Enforcement of timed properties

Y. Falcone and Th. Jéron, O. Nguena-Timo,
H. Marchand, S. Pinisetty, A. Rollet

Vacsim Project, Bordeaux

Outline

Context

Validation of a specification *Spec* on a Program $\mathcal{P}$: $\mathcal{P} \models Spec$

- About the specification :
  - Behavioral properties : $Spec = \{\Pi_1, \ldots, \Pi_n\}$
  - Formally defined by a temporal logic formula, a language,...
- A program $\mathcal{P}_\Sigma$: Generator of execution sequences
  - (observable) events of an alphabet Σ
  - $Exec(\mathcal{P}_\Sigma) = \Sigma^\infty$: set of execution sequences

Underlying common alphabet Σ

Runtime techniques to validate $\mathcal{P}_\Sigma$:
$\hookrightarrow$ a "run" of $\mathcal{P}_\Sigma$ satisfies Π

- runtime verification
- runtime enforcement
- testing

Context

Validation of a specification *Spec* on a Program $\mathcal{P}$: $\mathcal{P} \models Spec$

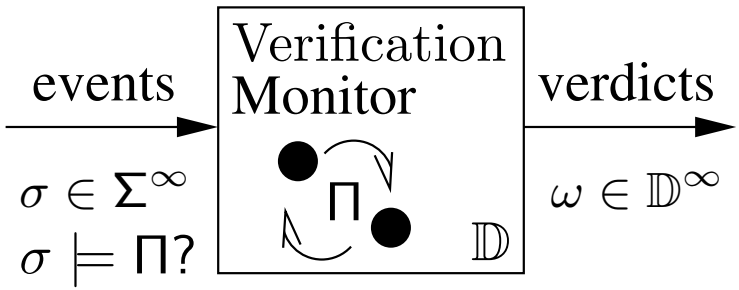
- About the specification :
  - Behavioral properties : $Spec = \{\Pi_1, \ldots, \Pi_n\}$
  - Formally defined by a temporal logic formula, a language,...
- A program $\mathcal{P}_\Sigma$: Generator of execution sequences
  - (observable) events of an alphabet Σ
  - $Exec(\mathcal{P}_\Sigma) = \Sigma^\infty$: set of execution sequences

Underlying common alphabet Σ

Runtime techniques to validate $\mathcal{P}_\Sigma$:
$\hookrightarrow$ a "run" of $\mathcal{P}_\Sigma$ satisfies Π

- runtime verification
- runtime enforcement
- testing

"Classical" runtime validation method : monitoring



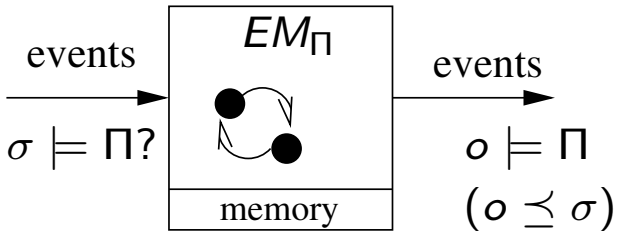
Checking whether a run of the system under scrutiny satisfies a given correctness property

Characteristics

- On-line / Off-line
- Non intrusive technique
- No need to have a model of the system

Enforcement Monitoring : extension of monitoring

"Extension" of runtime verification



Gaining more confidence ?

- Quid when the property is violated ?
- Prevent a misbehavior caused by the possibly unsafe sequence ?

Definition (Runtime Enforcement (RE))

Runtime enforcement is the technique dedicated to *ensure* that a run of a system satisfies a given desired property

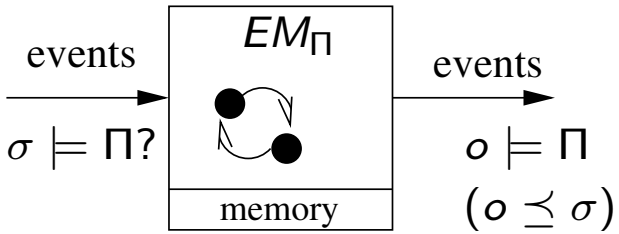
RV and RE share many concepts together :

properties, run, monitor placement

The main conceptual differences lie in the monitor and his purpose.

Enforcement Monitoring : extension of monitoring

"Extension" of runtime verification



Gaining more confidence ?

- Quid when the property is violated ?
- Prevent a misbehavior caused by the possibly unsafe sequence ?

Definition (Runtime Enforcement (RE))

Runtime enforcement is the technique dedicated to *ensure* that a run of a system satisfies a given desired property

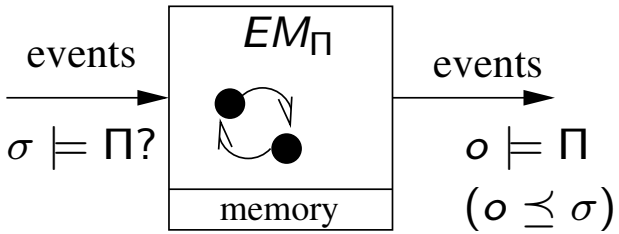
RV and RE share many concepts together :

properties, run, monitor placement

The main conceptual differences lie in the monitor and his purpose.

Enforcement Monitoring : extension of monitoring

"Extension" of runtime verification



Gaining more confidence ?

- Quid when the property is violated ?
- Prevent a misbehavior caused by the possibly unsafe sequence ?

Definition (Runtime Enforcement (RE))

Runtime enforcement is the technique dedicated to *ensure* that a run of a system satisfies a given desired property

RV and RE share many concepts together :

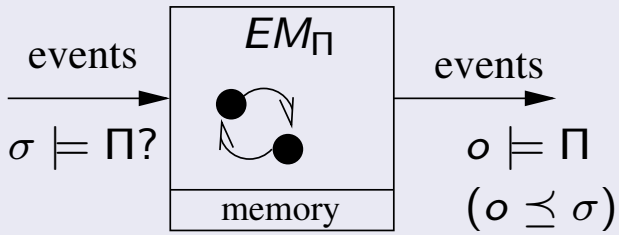
properties, run, monitor placement

The main conceptual differences lie in the monitor and his purpose.

