncontrollable Events [ICTAC’15]

Considering Events with Data [WODES’14]
  Parameterized Timed Automata with Variables (PTAVs)
  Runtime Enforcement of PTAVs

Application Domains

Conclusions and Future Work
Application Domains

Resource allocation in a client-server model

Clients | Services | Resources
---|---|---
C1 | S1 | R1
C2 | S2 | R2
C3 | S2 | R3

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 $R_1$, there should be a delay of at least 2 t.u. before allocating $R_2$.

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.

---

After releasing $R_1$, there should be a delay of at least 2 t.u. before allocating $R_2$.

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


**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: [RenardFRPJMJ15].
- Enforcement mechanism as delayers+suppression (reducing delay between events): [FalconeJMP16].
- Using Game Theory to synthesize mechanisms
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]
Danièle Beauquier, Joëlle Cohen, and Ruggero Lanotte. 
**Security policies enforcement using finite edit automata.**

Nataliia Bielova and Fabio Massacci.  
**Do you really mean what you actually enforced?**  

Hugues Chabot, Raphael Khoury, and Nadia Tawbi.  
**Generating in-line monitors for Rabin automata.**  
References

Severine Colin and Leonardo Mariani.

Run-time verification.

Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier.

Synthesizing enforcement monitors wrt. the safety-progress classification of properties.

Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier.

Enforcement monitoring wrt. the safety-progress classification of properties.
Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier.  
**What can you verify and enforce at runtime?**  

Yliès Falcone, Klaus Havelund, and Giles Reger.  
**A tutorial on runtime verification.**  

Yliès Falcone, Thierry Jéron, Hervé Marchand, and Srinivas Pinisetty.  
**Runtime enforcement of regular timed properties by suppressing and delaying events.**  
Yliès Falcone, Laurent Mounier, Jean-Claude Fernandez, and Jean-Luc Richier.

**Runtime enforcement monitors: composition, synthesis, and enforcement abilities.**


Philip W. L. Fong.

**Access control by tracking shallow execution history.**


Kevin W. Hamlen, Greg Morrisett, and Fred B. Schneider.

**Computability classes for enforcement mechanisms.**

References

Klaus Havelund and Allen Goldberg.  
**Verify your runs.**  

Martin Leucker and Christian Schallhart.  
**A brief account of runtime verification.**  

Jay Ligatti, Lujo Bauer, and David Walker.  
**Enforcing non-safety security policies with program monitors.**  
Jay Ligatti, Lujo Bauer, and David Walker.

Run-time enforcement of nonsafety policies.


Jarred Adam Ligatti.

Policy Enforcement via Program Monitoring.


Fabio Martinelli and Ilaria Matteucci.

Through modeling to synthesis of security automata.


Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, and Hervé Marchand.

Runtime enforcement of regular timed properties.

Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, and Hervé Marchand. 
Runtime enforcement of parametric timed properties with practical applications. 

Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, Hervé Marchand, Antoine Rollet, and Omer Landry Nguena-Timo. 
Runtime enforcement of timed properties. 
Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, Hervé Marchand, Antoine Rollet, and Omer Nguena-Timo.

**Runtime enforcement of timed properties revisited.**


B. Plattner and J. Nievergelt.

**Special feature: Monitoring program execution: A survey.**


Matthieu Renard, Yliès Falcone, Antoine Rollet, Srinivas Pinisetty, Thierry Jéron, and Hervé Marchand.

**Enforcement of (timed) properties with uncontrollable events.**

References

Fred B. Schneider.

**Enforceable security policies.**

*ACM Transactions on Information and System Security, 3(1), 2000.*

Beth A. Schroeder.

**On-line monitoring: A tutorial.**


Chamseddine Talhi, Nadia Tawbi, and Mourad Debbabi.

**Execution monitoring enforcement for limited-memory systems.**