Runtime Verification of Safety/Progress Properties

VERIMAG Lab & Université Joseph Fourier – Grenoble, France

Yliès Falcone and Jean-Claude Fernandez and Laurent Mounier

November 24th 2009, Prague, Czesh Republic
Verimag DCS Team (Distributed & Complex Systems)

(18 permanents, 10 post-doc students, 20 PhD students)

- specification & design of component-based systems
  - embedded systems
  - service-oriented applications

- verification and validation techniques
  - model-based techniques
  - program verification
  - test

- security issues
  - cryptographic protocols
  - software vulnerabilities
Verification and validation activities within DCS

- **model-based techniques**
  - model-checking [IF]
  - model-driven test generation [TGV]
  - simulation, performance evaluation [Glonemo]
  - correct-by-construction paradigm [BIP]

- **program-based techniques**
  - static analysis, shape analysis
  - proof [Coq]

- **execution-based techniques**
  - test execution [j-POST]
  - run-time monitoring, run-time enforcement (j-VETO)
Online runtime monitoring [Havelund, Rosu, ...]

“A lightweight verification technique, bridging the gap between model-checking and test”

Given a program $P$, a (linear) property $\Pi$

→ check if $P \models \Pi$ on the current execution run $\sigma$ of $P$?

- **synthesize** a monitor $M_\Pi$ (= decision procedure) for $\Pi$
- **instrument** the program to observe relevant events (w.r.t. $\Pi$)
- **at each execution step** of a run $\sigma$ of $P$ ⇒ a **verdict** is provided by $M_\Pi$
Example

- Program $P$, calling methods $e_1, e_2, \ldots e_n$
- Property $\Pi$: the 1st method called cannot be called again
Example

- Program $P$, calling methods $e_1, e_2, \ldots e_n$
- Property $\Pi$: the 1st method called cannot be called again
Example

- Program $P$, calling methods $e_1, e_2, \ldots e_n$
- Property $\Pi$: the 1st method called cannot be called again
Example

- Program $P$, calling methods $e_1, e_2, \ldots e_n$
- Property $\Pi$: the 1st method called cannot be called again
Example

- Program $P$, calling methods $e_1, e_2, \ldots e_n$
- Property $\Pi$: the 1st method called cannot be called again
Example

- Program $P$, calling methods $e_1, e_2, \ldots e_n$
- Property $\Pi$: the 1st method called cannot be called again
Example

- Program $P$, calling methods $e_1, e_2, \ldots e_n$
- Property $\Pi$: the 1st method called cannot be called again
Runtime enforcement [Schneider, Ligatti, ...]

“modify the current execution sequence when it goes wrong . . .”

Monitor $M_\Pi$ acts as a transducer (a filter) for the current run $\sigma$ by:

- **leaving it unchanged** if correct wrt to $\Pi$ (**transparency**)
- **changing it into a valid one** otherwise (**soundness**)

![Diagram of program and monitor interaction]
Main results

Some (clear) benefits of runtime verification:

- well-targeted: “what you verify is what you execute”
- scalable: independent w.r.t program size
- easy to use: do not rely on a program model
- highly dynamic:
  - set of expected properties can evolve
  - set of program components can evolve

Many (potential) application domains:

- security (OS level, application level),
- safety-critical systems,
- pervasive applications,
- “open” component-based systems (web services) etc.

Several existing tool implementations: JavaMOP, RuleR, Polymer, …
But still many open issues . . .

- how to instrument the code?
  quality of the probes, source vs binary instrumentation, etc.

- how to limit the execution time overhead?
  monitor integration, efficiency of the decision procedure

- what properties could be addressed?
  going beyond safety properties, non-functionnal requirements

- how to synthesize the corresponding monitors?

- how to cope with a limited observability/controlability over the program events?

- etc.
In this talk

Focus on RV expressivity and monitor synthesis:

1. delineate the space of properties that could be addressed by runtime monitoring & enforcement

2. revisit some previous results on this topic

3. propose an (automated) monitor generation technique

4. present a prototype tool

Based on a unified framework:

safety-progress property classification [Manna, Pnueli]
Outline

1. The Safety-Progress Classification of Properties [Manna, Pnueli]
2. Applicability of Runtime Validation Techniques
3. Runtime Verification and Enforcement Monitors
4. j-VETO: A Java Verification and Enforcement Toolbox
5. Conclusion and future works
Outline

1. The Safety-Progress Classification of Properties [Manna, Pnueli]
   - Overview
   - The automata view

2. Applicability of Runtime Validation Techniques

3. Runtime Verification and Enforcement Monitors

4. j-VETO: A Java Verification and Enforcement Toolbox

5. Conclusion and future works
Overview (1)

General classification of linear temporal properties
Overview (1)

General classification of linear temporal properties

Fine-grain definition of classes of properties

- basic classes: safety, guarantee, response, persistence
- compound classes: obligation, reactivity
General classification of linear temporal properties

Fine-grain definition of classes of properties

- basic classes: safety, guarantee, response, persistence
- compound classes: obligation, reactivity

The intuitive/informal idea
Examples of properties

Example

Operating system where an operation \( op \) is allowed only when an authorization \( auth \) has been granted before.

\[ \pi_1: \text{“an authorization grant \( auth \) should precede any occurrence of \( op \)"} \] is a safety property;

\[ \pi_2: \text{“the 1st authorization request \( req auth \) should be eventually followed by a grant (\( grant auth \)) or a deny (\( deny auth \))"} \] is a guarantee property;

\[ \pi_3: \text{“each occurrence of \( req auth \) should be first written in a log file and then answered either with a grant (\( grant auth \)) or a deny (\( deny auth \)) without any occurrence of \( op \) in the meantime"} \] is a response property;

\[ \pi_4: \text{“an incorrect use of operation \( op \) should imply that any future call to \( req auth \) will always result in a deny (\( auth \)) answer"} \] is a persistence property.
Examples of properties

**Example**

Operating system where an operation \( op \) is allowed only when an authorization \( auth \) has been granted before.