Enforcement Monitor

Shares features with a verification monitor :

- still dedicated to a property Π
- possibly augmented with a memorization mechanism



Enforcement mechanism

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

- reads an input sequence $\sigma \in \Sigma^\infty$
- outputs a new sequence $o \in \Sigma^\infty$
- endowed with a set P of enforcement primitives
  - operates on the memorisation mechanism $\mathcal{M}$
  - delete or insert events using the memory content and the current input

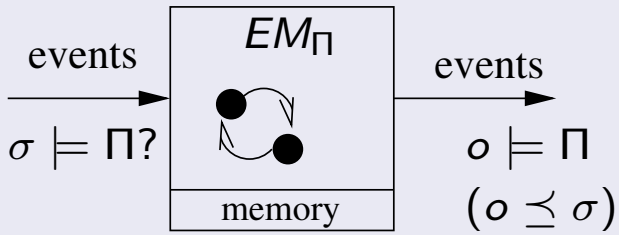
An EM behaves as a function

$$\begin{array}{rccc} [\![\Pi]\!]_P(\cdot): & \Sigma^* & \rightarrow & \Sigma^* \\ & \sigma & \mapsto & o = [\![\Pi]\!]_P(\sigma) \end{array}$$

Enforcement Monitor

Shares features with a verification monitor :

- still dedicated to a property Π
- possibly augmented with a memorization mechanism



Enforcement mechanism

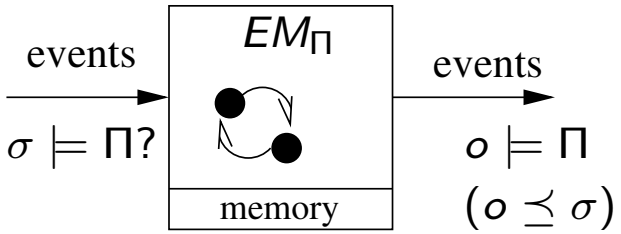
An EM modifies the current execution sequence (sometimes like a "filter")

- reads an input sequence $\sigma \in \Sigma^\infty$
- outputs a new sequence $o \in \Sigma^\infty$
- endowed with a set P of enforcement primitives
  - operates on the memorisation mechanism $\mathcal{M}$
  - delete or insert events using the memory content and the current input

An EM behaves as a function

$$[\![\Pi]\!]_P(\cdot) : \begin{array}{ccc} \Sigma^* & \rightarrow & \Sigma^* \\ \sigma & \mapsto & o = [\![\Pi]\!]_P(\sigma) \end{array}$$

Summary of the results (untimed properties)



Properties of the monitor

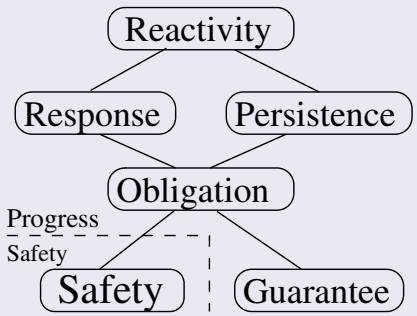
1. Output sequences are correct : soundness
2. Output seq. is the longest correct prefix of the input seq. : transparency

Monitor Synthesis

Generation of a finite state machine equipped with operations (Store, Dump, halt, off). It is synthesized from the street automata encoding the properties

Runtime Enforceable properties

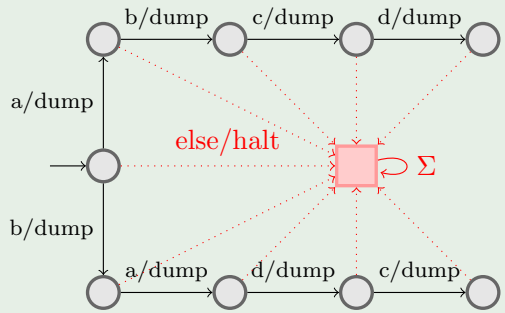
Response properties
within the Safety-Progress classification



Instantiated GEMs : some examples

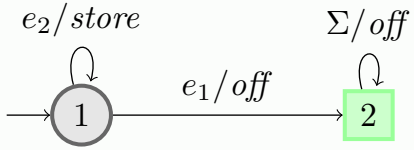
Example (Enforcement of a finitary property)

$Pref\, a \cdot b \cdot c \cdot d \cup Pref\, b \cdot a \cdot d \cdot c$



Example (Enforcement of a guarantee property)

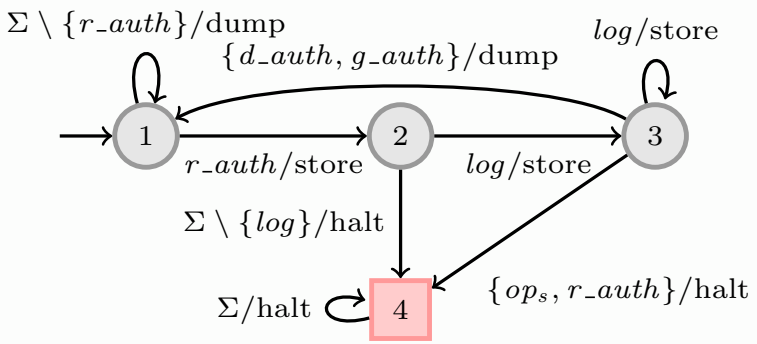
- alphabet=$\{e_1, e_2\}$
- property= "eventually the event e_1 occurs"



Example (Logging authentication requests)

Each occurrence of *r_auth* should be :

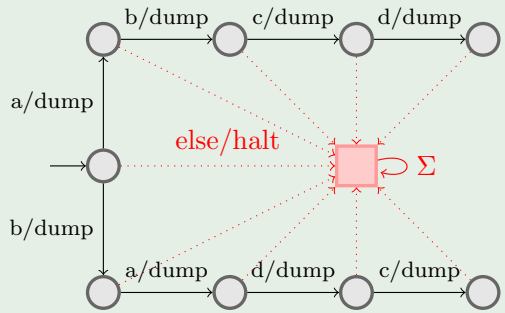
1. written in a log file
2. answered
   - either with a *g_auth* or a *d_auth*
   - without any op_s or *r_auth* meanwhile



Instantiated GEMs : some examples

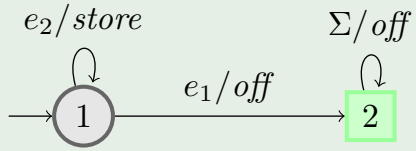
Example (Enforcement of a finitary property)

$Pref\, a \cdot b \cdot c \cdot d \cup Pref\, b \cdot a \cdot d \cdot c$



Example (Enforcement of a guarantee property)

