Enforcement Monitoring wrt. the Safety-Progress Classification of Properties

DCS days'09

Ylies Falcone   Jean-Claude Fernandez   Laurent Mounier

Verimag, Grenoble Universities

March 26 2009, Autrans

Validation of a policy *Pol* on a system S: $S \models Pol$

- About the policy:
  - Behavioral properties
  - Formally defined by a temporal logical formula, a language,...
- A program $\mathcal{P}_\Sigma$: Generator of execution sequences
  - (observable) events of an alphabet Σ
  - $Exec(\mathcal{P}_\Sigma) = \Sigma^\infty = \Sigma^* \cup \Sigma^\omega$: set of execution sequences
- Several approaches to validate *Pol*:
  - (formal) proof
  - testing
  - runtime validation

Validation of a policy *Pol* on a system S: $S \models Pol$

- About the policy:
  - Behavioral properties
  - Formally defined by a temporal logical formula, a language,...
- A program $\mathcal{P}_\Sigma$: Generator of execution sequences
  - (observable) events of an alphabet Σ
  - $Exec(\mathcal{P}_\Sigma) = \Sigma^\infty = \Sigma^* \cup \Sigma^\omega$: set of execution sequences
- Several approaches to validate *Pol*:
  - (formal) proof
  - testing
  - runtime validation

"Classical" runtime validation method: monitoring

- Instrument the underlying program to observe relevant events
- A monitor acts as an oracle for the property (validation/violation)

Enforcement Monitoring: extension of monitoring

Gaining more confidence?

- Quid when the property is violated?
- Prevent a misbehavior of the program?

Enforcement Monitoring: extension of monitoring

Gaining more confidence?

- Quid when the property is violated?
- Prevent a misbehavior of the program?

Informal principle [Schneider, Ligatti and al.]

1. Correct original execution sequences remained unchanged (transparency)
2. Incorrect original execution sequences are changed into their longest correct prefix (soundness)

Enforcement Monitoring: extension of monitoring

Gaining more confidence?

- Quid when the property is violated?
- Prevent a misbehavior of the program?

Informal principle [Schneider, Ligatti and al.]

1. Correct original execution sequences remained unchanged (transparency)
2. Incorrect original execution sequences are changed into their longest correct prefix (soundness)

In this work:

- **Synthesis** of "enforcers" from "property recognizers" (ω-automata)
- **Characterization** of the "enforceable properties" wrt. the Safety-Progress Classification
- **Prototype toolbox** implementing those features

Outline

Outline

1. The Safety-Progress Classification of Properties [Manna,Pnueli]
   - Overview
   - The automata view
2. Property Enforcement via Enforcement Monitors
3. Enforcement Monitoring wrt. the SP Classification
4. A prototype toolbox

General classification of linear temporal properties

Alternative to the Safety-Liveness classification

General classification of linear temporal properties

Alternative to the Safety-Liveness classification

Finer-grain definition of classes of properties

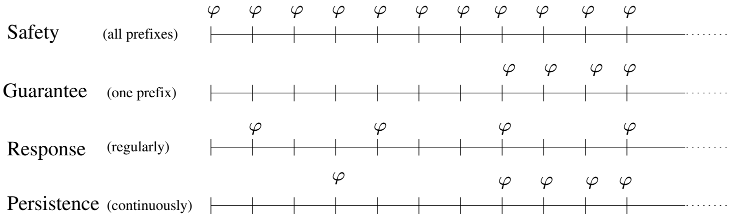
- basic classes: safety, guarantee, response, persistence
- compound classes: obligation, reactivity (cf. papers)

General classification of linear temporal properties

Alternative to the Safety-Liveness classification

Finer-grain definition of classes of properties

- basic classes: safety, guarantee, response, persistence
- compound classes: obligation, reactivity (cf. papers)

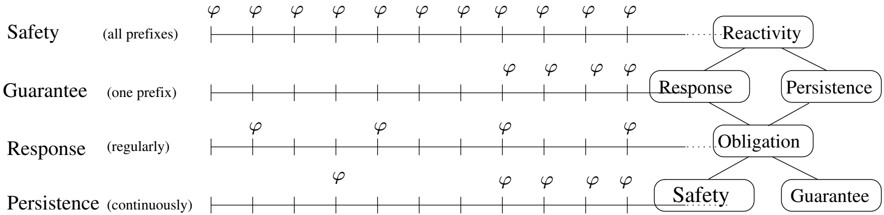


General classification of linear temporal properties

Alternative to the Safety-Liveness classification

Finer-grain definition of classes of properties

- basic classes: safety, guarantee, response, persistence
- compound classes: obligation, reactivity (cf. papers)