- \( \Pi_1 \): "an authorization grant \( grant\_auth \) should precede any occurrence of \( op \)"
  - is a **safety** property;
Examples of properties

Example

Operating system where an operation $op$ is allowed only when an authorization $auth$ has been granted before.

- $\Pi_1$: “an authorization grant $grant\_auth$ should precede any occurrence of $op$” is a safety property;
- $\Pi_2$: “the 1st authorization request $req\_auth$ should be eventually followed by a grant ($grant\_auth$) or a deny ($deny\_auth$)” is a guarantee property;
Examples of properties

Example

Operating system where an operation $op$ is allowed only when an authorization $auth$ has been granted before.

- $\Pi_1$: “an authorization grant $grant_auth$ should precede any occurrence of $op$” is a safety property;
- $\Pi_2$: “the 1st authorization request $req_auth$ should be eventually followed by a grant ($grant_auth$) or a deny ($deny_auth$)” is a guarantee property;
- $\Pi_3$: “each occurrence of $req_auth$ should be first written in a log file and then answered either with a $grant_auth$ or a $deny_auth$ without any occurrence of $op$ in the meantime” is a response property;
Examples of properties

Example

Operating system where an operation \( op \) is allowed only when an authorization \( auth \) has been granted before.

- \( \Pi_1 \): “an authorization grant \( grant\_auth \) should precede any occurrence of \( op \)” is a **safety** property;
- \( \Pi_2 \): “the 1st authorization request \( req\_auth \) should be eventually followed by a grant (\( grant\_auth \)) or a deny (\( deny\_auth \))” is a **guarantee** property;
- \( \Pi_3 \): “each occurrence of \( req\_auth \) should be first written in a log file and then answered either with a \( grant\_auth \) or a \( deny\_auth \) without any occurrence of \( op \) in the meantime” is a **response** property;
- \( \Pi_4 \): “an incorrect use of operation \( op \) should imply that any future call to \( req\_auth \) will always result in a \( deny\_auth \) answer” is a **persistence** property.
Overview (2)

Characterization according to several views

- logical, language-theoretic, topological
- **automata**: Streett automata
Overview (2)

Characterization according to several views

- logical, language-theoretic, topological
- **automata**: Streett automata

Customizing the SP classification for runtime verification

- Initially defined for *infinite execution sequences*
- Monitoring context
  - Processing incremental **finite** sequences
  - Verdict taken on finite sequences

Our properties: **r-properties**: \((\phi, \varphi)\)

- \(\phi\): the finitary property
- \(\varphi\): the infinitary property

There is be a “link” between \(\phi\) and \(\varphi\)
Outline

1. The Safety-Progress Classification of Properties [Manna, Pnueli]
   - Overview
   - The automata view

2. Applicability of Runtime Validation Techniques

3. Runtime Verification and Enforcement Monitors

4. j-VETO: A Java Verification and Enforcement Toolbox

5. Conclusion and future works
Finite state automata: Streett automata

Definition of a deterministic Streett \textit{m}-automaton

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

- \(Q\) is the set of automaton states \((q_{\text{init}} \in Q\) is the initial state),
- total function \(\rightarrow: Q \times \Sigma \rightarrow Q\) 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 \textit{recurrent} states,
  - and \(P_i \subseteq Q\) are the sets of \textit{persistent} states.
Finite state automata: Streett automata

Definition of a deterministic Streett \( m \)-automaton

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

- \( Q \) is the set of automaton states (\( q_{init} \in Q \) is the initial state),
- total function \( \rightarrow: Q \times \Sigma \rightarrow Q \) 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.

Example (Streett automaton)

“Every request is acknowledged, and never two successive requests”

\[ \Sigma = \{\text{req}, \text{ack}\} \]

\[ R = \{1\}, P = \emptyset \]
Acceptance criteria

Acceptance condition for **Infinite sequences**

For \( \sigma \in \Sigma^\omega \), \( A \) accepts \( \sigma \) if \( \forall i \in [1, m] \), \( \text{vinf}(\sigma, A) \cap R_i \neq \emptyset \) \( \lor \) \( \text{vinf}(\sigma, A) \subseteq P_i \)

where \( \text{vinf}(\sigma, A) \): set of states visited infinitely often
Acceptance criteria

Acceptance condition for **Infinite sequences**

For $\sigma \in \Sigma^\omega$, $A$ accepts $\sigma$ if $\forall i \in [1, m], vinf(\sigma, A) \cap R_i \neq \emptyset \lor vinf(\sigma, A) \subseteq P_i$

where $vinf(\sigma, A)$: set of states visited infinitely often

Acceptance condition for **Finite sequences**

For $\sigma \in \Sigma^*$ s.t. $|\sigma| = n$, $A$ accepts $\sigma$ if $\exists q_1, \ldots, q_n \in Q^A$, $run(\sigma, A) = q_1 \cdots q_n$ and $q_1 = q_{init}^A$ and $\forall i \in [1, m], q_n \in P_i \cup R_i$

(This semantics is similar to the semantics of RV-LTL [Bauer and al.])
Acceptance criteria

**Acceptance condition for Infinite sequences**

For $\sigma \in \Sigma^\omega$, $A$ accepts $\sigma$ if $\forall i \in [1, m]$, $vinf(\sigma, A) \cap R_i \neq \emptyset \lor vinf(\sigma, A) \subseteq P_i$

where $vinf(\sigma, A)$: set of states visited infinitely often

**Acceptance condition for Finite sequences**

For $\sigma \in \Sigma^*$ s.t. $|\sigma| = n$, $A$ accepts $\sigma$ if $\exists q_1, \ldots, q_n \in Q^A$,

$run(\sigma, A) = q_1 \cdots q_n \land q_1 = q_{init}^A$ and $\forall i \in [1, m], q_n \in P_i \cup R_i$

(This semantics is similar to the semantics of RV-LTL [Bauer and al.])