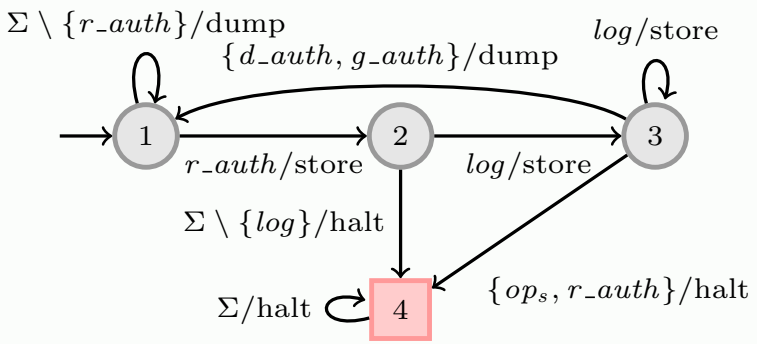
- alphabet=$\{e_1, e_2\}$
- property= "eventually the event e_1 occurs"



Example (Logging authentication requests)

Each occurrence of *r_auth* should be :

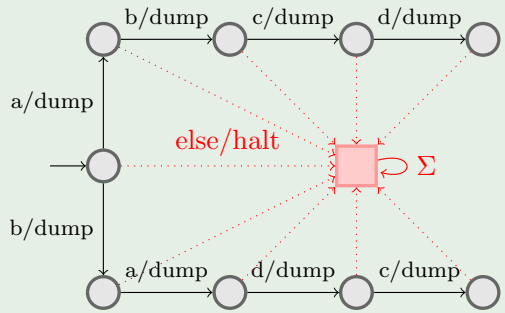
1. written in a log file
2. answered
   - either with a *g_auth* or a *d_auth*
   - without any op_s or *r_auth* meanwhile



Instantiated GEMs : some examples

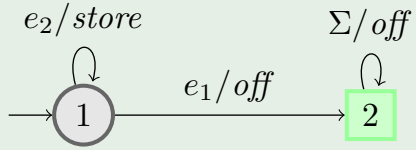
Example (Enforcement of a finitary property)

$Pref\, a \cdot b \cdot c \cdot d \cup Pref\, b \cdot a \cdot d \cdot c$



Example (Enforcement of a guarantee property)

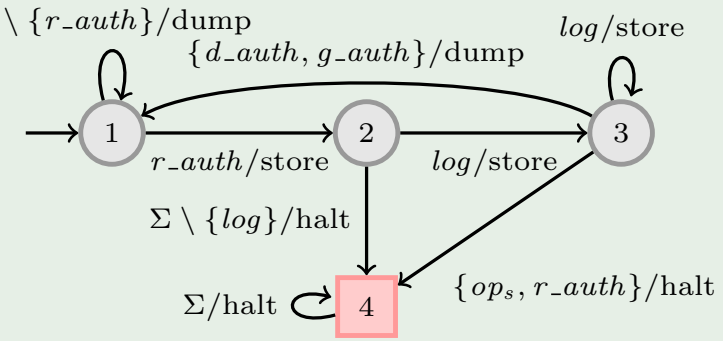
- alphabet=$\{e_1, e_2\}$
- property= "eventually the event e_1 occurs"



Example (Logging authentication requests)

Each occurrence of *r_auth* should be :

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



Outline

Enforcement of timed properties

From untimed to timed properties enforcement

New elements that have to be taken into account

- Input/output Sequences are timed words :
  $\sigma = (a_1, \delta_1), (a_2, \delta_2), \cdots (a_n, \delta_n), \delta_i \in \mathbb{R}_{\geq}, a_i \in \Sigma$
- Property Π described by a timed automaton or a timed logic

Synthesis of the corresponding enforcer ?

- Class of enforceable properties ?
  $\rightarrow$ Focus on safety and reachability properties modeled by TA.
- Model of the enforcer ?
  $\rightarrow$ Similar operations (Store, Dump, halt, off) + memory
  $\rightarrow$ No finite structure
- New notions of transparency
  1. The enforcer can only increase the delay between two actions
  2. The enforcer can try to make up for lost time
- Real-time constraints (e.g. computation time)

Enforcement of timed properties

From untimed to timed properties enforcement

New elements that have to be taken into account

- Input/output Sequences are timed words :
  $\sigma = (a_1, \delta_1), (a_2, \delta_2), \cdots (a_n, \delta_n), \delta_i \in \mathbb{R}_{\geq}, a_i \in \Sigma$
- Property Π described by a timed automaton or a timed logic

Synthesis of the corresponding enforcer ?

- Class of enforceable properties ?
  $\rightarrow$ Focus on safety and reachability properties modeled by TA.
- Model of the enforcer ?
  $\rightarrow$ Similar operations (Store, Dump, halt, off) + memory
  $\rightarrow$ No finite structure
- New notions of transparency
  1. The enforcer can only increase the delay between two actions
  2. The enforcer can try to make up for lost time
- Real-time constraints (e.g. computation time)

Model & Properties

Definition (Timed word)

A timed word over Σ is a finite or infinite sequence of pairs $(a_0, \delta_0)(a_1, \delta_1)(a_2, \delta_2)....$ such that for every $i \in \mathbb{N}$, $a_i \in \Sigma$, and $\forall i$, $\delta_i \in \mathbb{R}^+ \geq 0$.

Definition (Projection of a timed word)

Let $\sigma = (a_0, \delta_0)(a_1, \delta_1)(a_2, \delta_2)\cdots$ The projection of σ is defined as $Proj_\Sigma(\sigma) = a_0 a_1 a_2 \cdots$

Properties represented by timed automata

Safety Property

Reachability Property

Model & Properties

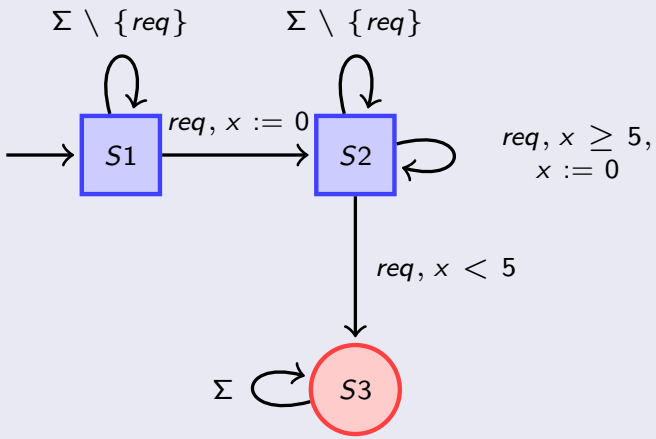
Definition (Timed word)

A timed word over Σ is a finite or infinite sequence of pairs $(a_0, \delta_0)(a_1, \delta_1)(a_2, \delta_2)\ldots$ such that for every $i \in \mathbb{N}$, $a_i \in \Sigma$, and $\forall i$, $\delta_i \in \mathbb{R}^+ \geq 0$.