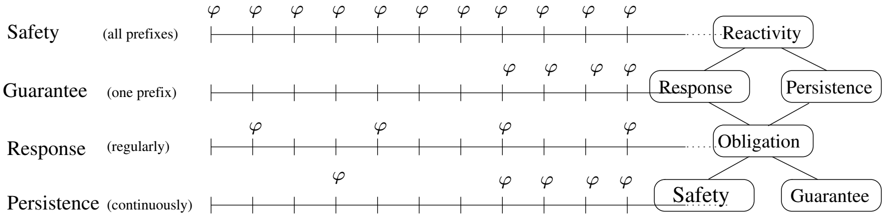


General classification of linear temporal properties

Alternative to the Safety-Liveness classification

Finer-grain definition of classes of properties

- basic classes: safety, guarantee, response, persistence
- compound classes: obligation, reactivity (cf. papers)



Characterization according to 4 views
↪ language, logical, topological, **automata**

Streett automata

The automata view:

- finite state automata: Streett automata
- classes of properties depend on **syntactic restrictions** on the automata

Definition of a deterministic Streett automaton

A tuple $(Q, q_{\text{init}}, \Sigma, \longrightarrow, \{(R_1, P_1), \ldots, (R_m, P_m)\})$

- Q is the set of automaton states ($q_{\text{init}} \in Q$ is the initial state),
- total function $\longrightarrow: Q \times \Sigma \rightarrow Q$ is the transition function,
- $\{(R_1, P_1), \ldots, (R_m, P_m)\}$ is the set of accepting pairs, $\forall i \leq n$,
  - $R_i \subseteq Q$ are the sets of recurrent states,
  - and $P_i \subseteq Q$ are the sets of persistent states.

$\hookrightarrow$ Basic classes $\Rightarrow m = 1$ and R_1, P_1 are noted R, P

Acceptance condition for **Finite sequences**

For $\sigma \in \Sigma^*$ such that $|\sigma| = n$, we say that $\mathcal{A}$ accepts σ if
$(\exists q_0, \ldots, q_n \in Q^{\mathcal{A}} \cdot run(\sigma, \mathcal{A}) = q_0 \cdots q_n \wedge q_0 = q_{\mathrm{init}}{}^{\mathcal{A}}$ and $q_n \in P \cup R)$

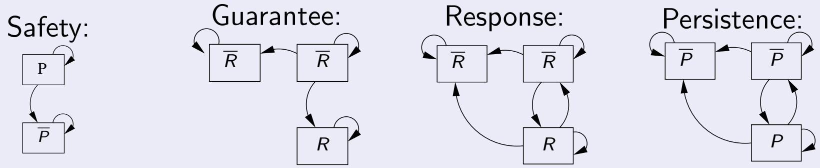
Acceptance condition for **Infinite sequences**

For $\sigma \in \Sigma^\omega$, we say that $\mathcal{A}$ accepts σ if
$vinf(\sigma, \mathcal{A}) \cap R \neq \emptyset \vee vinf(\sigma, \mathcal{A}) \subseteq P$
where $vinf(\sigma, \mathcal{A})$: set of states visited infinitely often

The automata view

Classification according to syntactic restrictions on automata

- **safety**: $R = \emptyset$ and no transition from $q \in \overline{P}$ to $q' \in P$.
- **guarantee**: $P = \emptyset$ and no transition from $q \in R$ to $q' \in \overline{R}$
- **response**: $P = \emptyset$
- **persistence**: $R = \emptyset$
- *m-obligation*: m-automaton (composition of safety and guarantee, cf. the paper)
- *m-reactivity*: any unrestricted m-automaton



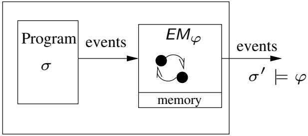
Outline

1. The Safety-Progress Classification of Properties [Manna,Pnueli]
2. Property Enforcement via Enforcement Monitors
   - Enforcement Monitors
   - Enforcing a property
3. Enforcement Monitoring wrt. the SP Classification
4. A prototype toolbox

Informal description and requirements

Runtime device: **I/O automaton**:

- processes an execution sequence of an underlying program
- event-by-event
- dedicated to a property φ
- performs an enforcement operation: induces a transformation of the current execution sequence ($\sigma \rightsquigarrow \sigma'$)



Requirements wrt. φ:

- Soundness
- Transparency

Enforcement Monitor

Definition (Enforcement monitor (EM))

A 4-tuple $(Q^{\mathcal{A}_\downarrow}, {q_{\text{init}}}^{\mathcal{A}_\downarrow}, \mathit{Stop}^{\mathcal{A}_\downarrow}, \longrightarrow_{\mathcal{A}_\downarrow})$ with enforcement operations Ops.