---

```
Σ = {req, ack}
R = \{1\}, P = \emptyset
```

"Every request is acknowledged, and never two successive requests"
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
  - no transition from $q \in \overline{P_i}$ to $q' \in P_i$,
  - no transition from $q \in R_i$ to $q' \in \overline{R_i}$,
- **$m$-reactivity**: unrestricted $m$-automaton.
Outline

1. The Safety-Progress Classification of Properties [Manna,Pnueli]

2. Applicability of Runtime Validation Techniques
   - Monitorable Properties
   - Enforceable properties

3. Runtime Verification and Enforcement Monitors

4. j-VETO: A Java Verification and Enforcement Toolbox

5. Conclusion and future works
Classical definition of monitorability [Pnueli,Zaks]

“Determine verdict of infinite sequences with (finite) observations”

\[ \rightarrow \text{evaluation depends on the satisfaction of the current sequence and its continuations} \]
Classical definition of monitorability [Pnueli, Zaks]

“Determine verdict of infinite sequences with (finite) observations”

→ evaluation depends on the satisfaction of the current sequence and its continuations

**Monitorable properties and ⊕/⊖-determinacy**

Considering $\sigma \in \Sigma^*$, a $r$-property $\Pi$ is said to be:

- **⊖-determined** by $\sigma$, if $\neg \Pi(\sigma \cdot \mu)$ for all completions $\mu \in \Sigma^\infty$
  → verdict ⊥

- **⊕-determined** by $\sigma$, if $\Pi(\sigma \cdot \mu)$ for all completions $\mu \in \Sigma^\infty$
  → verdict ⊤

- **σ-monitorable** if there exists $\mu \in \Sigma^*$ s.t. $\Pi$ is ⊕/⊖-determined by $\sigma \cdot \mu$
Classical definition of monitorability [Pnueli, Zaks]

“Determine verdict of infinite sequences with (finite) observations”

→ evaluation depends on the satisfaction of the current sequence and its continuations

Monitorable properties and $\oplus/\ominus$-determinacy

Considering $\sigma \in \Sigma^*$, a \textit{r}-property $\Pi$ is said to be:

- $\ominus$-determined by $\sigma$, if $\neg \Pi(\sigma \cdot \mu)$ for all completions $\mu \in \Sigma^\infty$
  → verdict $\bot$

- $\oplus$-determined by $\sigma$, if $\Pi(\sigma \cdot \mu)$ for all completions $\mu \in \Sigma^\infty$
  → verdict $\top$

- $\sigma$-monitorable if there exists $\mu \in \Sigma^*$ s.t. $\Pi$ is $\ominus/\oplus$-determined by $\sigma \cdot \mu$

Truth-domain $\mathbb{B}_3 = \{\bot, \top, ?\}$ determines the class of monitorable properties ($\text{MP}(\mathbb{B}_3)$)
Characterization of monitorable properties [RV’09]

Truth-domain of cardinality 3: \( \text{Obligation} \subset MP(\{?, \bot, \top\}) \)

- Safety and Guarantee properties are monitorable
- Union and intersection of monitorable properties is monitorable
Characterization of monitorable properties [RV’09]

Truth-domain of cardinality 3: \( \text{Obligation} \subseteq MP(\{?, \bot, \top\}) \)

- Safety and Guarantee properties are monitorable
- Union and intersection of monitorable properties is monitorable

Example (Monitorable)

\[
\begin{align*}
R &= \{1\}, P = \emptyset \\
\end{align*}
\]

Example (Not monitorable)

\[
\begin{align*}
R &= \{1\}, P = \emptyset \\
\end{align*}
\]
Characterization of monitorable properties \([\text{RV’09}]\)

Truth-domain of cardinality 3: \(\text{Obligation} \subset MP(\{?, \bot, \top\})\)
- Safety and Guarantee properties are monitorable
- Union and intersection of monitorable properties is monitorable

Example (Monitorable)
\[
\begin{array}{c}
1 \xrightarrow{\text{req}} 2 \xrightarrow{\text{req}} 3 \\
\end{array}
\]
\(R = \{1\}, P = \emptyset\)

Example (Not monitorable)
\[
\begin{array}{c}
1 \xrightarrow{\text{req}} 2 \\
\end{array}
\]
\(R = \{1\}, P = \emptyset\)

Non-monitorable properties
- (some) Response, Persistence, and Reactivity properties
- Impossible to detect \(\top\) or \(\bot\)
- \(\mapsto\) the output sequence of a monitor belongs to (?)*
Characterization of monitorable properties (cont.)