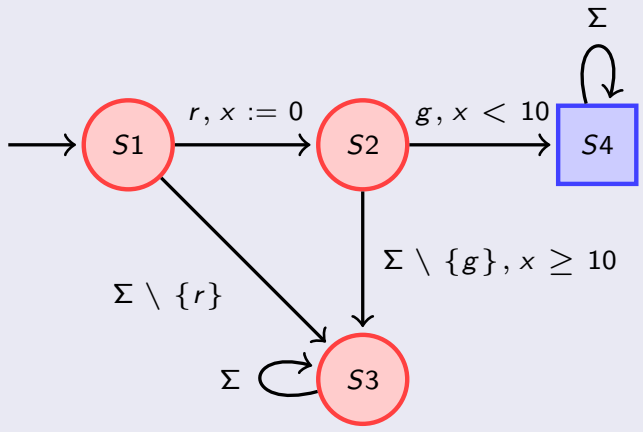
Definition (Projection of a timed word)

Let $\sigma = (a_0, \delta_0)(a_1, \delta_1)(a_2, \delta_2)\cdots$ The projection of σ is defined as $Proj_\Sigma(\sigma) = a_0 a_1 a_2 \cdots$

Properties represented by timed automata

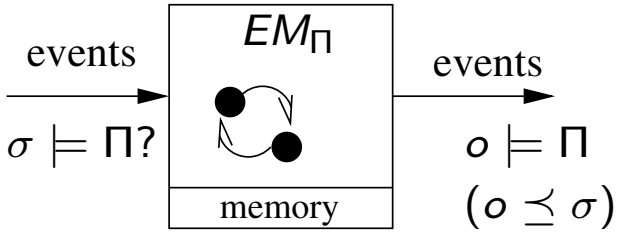


Safety Property



Reachability Property

The enforcer



Memory

Set of pairs (action,delay)

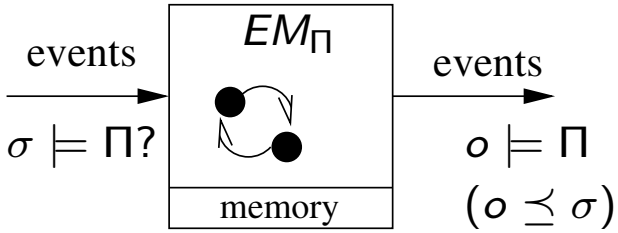
Operations

- **Store** stores the received event and a delay in the memory.
- **Dump** removes the event from memory and releases it as output.
- **Halt** stops reading the inputs, dumps the elements in memory and halts the monitor.
- **Off** (optimization)

Assumption

The time taken by the enforcer to perform the required computations is negligible.

The enforcer



Memory

Set of pairs (action,delay)

Operations

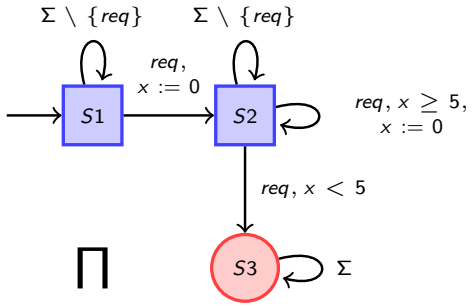
- **Store** stores the received event and a delay in the memory.
- **Dump** removes the event from memory and releases it as output.
- **Halt** stops reading the inputs, dumps the elements in memory and halts the monitor.
- **Off** (optimization)

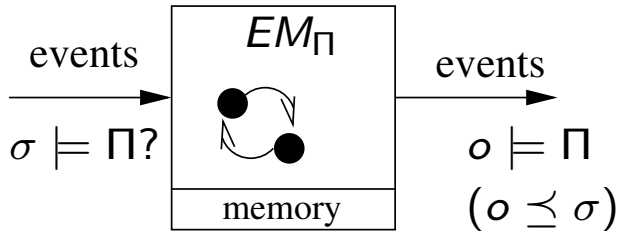
Assumption

The time taken by the enforcer to perform the required computations is negligible.

Outline

Enforcement of a safety property





Soundness conditon

$\forall \sigma \in (\Sigma \times \mathbb{R})^*,\ E(\sigma) \models \Pi.$

Transparency Relations

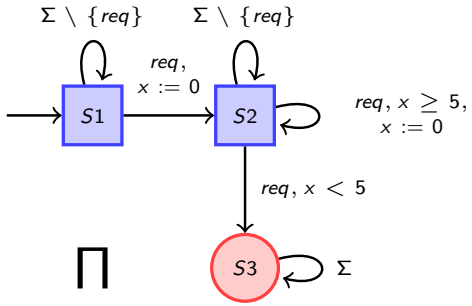
TR1 $Proj_\Sigma(E(\sigma)) \leq_{pref} Proj_\Sigma(\sigma)$

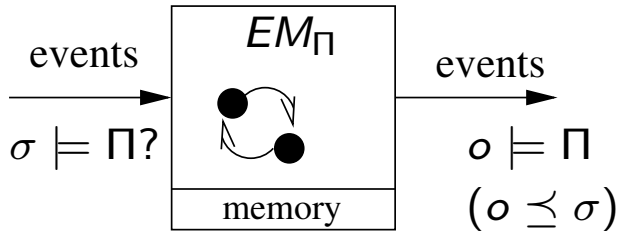
TR2 Let $\sigma = (a_1, \delta_1).(a_2, \delta_2).\cdots.(a_n, \delta_n)$ be the input
$\omega = E(\sigma) = (a'_1, \delta'_1).(a'_2, \delta'_2).\cdots.(a'_n, \delta'_n)$ be the output.