- $Q^{\mathcal{A}_\downarrow}$: control states, (${q_{\text{init}}}^{\mathcal{A}_\downarrow} \in Q^{\mathcal{A}_\downarrow}$ is the initial state)
- $\mathit{Stop}^{\mathcal{A}_\downarrow}$ is the set of stopping states ($\mathit{Stop}^{\mathcal{A}_\downarrow} \subseteq Q^{\mathcal{A}_\downarrow}$)
- $\longrightarrow_{\mathcal{A}_\downarrow}: Q^{\mathcal{A}_\downarrow} \times \Sigma \rightarrow \mathit{Ops} \times Q^{\mathcal{A}_\downarrow}$ is the transition function.

Enforcement Monitor

Definition (Enforcement monitor (EM))

A 4-tuple $(Q^{\mathcal{A}_\downarrow}, {q_{\text{init}}}^{\mathcal{A}_\downarrow}, Stop^{\mathcal{A}_\downarrow}, \longrightarrow_{\mathcal{A}_\downarrow})$ with enforcement operations *Ops*.

- $Q^{\mathcal{A}_\downarrow}$: control states, (${q_{\text{init}}}^{\mathcal{A}_\downarrow} \in Q^{\mathcal{A}_\downarrow}$ is the initial state)
- $Stop^{\mathcal{A}_\downarrow}$ is the set of stopping states ($Stop^{\mathcal{A}_\downarrow} \subseteq Q^{\mathcal{A}_\downarrow}$)
- $\longrightarrow_{\mathcal{A}_\downarrow}: Q^{\mathcal{A}_\downarrow} \times \Sigma \rightarrow Ops \times Q^{\mathcal{A}_\downarrow}$ is the transition function.

Enforcement operations *Ops*:

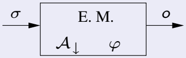
- Take as inputs an event and a memory content (i.e., a sequence of events) to produce a new memory content and an output sequence.
- The set $Ops = \{halt, store, dump\}$:
  - halt stops the program
  - store memorizes input event in the memory
  - dump outputs the current memory content

Enforcing a property

Definition (Property Enforcement)

$Enf(\mathcal{A}_{\downarrow}, \varphi, \mathcal{P}_{\Sigma})$: $\forall \sigma \in Exec(\mathcal{P}_{\Sigma})$:

- $\mathcal{A}_{\downarrow}$ transforms $\sigma \in \Sigma^{\infty}$ into $o \in \Sigma^{\infty}$:



- Correct execution sequences are not changed:

$$\sigma \models \varphi \Rightarrow \sigma = o$$

- Incorrect execution sequences are truncated to their longest correct prefix

$$\sigma \not\models \varphi \Rightarrow o = Max(Pref(\varphi, \sigma))$$

Outline

1. The Safety-Progress Classification of Properties [Manna,Pnueli]
2. Property Enforcement via Enforcement Monitors
3. Enforcement Monitoring wrt. the SP Classification
   - Synthesizing EMs wrt. the Safety-Progress Classification
   - Enforceable Properties
4. A prototype toolbox

Transformation for basic classes of properties

Transformations for safety, guarantee, obligation, and response properties

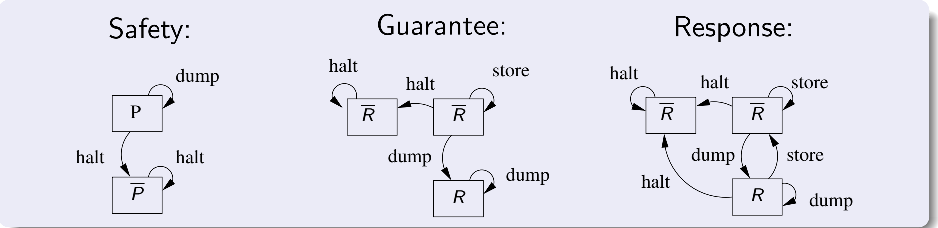
Informal behavior of the expected EM $\mathcal{A}_{\downarrow\varphi}$

- Current execution sequence (now) satisfies the property
  $\Rightarrow$ **dump** current event and memory content
- Current execution sequence does not (yet) satisfy the property
  $\Rightarrow$ **store** each input event
- Current execution sequence deviates (for ever) from the property
  $\Rightarrow$ **halt** immediately the underlying program with a *halt* operation.