**Definition (Characterization of a Streett automaton states: $\mathbb{P}$)**

- $\text{Good}^A = \{ q \in Q^A \cap \bigcap_{i=1}^{m}(R_i \cup P_i) \mid \text{Reach}_A(q) \subseteq \bigcap_{i=1}^{m}(R_i \cup P_i) \}$
- $\text{Good}_p^A = \{ q \in Q^A \cap \bigcap_{i=1}^{m}(R_i \cup P_i) \mid \text{Reach}_A(q) \not\subseteq \bigcap_{i=1}^{m}(R_i \cup P_i) \}$
- $\text{Bad}_p^A = \{ q \in Q^A \cap \bigcup_{i=1}^{m}(R_i \cap \overline{P_i}) \mid \text{Reach}_A(q) \not\subseteq \bigcup_{i=1}^{m}(R_i \cap \overline{P_i}) \}$
- $\text{Bad}^A = \{ q \in Q^A \cap \bigcup_{i=1}^{m}(R_i \cap \overline{P_i}) \mid \text{Reach}_A(q) \subseteq \bigcup_{i=1}^{m}(R_i \cap \overline{P_i}) \}$
Characterization of monitorable properties (cont.)

Definition (Characterization of a Streett automaton states: $\mathbb{P}$)

- $\text{Good}^A = \{q \in Q^A \cap \bigcap_{i=1}^{m} (R_i \cup P_i) \mid \text{Reach}_A(q) \subseteq \bigcap_{i=1}^{m} (R_i \cup P_i)\}$
- $\text{Good}^p_A = \{q \in Q^A \cap \bigcap_{i=1}^{m} (R_i \cup P_i) \mid \text{Reach}_A(q) \not\subseteq \bigcap_{i=1}^{m} (R_i \cup P_i)\}$
- $\text{Bad}^p_A = \{q \in Q^A \cap \bigcup_{i=1}^{m} (\overline{R_i} \cap \overline{P_i}) \mid \text{Reach}_A(q) \not\subseteq \bigcup_{i=1}^{m} (\overline{R_i} \cap \overline{P_i})\}$
- $\text{Bad}^A = \{q \in Q^A \cap \bigcup_{i=1}^{m} (\overline{R_i} \cap \overline{P_i}) \mid \text{Reach}_A(q) \subseteq \bigcup_{i=1}^{m} (\overline{R_i} \cap \overline{P_i})\}$

Theorem

A r-property $\Pi$, recognized by $A_{\Pi}$, is $\mathbb{B}_3$-monitorable iff

$$\forall q \in \text{Reach}(q_{\text{init}}), \text{Reach}(q) \cap (\text{Bad}^{A_{\Pi}} \cup \text{Good}^{A_{\Pi}}) \neq \emptyset$$
Characterization of monitorable properties (cont.)

Definition (Characterization of a Streett automaton states: \( P \))

- \( \text{Good}^A = \{ q \in Q^A \cap \bigcap_{i=1}^{m} (R_i \cup P_i) \mid \text{Reach}_A(q) \subseteq \bigcap_{i=1}^{m} (R_i \cup P_i) \} \)
- \( \text{Good}_p^A = \{ q \in Q^A \cap \bigcap_{i=1}^{m} (R_i \cup P_i) \mid \text{Reach}_A(q) \nsubseteq \bigcap_{i=1}^{m} (R_i \cup P_i) \} \)
- \( \text{Bad}_p^A = \{ q \in Q^A \cup \bigcup_{i=1}^{m} (\overline{R_i} \cap \overline{P_i}) \mid \text{Reach}_A(q) \nsubseteq \bigcup_{i=1}^{m} (\overline{R_i} \cap \overline{P_i}) \} \)
- \( \text{Bad}^A = \{ q \in Q^A \cup \bigcup_{i=1}^{m} (\overline{R_i} \cap \overline{P_i}) \mid \text{Reach}_A(q) \subseteq \bigcup_{i=1}^{m} (\overline{R_i} \cap \overline{P_i}) \} \)

Theorem

A r-property \( \Pi \), recognized by \( A_\Pi \), is \( B_3 \)-monitorable iff

\( \forall q \in \text{Reach}(q_{\text{init}}), \text{Reach}(q) \cap (\text{Bad}^{A_n} \cup \text{Good}^{A_n}) \neq \emptyset \)
Characterization of monitorable properties (cont.)

Definition (Characterization of a Streett automaton states: \(\mathbb{P}\))

- \(\text{Good}^A = \{ q \in Q^A \cap \bigcap_{i=1}^{m} (R_i \cup P_i) \mid \text{Reach}_A(q) \subseteq \bigcap_{i=1}^{m} (R_i \cup P_i) \}\)
- \(\text{Good}_p^A = \{ q \in Q^A \cap \bigcap_{i=1}^{m} (R_i \cup P_i) \mid \text{Reach}_A(q) \not\subseteq \bigcap_{i=1}^{m} (R_i \cup P_i) \}\)
- \(\text{Bad}_p^A = \{ q \in Q^A \cap \bigcup_{i=1}^{m} (\overline{R_i} \cap \overline{P_i}) \mid \text{Reach}_A(q) \not\subseteq \bigcup_{i=1}^{m} (\overline{R_i} \cap \overline{P_i}) \}\)
- \(\text{Bad}^A = \{ q \in Q^A \cap \bigcup_{i=1}^{m} (\overline{R_i} \cap \overline{P_i}) \mid \text{Reach}_A(q) \subseteq \bigcup_{i=1}^{m} (\overline{R_i} \cap \overline{P_i}) \}\)

Theorem

A r-property \(\Pi\), recognized by \(A_{\Pi}\), is \(\mathbb{B}_3\)-monitorable iff

\[ \forall q \in \text{Reach}(q_{\text{init}}), \text{Reach}(q) \cap (\text{Bad}^{A_{\Pi}} \cup \text{Good}^{A_{\Pi}}) \neq \emptyset \]
Refinement of the notion of monitorability [RV’09]

Interest of the classical definition:

- 3-valued truth-domain
- detection of verdicts for infinitary properties from finite observations
Refinement of the notion of monitorability [RV’09]

Interest of the classical definition:
- 3-valued truth-domain
- detection of verdicts for infinitary properties from finite observations

Following [Bauer and al.] and the motivations of RV-LTL:
- Is $\oplus/\ominus$-determinacy always needed?
- Answer “What happens if the program execution stops here?”
- Distinguish prefixes which evaluated previously to “?”