$(\sigma, \omega) \in TR2$ if and only if

1. $\forall 1 \leq i \leq n, \delta_i \leq \delta'_i$
2. $\forall 1 \leq i \leq n, \forall \delta''_i$ where $\delta_i \leq \delta''_i \leq \delta'_i \Rightarrow \omega[1..i-1](a'_i, \delta''_i) \not\models \Pi$

Enforcement of a safety property





Soundness conditon

$\forall \sigma \in (\Sigma \times \mathbb{R})^*,\ E(\sigma) \models \Pi.$

Transparency Relations

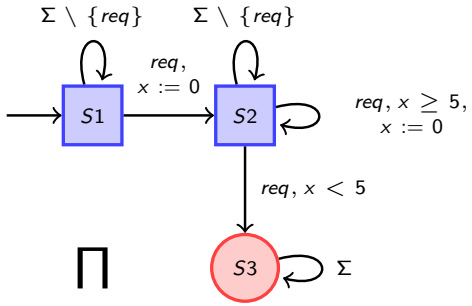
TR1 $Proj_\Sigma(E(\sigma)) \leq_{pref} Proj_\Sigma(\sigma)$

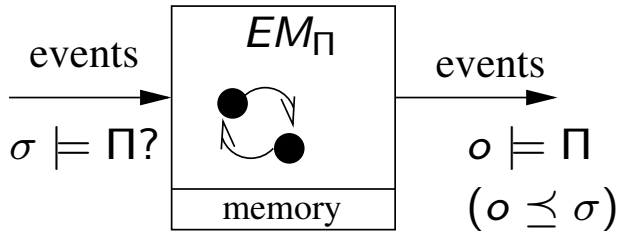
TR2 Let $\sigma = (a_1, \delta_1).(a_2, \delta_2). \cdots .(a_n, \delta_n)$ be the input
$\omega = E(\sigma) = (a'_1, \delta'_1).(a'_2, \delta'_2). \cdots .(a'_n, \delta'_n)$ be the output.

$(\sigma, \omega) \in TR2$ if and only if

1. $\forall 1 \leq i \leq n, \delta_i \leq \delta'_i$
2. $\forall 1 \leq i \leq n, \forall \delta''_i$ where $\delta_i \leq \delta''_i \leq \delta'_i \Rightarrow \omega[1..i-1](a'_i, \delta''_i) \not\models \Pi$

Enforcement of a safety property





Soundness conditon

$\forall \sigma \in (\Sigma \times \mathbb{R})^*$, $E(\sigma) \models \Pi$.

Transparency Relations

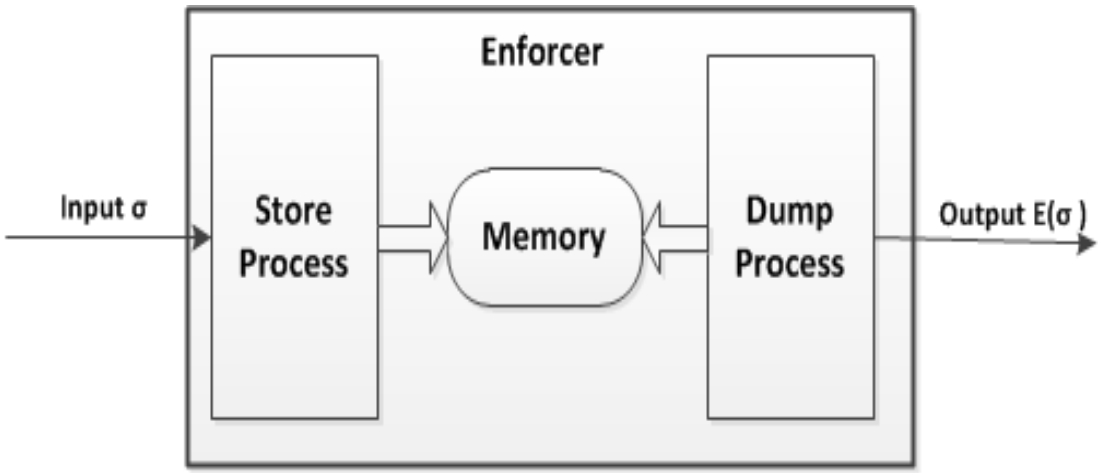
TR1 $Proj_\Sigma(E(\sigma)) \leq_{pref} Proj_\Sigma(\sigma)$

TR2 Let $\sigma = (a_1, \delta_1).(a_2, \delta_2). \cdots .(a_n, \delta_n)$ be the input
$\omega = E(\sigma) = (a'_1, \delta'_1).(a'_2, \delta'_2). \cdots .(a'_n, \delta'_n)$ be the output.

$(\sigma, \omega) \in TR2$ if and only if

1. $\forall 1 \leq i \leq n, \delta_i \leq \delta'_i$
2. $\forall 1 \leq i \leq n, \forall \delta''_i$ where $\delta_i \leq \delta''_i \leq \delta'_i \Rightarrow \omega[1..i-1](a'_i, \delta''_i) \not\models \Pi$

Semantic of the enforcer (to fulfill TR1 and TR2)



Configurations $\mathbf{C} = (\sigma_\mathbf{d}, \sigma_\mathbf{s}, \sigma_\mathbf{f}, \mathbf{s}, \mathbf{d})$

- $\sigma_\mathbf{d} = \mathbf{E}(\sigma)$: sequence of events delivered as output by the enforcer.
- $\sigma_\mathbf{s}$: sequence of events which is stored in its memory.
- $\sigma_\mathbf{f}$: sequence of input events that remain to be read
- $\mathbf{s}$: clock which keeps track of the time since last store.
- $\mathbf{d}$: clock which keeps track of the time since last dump.

Semantic of the enforcer (to fulfill TR1 and TR2)

Store rule

$$\frac{C = (\sigma_d, \sigma_s, \sigma_f, s, d); \sigma_f = (a, \delta).\sigma_f'; s = \delta; \delta' = update(\delta, a, \sigma_d, \sigma_s); \delta' < \infty}{C \xrightarrow{store} (\sigma_d, \sigma_s.(a, \delta'), \sigma_f', 0, d);}$$

with $Update(\delta, a, \sigma_d, \sigma_s) = min\{\delta' \in \mathbb{R}^+ : \delta' \geq \delta, \Delta(s_{init}, \sigma_d.\sigma_s.(a, \delta')) \notin Bad\}$

Halt rule

$$\frac{C = (\sigma_d, \sigma_s, \sigma_f, s, d); \sigma_f = (a, \delta).\sigma_f'; s = \delta; \delta' = update(\delta, a, \sigma_d, \sigma_s); \delta' = \infty}{C \xrightarrow{halt} (\sigma_d, \sigma_s.(halt, \delta), \epsilon, 0, d)}$$

Dump rule

$$\frac{C = (\sigma_d, \sigma_s, \sigma_f, s, d); \sigma_s = (a, \delta).\sigma_s'; d = \delta; a \neq halt}{C \xrightarrow{dump} (\sigma_d.(a, \delta), \sigma_s', \sigma_f, s, 0);}$$