Transformation for basic classes of properties

On the initial Streett automaton:

- Reaching a P or R state: **dump** operation
- Reaching a $\overline{P}$ or $\overline{R}$ in Reach of an P or R state: **store** operation
- Reaching a $\overline{P}$ or $\overline{R}$ not in Reach of an P or R state: **halt** operation



Non-Enforceable Properties

Persistence properties are not enforceable by our enforcement monitors.

Example

"an incorrect use of operation *op* should imply that any future call to *req_auth* will always result in a *deny_auth* answer"

Enforcement limitation:

- decide from a certain point that the underlying program will always produce the event *deny_auth* in response to a *req_auth*
- decision cannot be taken without reading and memorizing first the entire execution sequence.

Straightforward consequence: **reactivity** class is not enforceable

Characterizing the set of **Enforceable Properties**

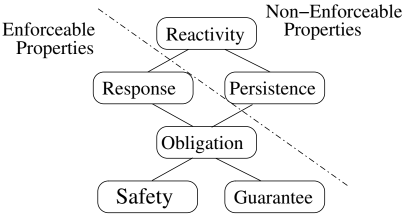
A program $\mathcal{P}_\Sigma$

A property φ (safety, guarantee, obligation or response) recognized by $\mathcal{A}_\varphi$

An EM $\mathcal{A}_{\downarrow\varphi}$ obtained by previous transformations

Theorem

$$(\varphi \in \text{Safety}_\Sigma \wedge \mathcal{A}_{\downarrow\varphi} = \text{TransSafety}(\mathcal{A}_\varphi)) \Rightarrow \mathit{Enf}(\mathcal{A}_{\downarrow\varphi}, \varphi, \mathcal{P}_\Sigma),$$
$$(\varphi \in \text{Guarantee}_\Sigma \wedge \mathcal{A}_{\downarrow\varphi} = \text{TransGuarantee}(\mathcal{A}_\varphi)) \Rightarrow \mathit{Enf}(\mathcal{A}_{\downarrow\varphi}, \varphi, \mathcal{P}_\Sigma).$$
$$(\varphi \in \text{Obligation}_\Sigma \wedge \mathcal{A}_{\downarrow\varphi} = \text{TransObligation}(\mathcal{A}_\varphi)) \Rightarrow \mathit{Enf}(\mathcal{A}_{\downarrow\varphi}, \varphi, \mathcal{P}_\Sigma).$$
$$(\varphi \in \text{Response}_\Sigma \wedge \mathcal{A}_{\downarrow\varphi} = \text{TransResponse}(\mathcal{A}_\varphi)) \Rightarrow \mathit{Enf}(\mathcal{A}_{\downarrow\varphi}, \varphi, \mathcal{P}_\Sigma).$$



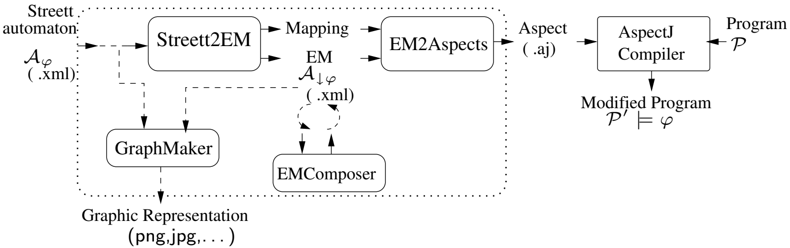
Outline

1. The Safety-Progress Classification of Properties [Manna,Pnueli]
2. Property Enforcement via Enforcement Monitors
3. Enforcement Monitoring wrt. the SP Classification
4. A prototype toolbox

Our prototype toolbox

Tool implementing this approach: 2 main stages and additional features

- Monitor synthesis: **Streett2EM**
  ↪ XSLT transformation (XML to XML)
- Monitor integration: **EM2Aspects**
  ↪ using program-transformation frameworks (here AOP)
- Monitor composition (boolean operations): **EMComposer**
- Graphic representation of Streett automata and EMs: **GraphMaker**



Conclusion

Extensions of property validation at runtime through Enforcement Monitoring

Generic notion of finite-state enforcement monitor

Specification of their enforcement ability wrt. the Safety-Progress Classification of Properties
$\hookrightarrow$ fine-grain characterization of enforceable properties

(Simple) transformations from Streett automata

A prototype toolbox

Future Works

Further study the **practical feasibility** of the approach

- Observable/Controllable events
- data dependency between events
- Memory limitation for the EM

$\hookrightarrow$ Influence on the enforcement ability: how the set of enforceable properties is impacted?

Monitor integration: other program-transformation frameworks ?

- Integration level: source/binaries
- System architecture: distributed/centralized

Assessing the **Toolbox** capability to validate properties