Considering the truth-domain $\mathbb{B}_4 = \{\bot, \bot^p, \top^p, \top\}$:
- $\bot^p$: presumably false
- $\top^p$: presumably true
Alternative definition of monitorability

Definition (Evaluation of a sequence wrt. a property)

- $\llbracket \Pi \rrbracket (\sigma) = \top$ if $\Pi(\sigma) \land \forall \mu \in \Sigma^\infty \cdot \Pi(\sigma \cdot \mu)$
- $\llbracket \Pi \rrbracket (\sigma) = \top_p$ if $\Pi(\sigma) \land \exists \mu \in \Sigma^\infty \cdot \neg \Pi(\sigma \cdot \mu)$
- $\llbracket \Pi \rrbracket (\sigma) = \bot_p$ if $\neg \Pi(\sigma) \land \exists \mu \in \Sigma^\infty \cdot \Pi(\sigma \cdot \mu)$
- $\llbracket \Pi \rrbracket (\sigma) = \bot$ if $\neg \Pi(\sigma) \land \forall \mu \in \Sigma^\infty \cdot \neg \Pi(\sigma \cdot \mu)$
Alternative definition of monitorability

Definition (Evaluation of a sequence wrt. a property)

- $\llbracket \Pi \rrbracket (\sigma) = \top$ if $\Pi(\sigma) \land \forall \mu \in \Sigma^\infty \cdot \Pi(\sigma \cdot \mu)$
- $\llbracket \Pi \rrbracket (\sigma) = \top_p$ if $\Pi(\sigma) \land \exists \mu \in \Sigma^\infty \cdot \neg \Pi(\sigma \cdot \mu)$
- $\llbracket \Pi \rrbracket (\sigma) = \bot_p$ if $\neg \Pi(\sigma) \land \exists \mu \in \Sigma^\infty \cdot \Pi(\sigma \cdot \mu)$
- $\llbracket \Pi \rrbracket (\sigma) = \bot$ if $\neg \Pi(\sigma) \land \forall \mu \in \Sigma^\infty \cdot \neg \Pi(\sigma \cdot \mu)$

Several possible truth-domains:

- $\mathbb{B}_2^\bot = \{\bot, ?\}$
- $\mathbb{B}_2^\top = \{?, \top\}$
- $\mathbb{B}_3 = \{\bot, ?, \top\}$
- $\mathbb{B}_4 = \{\bot, \bot_p, \top_p, \top\}$
Alternative definition of monitorability

**Definition (Evaluation of a sequence wrt. a property)**

- \([\Pi](\sigma) = \top\) if \(\Pi(\sigma) \land \forall \mu \in \Sigma^\infty \cdot \Pi(\sigma \cdot \mu)\)
- \([\Pi](\sigma) = \top_p\) if \(\Pi(\sigma) \land \exists \mu \in \Sigma^\infty \cdot \neg \Pi(\sigma \cdot \mu)\)
- \([\Pi](\sigma) = \bot_p\) if \(\neg \Pi(\sigma) \land \exists \mu \in \Sigma^\infty \cdot \Pi(\sigma \cdot \mu)\)
- \([\Pi](\sigma) = \bot\) if \(\neg \Pi(\sigma) \land \forall \mu \in \Sigma^\infty \cdot \neg \Pi(\sigma \cdot \mu)\)

Several possible truth-domains:

- \(\mathbb{B}^\bot_2 = \{\bot, \, ?\}\)
- \(\mathbb{B}^\top_2 = \{?, \, \top\}\)
- \(\mathbb{B}_3 = \{\bot, ?, \, \top\}\)
- \(\mathbb{B}_4 = \{\bot, \bot_p, \, \top_p, \, \top\}\)

**Definition (Alternative monitorability)**

A \(r\)-property \(\Pi = (\phi, \varphi)\) is said to be monitorable with \(\mathbb{B}\) iff

\[\forall \sigma_{good} \in \phi, \forall \sigma_{bad} \in \overline{\phi}, \ [\Pi]_{\mathbb{B}}(\sigma_{good}) \neq [\Pi]_{\mathbb{B}}(\sigma_{bad})\]

We note \(\text{MP}^*(\mathbb{B})\), the set of monitorable properties with \(\mathbb{B}\).
New characterization of monitorable properties

Theorem (Multi-valued characterization of monitorability)

The sets of monitorable r-properties are:

(i) \( \text{MP}^*(\mathbb{B}^\perp_2) = \text{Safety} \)

(ii) \( \text{MP}^*(\mathbb{B}^\top_2) = \text{Guarantee} \)

(iii) \( \text{MP}^*(\mathbb{B}_3) = \text{Safety} \cup \text{Guarantee} \)

(iv) \( \text{MP}^*(\mathbb{B}_4) = \text{Reactivity} \)
New characterization of monitorable properties

Theorem (Multi-valued characterization of monitorability)

The sets of monitorable r-properties are:

(i) \( MP^*(B_2^\perp) = Safety \)
(ii) \( MP^*(B_2^\top) = Guarantee \)
(iii) \( MP^*(B_3) = Safety \cup Guarantee \)
(iv) \( MP^*(B_4) = Reactivity \)