Default rule

$$\frac{C = (\sigma_d, \sigma_s, \sigma_f, s, d);}{C \xrightarrow{\delta} (\sigma_d, \sigma_s, \sigma_f, s+\delta, d+\delta)}$$

Semantic of the enforcer (to fulfill TR1 and TR2)

Store rule

$$\frac{C = (\sigma_d, \sigma_s, \sigma_f, s, d); \sigma_f = (a, \delta).\sigma_f'; s = \delta; \delta' = update(\delta, a, \sigma_d, \sigma_s); \delta' < \infty}{C \xrightarrow{store} (\sigma_d, \sigma_s.(a, \delta'), \sigma_f', 0, d);}$$

with $Update(\delta, a, \sigma_d, \sigma_s) = min\{\delta' \in \mathbb{R}^+ : \delta' \geq \delta, \Delta(s_{init}, \sigma_d.\sigma_s.(a, \delta')) \notin Bad\}$

Halt rule

$$\frac{C = (\sigma_d, \sigma_s, \sigma_f, s, d); \sigma_f = (a, \delta).\sigma_f'; s = \delta; \delta' = update(\delta, a, \sigma_d, \sigma_s); \delta' = \infty}{C \xrightarrow{halt} (\sigma_d, \sigma_s.(halt, \delta), \epsilon, 0, d)}$$

Dump rule

$$\frac{C = (\sigma_d, \sigma_s, \sigma_f, s, d); \sigma_s = (a, \delta).\sigma_s'; d = \delta; a \neq halt}{C \xrightarrow{dump} (\sigma_d.(a, \delta), \sigma_s', \sigma_f, s, 0);}$$

Default rule

$$\frac{C = (\sigma_d, \sigma_s, \sigma_f, s, d);}{C \xrightarrow{\delta} (\sigma_d, \sigma_s, \sigma_f, s+\delta, d+\delta)}$$

Semantic of the enforcer (to fulfill TR1 and TR2)

Store rule

$$\frac{C = (\sigma_d, \sigma_s, \sigma_f, s, d); \sigma_f = (a, \delta).\sigma_f'; s = \delta; \delta' = update(\delta, a, \sigma_d, \sigma_s); \delta' < \infty}{C \xrightarrow{store} (\sigma_d, \sigma_s.(a, \delta'), \sigma_f', 0, d);}$$

with $Update(\delta, a, \sigma_d, \sigma_s) = min\{\delta' \in \mathbb{R}^+ : \delta' \geq \delta, \Delta(s_{init}, \sigma_d.\sigma_s.(a, \delta')) \notin Bad\}$

Halt rule

$$\frac{C = (\sigma_d, \sigma_s, \sigma_f, s, d); \sigma_f = (a, \delta).\sigma_f'; s = \delta; \delta' = update(\delta, a, \sigma_d, \sigma_s); \delta' = \infty}{C \xrightarrow{halt} (\sigma_d, \sigma_s.(halt, \delta), \epsilon, 0, d)}$$

Dump rule

$$\frac{C = (\sigma_d, \sigma_s, \sigma_f, s, d); \sigma_s = (a, \delta).\sigma_s'; d = \delta; a \neq halt}{C \xrightarrow{dump} (\sigma_d.(a, \delta), \sigma_s', \sigma_f, s, 0);}$$

Default rule

$$\frac{C = (\sigma_d, \sigma_s, \sigma_f, s, d);}{C \xrightarrow{\delta} (\sigma_d, \sigma_s, \sigma_f, s+\delta, d+\delta)}$$

Semantic of the enforcer (to fulfill TR1 and TR2)

Store rule

$$\frac{C = (\sigma_d, \sigma_s, \sigma_f, s, d); \sigma_f = (a, \delta).\sigma_f'; s = \delta; \delta' = update(\delta, a, \sigma_d, \sigma_s); \delta' < \infty}{C \xrightarrow{store} (\sigma_d, \sigma_s.(a, \delta'), \sigma_f', 0, d);}$$

with $Update(\delta, a, \sigma_d, \sigma_s) = min\{\delta' \in \mathbb{R}^+ : \delta' \geq \delta, \Delta(s_{init}, \sigma_d.\sigma_s.(a, \delta')) \notin Bad\}$

Halt rule

$$\frac{C = (\sigma_d, \sigma_s, \sigma_f, s, d); \sigma_f = (a, \delta).\sigma_f'; s = \delta; \delta' = update(\delta, a, \sigma_d, \sigma_s); \delta' = \infty}{C \xrightarrow{halt} (\sigma_d, \sigma_s.(halt, \delta), \epsilon, 0, d)}$$

Dump rule

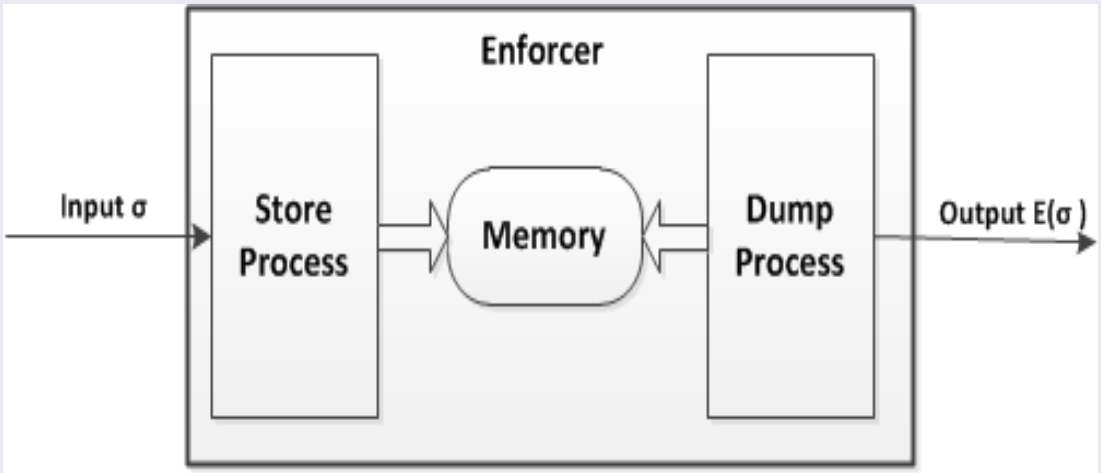
$$\frac{C = (\sigma_d, \sigma_s, \sigma_f, s, d); \sigma_s = (a, \delta).\sigma_s'; d = \delta; a \neq halt}{C \xrightarrow{dump} (\sigma_d.(a, \delta), \sigma_s', \sigma_f, s, 0);}$$

Default rule

$$\frac{C = (\sigma_d, \sigma_s, \sigma_f, s, d);}{C \xrightarrow{\delta} (\sigma_d, \sigma_s, \sigma_f, s+\delta, d+\delta)}$$

Realizing the enforcer

Architecture



Realizing the enforcer (Algorithms)

Algorithm 1 : DumpProcess

$d \leftarrow 0$
while true **do**
  await $(|\sigma_s| \geq 1)$
  $(a, \delta) \leftarrow$ dequeue (σ_s)
  if $a == halt$ **then**
    Halt the DumpProcess
  else
    wait $(\delta - d)$
    dump (a)
    $d \leftarrow 0$
  end if
end while

Algorithm 2 : StoreProcess

$(l, X) \leftarrow (l_{Init}, 0)$
while true **do**
  $(a, \delta) \leftarrow$ *await event*
  if $(post(l, a, X, \delta) \in Bad)$ **then**
    $\delta' \leftarrow$ *update*
    if $\delta' = \infty$ **then**
      enqueue(halt,δ)
      Halt the StoreProcess
    end if
  else
    $\delta' \leftarrow \delta$
  end if
  $(l, X) \leftarrow$ post (l, a, X, δ')
  enqueue (a, δ')
end while

Realizing the enforcer (Algorithms)

Algorithm 1 : DumpProcess

$d \leftarrow 0$
while true **do**
  await $(|\sigma_s| \geq 1)$
  $(a, \delta) \leftarrow$ dequeue (σ_s)
  if $a == halt$ **then**
    Halt the DumpProcess
  else
    wait $(\delta - d)$
    dump (a)
    $d \leftarrow 0$
  end if
end while

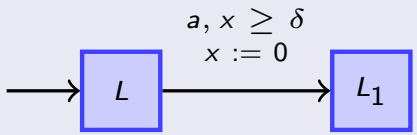
Algorithm 2 : StoreProcess

$(l, X) \leftarrow (l_{Init}, 0)$
while true **do**
  $(a, \delta) \leftarrow$ *await event*
  if $(post(l, a, X, \delta) \in Bad)$ **then**
    $\delta' \leftarrow$ *update*
    if $\delta' = \infty$ **then**
      enqueue(halt,δ)
      Halt the StoreProcess
    end if
  else
    $\delta' \leftarrow \delta$
  end if
  $(l, X) \leftarrow$ post (l, a, X, δ')
  enqueue (a, δ')
end while

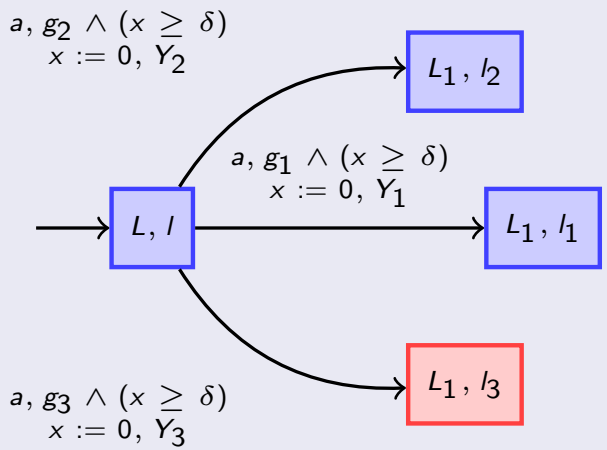
Algorithm 2

StoreProcess(a, δ) (update function)

Consider the TA $\mathcal{A}_\Pi$ modeling the property Π and the TA $\mathcal{A}$



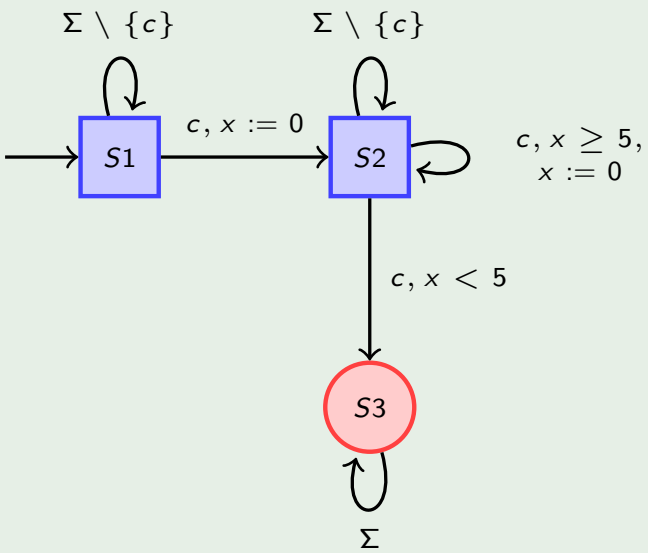
- $\mathcal{B} = \mathcal{A} \times \mathcal{A}_\Pi(l, X)$ where l is the current locality and X the current values of the clocks, stored in the StoreProcess
- find the optimal timed path from (L, l) path to (L_1, l') with $l' \in \mathit{Good}$