Example

Safety property with \( B_2^\perp = \{\perp, ?\} \):

“always a”

\[
\sigma_{bad} = a^* \cdot \overline{a} \cdot \Sigma^* \\
\sigma_{good} = a^* \\
\sigma_{good} \rightarrow ?
\]

\[
\Sigma = \{a, \overline{a}\} \\
P = \{1\} \\
R = \emptyset
\]
Outline

1. The Safety-Progress Classification of Properties [Manna, Pnueli]

2. Applicability of Runtime Validation Techniques
   - Monitorable Properties
   - Enforceable properties

3. Runtime Verification and Enforcement Monitors

4. J-VETO: A Java Verification and Enforcement Toolbox

5. Conclusion and future works
Enforcement, soundness, and transparency

Soundness and transparency:

1. Output sequences are correct: **soundness**
2. Correct original execution sequences remain unchanged: **transparency**
Enforcement, soundness, and transparency

Soundness and transparency:

1. Output sequences are correct: **soundness**
2. Correct original execution sequences remain unchanged: **transparency**

Our choice: input sequence $\sigma$ should be modified in a minimal way:

- $\sigma \models \Pi \Rightarrow$ it should remain unchanged (up to an equivalence relation),
- $\sigma \not\models \Pi \Rightarrow$ its longest prefix satisfying $\Pi$ should be issued.

Expected for both finite and infinite execution sequences
Enforcement, soundness, and transparency

Soundness and transparency:

1. Output sequences are correct: **soundness**
2. Correct original execution sequences remain unchanged: **transparency**

Our choice: input sequence $\sigma$ should be modified *in a minimal way*:

- $\sigma \models \Pi \Rightarrow$ it should remain unchanged (up to an equivalence relation),
- $\sigma \not\models \Pi \Rightarrow$ its *longest prefix* satisfying $\Pi$ should be issued.

Expected for both finite and infinite execution sequences

**Consequence:** enforceability criterion for $\Pi = (\phi, \varphi)$

Each infinite incorrect sequence has a *longest* correct prefix,
$\leftrightarrow *i.e.*, a finite number of correct prefixes.
Response are exactly the enforceable properties

**Theorem** (*Response = EP*)

Consequence: safety, guarantee, and obligation $r$-properties are enforceable.
Response are exactly the enforceable properties

**Theorem** \((\text{Response} = \text{EP})\)

\[\begin{align*}
\text{R} & \quad \text{R} \\
\text{R} & \quad \text{R}
\end{align*}\]

Consequence: safety, guarantee, and obligation \(r\)-properties are enforceable.

**Example (Pure persistence property not enforceable)**

"it will be eventually true that \(a\) always occur"

\[\Pi = (\Sigma^* \cdot a^+, \Sigma^* \cdot a^\omega)\]

\[\Sigma = \{a, b\}, \text{vinf}(\sigma, A_\Pi) \subseteq P\] and \(P = \{1\}\)
Response are exactly the enforceable properties

**Theorem (Response = EP)**

Consequence: safety, guarantee, and obligation r-properties are enforceable.

**Example (Pure persistence property not enforceable)**

"it will be eventually true that a always occur"

\[ \Pi = (\Sigma^* \cdot a^+, \Sigma^* \cdot a^\omega) \]
\[ \Sigma = \{a, b\}, \ \text{vinf}(\sigma, A_{\Pi}) \subseteq P \text{ and } P = \{1\} \]

Infinite incorrect sequence can have an infinite number of correct prefixes:

- \( \sigma_{bad} = (ab)^\omega \)
- \( \neg \Pi(\sigma_{bad}) \) since \( \text{vinf}(\sigma_{bad}, A_{\Pi}) = \{1, 2\} \)
- but \( \forall i \in \mathbb{N}, \Pi((ab)^i \cdot a) \) since \( P = \{1\} \)
Response are exactly the enforceable properties

**Theorem** (*Response* = *EP*)

Consequence: safety, guarantee, and obligation *r*-properties are enforceable.

**Example** (Pure persistence property not enforceable)

"it will be eventually true that a always occur"

\[ \Pi = (\Sigma^* \cdot a^+, \Sigma^* \cdot a^\omega) \]

\[ \Sigma = \{a, b\} \]

\[ \text{vinf}(\sigma, A_\Pi) \subseteq P \text{ and } P = \{1\} \]

Infinite incorrect sequence can have an infinite number of correct prefixes:

- \( \sigma_{bad} = (ab)^\omega \)
- \( \neg \Pi(\sigma_{bad}) \) since \( \text{vinf}(\sigma_{bad}, A_\Pi) = \{1, 2\} \)
- \( \forall i \in \mathbb{N}, \Pi((ab)^i \cdot a) \) since \( P = \{1\} \)

What is the problem if we build a sound and transparent enforcement monitor?
Response are exactly the enforceable properties

**Theorem** \((\text{Response} = \text{EP})\)

Consequence: safety, guarantee, and obligation \(r\)-properties are enforceable.

**Example (Pure persistence property not enforceable)**

"it will be eventually true that \(a\) always occur"

\[
\Pi = (\Sigma^* \cdot a^+, \Sigma^* \cdot a^\omega)
\]

\[
\Sigma = \{a, b\}, \ vinf(\sigma, A_\Pi) \subseteq P \text{ and } P = \{1\}
\]

Infinite incorrect sequence can have an infinite number of correct prefixes:

- \(\sigma_{bad} = (ab)^\omega\)
- \(\neg\Pi(\sigma_{bad})\) since \(vinf(\sigma_{bad}, A_\Pi) = \{1, 2\}\)
- but \(\forall i \in \mathbb{N}, \Pi((ab)^i \cdot a)\) since \(P = \{1\}\)

**What is the problem** if we build a sound and transparent enforcement monitor?

\(IN = a\)

\(OUT = a\)
Response are exactly the enforceable properties

**Theorem (Response = EP)**

Consequence: safety, guarantee, and obligation $r$-properties are enforceable.

**Example (Pure persistence property not enforceable)**

"it will be eventually true that $a$ always occur"

\[
\Pi = (\Sigma^* \cdot a^+, \Sigma^* \cdot a^\omega) \\
\Sigma = \{a, b\}, \text{vinf}(\sigma, A_\Pi) \subseteq P \text{ and } P = \{1\}
\]

Infinite incorrect sequence can have an infinite number of correct prefixes:

- $\sigma_{bad} = (ab)^\omega$
- $\neg \Pi(\sigma_{bad})$ since $\text{vinf}(\sigma_{bad}, A_\Pi) = \{1, 2\}$
- but $\forall i \in \mathbb{N}, \Pi((ab)^i \cdot a)$ since $P = \{1\}$

**What is the problem** if we build a sound and transparent enforcement monitor?

$IN = a \cdot b$

$OUT = a$
Response are exactly the enforceable properties

**Theorem** \((\text{Response} = EP)\)

Consequence: safety, guarantee, and obligation \(r\)-properties are enforceable.

**Example (Pure persistence property not enforceable)**

"it will be eventually true that \(a\) always occur"

\[
\Pi = (\Sigma^* \cdot a^+, \Sigma^* \cdot a^\omega)
\]

\[
\Sigma = \{a, b\}, \ vinf(\sigma, A_\Pi) \subseteq P \text{ and } P = \{1\}
\]

Infinite incorrect sequence can have an infinite number of correct prefixes:

- \(\sigma_{bad} = (ab)^\omega\)
- \(\neg \Pi(\sigma_{bad})\) since \(\vinf(\sigma_{bad}, A_\Pi) = \{1, 2\}\)
- but \(\forall i \in \mathbb{N}, \Pi((ab)^i \cdot a)\) since \(P = \{1\}\)

**What is the problem** if we build a **sound and transparent** enforcement monitor?

**IN** = \(a \cdot b \cdot a\)

**OUT** = \(a \cdot b \cdot a\)
Response are exactly the enforceable properties

**Theorem** \((Response = EP)\)

Consequence: safety, guarantee, and obligation \(r\)-properties are enforceable.

**Example** (Pure persistence property not enforceable)

"it will be eventually true that \(a\) always occur"

\[
\Pi = (\Sigma^* \cdot a^\dagger, \Sigma^* \cdot a^\omega)
\]

\(\Sigma = \{a, b\}, \text{vinf}(\sigma, A_\Pi) \subseteq P\) and \(P = \{1\}\)

Infinite incorrect sequence can have an infinite number of correct prefixes:
- \(\sigma_{bad} = (ab)^\omega\)
- \(\neg \Pi(\sigma_{bad})\) since \(\text{vinf}(\sigma_{bad}, A_\Pi) = \{1, 2\}\)
- but \(\forall i \in \mathbb{N}, \Pi((ab)^i \cdot a)\) since \(P = \{1\}\)

**What is the problem** if we build a sound and transparent enforcement monitor?

\(IN = a \cdot b \cdot a \cdot b\)

\(OUT = a \cdot b \cdot a\)
Response are exactly the enforceable properties

**Theorem (Response = EP)**

Consequence: safety, guarantee, and obligation $r$-properties are enforceable.

**Example (Pure persistence property not enforceable)**

"it will be eventually true that $a$ always occur"

$\Pi = (\Sigma^* \cdot a^+, \Sigma^* \cdot a^\omega)$

$\Sigma = \{a, b\}$, $v_{inf}(\sigma, A_{\Pi}) \subseteq P$ and $P = \{1\}$

Infinite incorrect sequence can have an infinite number of correct prefixes:

- $\sigma_{bad} = (ab)^\omega$
- $\neg \Pi(\sigma_{bad})$ since $v_{inf}(\sigma_{bad}, A_{\Pi}) = \{1, 2\}$
- but $\forall i \in \mathbb{N}, \Pi((ab)^i \cdot a)$ since $P = \{1\}$

**What is the problem** if we build a sound and transparent enforcement monitor?

$IN = a \cdot b \cdot a \cdot b \cdot a$

$OUT = a \cdot b \cdot a \cdot b \cdot a \rightarrow$ an infinite incorrect sequence (e.g. $\sigma_{bad}$) could be produced :-(
Outline

1. The Safety-Progress Classification of Properties [Manna, Pnueli]

2. Applicability of Runtime Validation Techniques

3. Runtime Verification and Enforcement Monitors
   - Synthesis
   - Composition

4. j-VETO: A Java Verification and Enforcement Toolbox

5. Conclusion and future works
Runtime verification and enforcement Monitors

Definition (Monitor)

\( \mathcal{A} \) is a 5-tuple \((Q^\mathcal{A}, q_{\text{init}}^\mathcal{A}, \rightarrow^\mathcal{A}, X^\mathcal{A}, \Gamma^\mathcal{A})\) (defined relatively to \( \Sigma \))

- a (classical) FSM
- The set of values \( X^\mathcal{A} \) depends on the purpose of the monitor (verification or enforcement)
- \( \Gamma^\mathcal{A} : Q^\mathcal{A} \rightarrow X^\mathcal{A} \), output function, producing values in \( X^\mathcal{A} \) from states.
Runtime verification and enforcement Monitors

Definition (Monitor)

A is a 5-tuple \( (Q^A, q_{init}^A, \rightarrow_A, X^A, \Gamma^A) \) (defined relatively to \( \Sigma \))

- a (classical) FSM
- The set of values \( X^A \) depends on the purpose of the monitor (verification or enforcement)
- \( \Gamma^A : Q^A \rightarrow X^A \), output function, producing values in \( X^A \) from states.