$\Rightarrow$ All operations available in the UppAal libraries

Example

Example

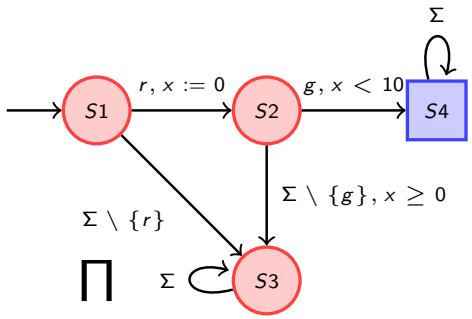


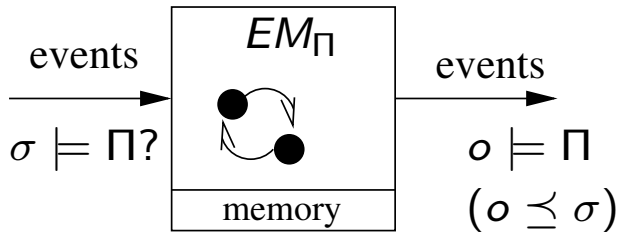
$$\sigma = (a,1).(c,3).(c,1)$$

$$(\epsilon, \epsilon, (a,1)(c,3)(c,1), 0, 0) \quad t \triangleq 0$$
$$\downarrow \text{Rule 4}$$
$$(\epsilon, \epsilon, (a,1)(c,3)(c,1), 1, 1) \quad t = 1$$
$$\downarrow \text{Store}$$
$$(\epsilon, (a,1), (c,3).(c,1), 0, 1) \quad t = 1$$
$$\downarrow \text{Dump}$$
$$((a,1), \epsilon, (c,3).(c,1), 0, 0) \quad t = 1$$
$$\downarrow \text{Rule4}$$
$$((a,1), \epsilon, (c,3).(c,1), 3, 3) \quad t = 4$$
$$\downarrow \text{Store}$$
$$((a,1), (c,3), (c,1), 0, 3) \quad t = 4$$
$$\downarrow \text{Dump}$$
$$((a,1).(c,3), \epsilon, (c,1), 0, 0) \quad t = 4$$
$$\downarrow \text{Rule4}$$
$$((a,1).(c,3), \epsilon, (c,1), 1, 1) \quad t = 5$$
$$\downarrow \text{Store}$$
$$((a,1).(c,3), (c,5), \epsilon, 0, 1) \quad t = 5$$
$$\downarrow \text{Rule4}$$
$$((a,1).(c,3), (c,5), \epsilon, 4, 5) \quad t = 9$$
$$\downarrow \text{Dump}$$
$$((a,1).(c,3).(c,5), \epsilon, \epsilon, 4, 0) \quad t = 9$$

Outline

Enforcement of a reachability property





Soundness condition

$\forall \sigma \in (\Sigma \times \mathbb{R})^*, E(\sigma) \models \Pi$

Transparency relations

TR1 $Proj_\Sigma(E(\sigma)) = Proj_\Sigma(\sigma)$

TR2 Let $\sigma = (a_1, \delta_1).(a_2, \delta_2).\cdots.(a_n, \delta_n)$ be the input to the enforcer and $\omega = (a'_1, \delta'_1).(a'_2, \delta'_2).\cdots.(a'_n, \delta'_n)$ be the output.

$(\sigma, \omega) \in TR2$ if and only if

1. $\forall 1 \leq i \leq |\omega|, \delta_i \leq \delta'_i$
2. $\forall i, \nexists (\delta_i'')_{i \leq n} \sum_{n=1}^{i-1} \delta_i'' \leq \sum_{n=1}^{i-1} \delta_i' \wedge (\delta_i'', a'_i)_{i \leq n} \models \Pi \wedge \delta_i \leq \delta_i''$

Semantic of the enforcer

store rule

$$\frac{C = (\sigma_d, (a_1, \delta_1).(a_2, \delta_2).\cdots(a_n, \delta_n), \sigma_f, s, d); \sigma_f = (a_{n+1}, \delta_{n+1}).\sigma'_f; s = \delta_{n+1}, \quad (\delta_i)_{i\leq n+1} = update((\delta_i)_{i\leq n}, (a_i)_{i\leq n+1}, \sigma_d);}{C \xrightarrow{store} (\sigma_d, (a_1, \delta_1).(a_2, \delta_2).\cdots(a_{n+1}, \delta_{n+1}), \sigma'_f, 0, d);}$$

$$Upadte((\delta_i)_{i\leq n}, (a_i)_{i\leq n+1}, \sigma_d) = \begin{cases} \min\{(\delta'_i)_{i\leq n+1} \in \mathbb{R}^{n+1}, \forall i \geq n+1, \delta'_i \geq \delta_i \wedge \\ \Delta(s_{init}, \sigma_d.(a_i, \delta'_i)_{i\leq n+1}) \in Good\} \\ \\ (\delta_i)_{i\leq n} \text{ if } (\delta'_i)_{i\leq n+1} \text{ not exists} \end{cases}$$

More concretely :

- consider the TA $\mathcal{A}_\Pi$ modeling the property Π and the TA $\mathcal{A}$

- $\mathcal{B} = \mathcal{A} \times \mathcal{A}_\Pi$ and check reachability of location $(l_{n+2}, Good)$
- If OK, then find the optimal timed path from initial state to $(Good, l_{n+2})$

Semantic of the enforcer

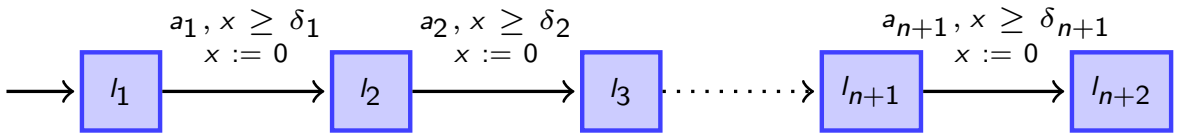
store rule

$$\frac{C = (\sigma_d, (a_1, \delta_1).(a_2, \delta_2).\cdots(a_n, \delta_n), \sigma_f, s, d); \sigma_f = (a_{n+1}, \delta_{n+1}).\sigma_f'; s = \delta_{n+1}, \quad (\delta_i)_{i\leq n+1} = update((\delta_i)_{i\leq n}, (a_i)_{i\leq n+1}, \sigma_d);}{C \xrightarrow{store} (\sigma_d, (a_1, \delta_1).(a_2, \delta_2).\cdots(a_{n+1}, \delta_{n+1}), \sigma_f', 0, d);}$$

$$Upadte((\delta_i)_{i\leq n}, (a_i)_{i\leq n+1}, \sigma_d) = \begin{cases} \min\{(\delta_i')_{i\leq n+1} \in \mathbb{R}^{n+1}, \forall i \geq n+1, \delta_i' \geq \delta_i \wedge \\ \Delta(s_{init}, \sigma_d.(a_i, \delta_i')_{i\leq n+1}) \in Good\} \\ \\ (\delta_i)_{i\leq n} \text{ if } (\delta_i')_{i\leq n+1} \text{ not exists} \end{cases}$$

More concretely :

- consider the TA $\mathcal{A}_\Pi$ modeling the property Π and the TA $\mathcal{A}$



- $\mathcal{B} = \mathcal{A} \times \mathcal{A}_\Pi$ and check reachability of location $(l_{n+2}, Good)$
- If OK, then find the optimal timed path from initial state to $(Good, l_{n+2})$

Example

Example

Soon Available

Outline

Conclusion

- Enforcement of timed properties
  - Focus on Safety/Reachability properties
  - Need to introduce new notions of soundness and transparency
  - No more finite structure to encode the enforcer
- Ongoing/Future work
  - New Transparency conditions
  - New Enforcer architecture
  - New class of properties
  - Implementation using UppAal librairies
  - Test on Case studies (ideas ?)