Runtime Verification and Enforcement Monitors

Using \( \mathbb{P} \) to define the output function \( \Gamma^A \) (depends on the current state)

- For runtime verification: \( X^A = \mathbb{B}_4 \)
- For runtime enforcement:
  - \( X^A = \{ \text{halt, store, dump, off} \} \)
  - using an internal memory: a FIFO queue
Enforcement operations:

- inputs: an event and a memory content (i.e., a sequence of events)
- outputs: a new memory content and an output sequence

**Definition (Enforcement operations \( Ops \))**

\[
Ops \subseteq 2((\Sigma \cup \{\epsilon\}) \times \Sigma^*) \rightarrow (\Sigma^* \times \Sigma^*).
\]

\( Ops = \{\text{halt}, \text{store}, \text{dump}, \text{off}\} \) defined as follows: \( \forall a \in \Sigma \cup \{\epsilon\}, \forall m \cdot \Sigma^* \)

- \( \text{halt}(a, m) = (\epsilon, m) \)
- \( \text{store}(a, m) = (\epsilon, m.a) \)
- \( \text{dump}(a, m) = (m.a, \epsilon) \)
- \( \text{off}(a, m) = (m.a, \epsilon) \)
Synthesis from Streett automata [ICISS’08, SAC’09]

\[ A_\Pi = (Q^{A_\Pi}, q_{init}^{A_\Pi}, \Sigma, \rightarrow^{A_\Pi}, \{(R_1, P_1), \ldots, (R_m, P_m)\}) \] recognizing \( \Pi \)

Correspondence between \( \mathbb{P} \) and \( \mathbb{B}_4 \)

- run ends in \( Good^{A_\Pi} \) \( \iff \) \( [\Pi]_{\mathbb{B}_4}(\sigma) = \top \), run ends in \( Bad^{A_\Pi}_p \) \( \iff \) \( [\Pi]_{\mathbb{B}_4}(\sigma) = \bot_p \),
- run ends in \( Good^{A_\Pi}_p \) \( \iff \) \( [\Pi]_{\mathbb{B}_4}(\sigma) = \top_p \), run ends in \( Bad^{A_\Pi} \) \( \iff \) \( [\Pi]_{\mathbb{B}_4}(\sigma) = \bot \).
Synthesis from Streett automata [ICISS’08, SAC’09]

\[ A_\Pi = (Q^{A_\Pi}, q_{\text{init}}^{A_\Pi}, \Sigma, \longrightarrow^{A_\Pi}, \{(R_1, P_1), \ldots, (R_m, P_m)\}) \] recognizing \( \Pi \)

Correspondence between \( \mathbb{P} \) and \( \mathbb{B}_4 \)

- Run ends in \( \text{Good}^{A_\Pi} \) \( \iff \) \( \bigwedge_{B}^{\mathbb{B}_4}(\sigma) = \top \), run ends in \( \text{Bad}^P_{A_\Pi} \) \( \iff \) \( \bigwedge_{B}^{\mathbb{B}_4}(\sigma) = \bot_p \).
- Run ends in \( \text{Good}^P_{A_\Pi} \) \( \iff \) \( \bigwedge_{B}^{\mathbb{B}_4}(\sigma) = \top_p \), run ends in \( \text{Bad}^{A_\Pi} \) \( \iff \) \( \bigwedge_{B}^{\mathbb{B}_4}(\sigma) = \bot \).

Transformation \( \text{Streett2VM}(A_\Pi) = (Q^{A_\Pi}, q_{\text{init}}^{A_\Pi}, \longrightarrow^{A_\Pi}, \mathbb{B}_4, \Gamma) \)

with \( \Gamma : Q^{A_\Pi} \rightarrow \mathbb{B}_4 \) s.t. \( \forall q \in Q^{A_\Pi}, \)

- \( q \in \text{Good}^{A_\Pi} \Rightarrow \Gamma(q) = \top \)
- \( q \in \text{Bad}^P_{A_\Pi} \Rightarrow \Gamma(q) = \bot_p \)

Transformation \( \text{Streett2EM}(A_\Pi) = (Q^{A_\Pi}, q_{\text{init}}^{A_\Pi}, \longrightarrow^{A_\Pi}, \text{Ops}, \Gamma) \)

with \( \Gamma : Q^{A_\Pi} \rightarrow \text{Ops} \) s.t. \( \forall q \in Q^{A_\Pi}, \)

- \( q \in \text{Good}^{A_\Pi} \Rightarrow \Gamma(q) = \text{off} \)
- \( q \in \text{Good}^P_{A_\Pi} \Rightarrow \Gamma(q) = \text{dump} \)
- \( q \in \text{Bad}^{A_\Pi} \Rightarrow \Gamma(q) = \text{store} \)
- \( q \in \text{Bad}^P_{A_\Pi} \Rightarrow \Gamma(q) = \text{halt} \)
About composition (overview)

The idea: order $\mathbb{B}_4 = \{\bot, \bot_p, \top_p, \top\}$ and $Ops = \{halt, store, dump, off\}$

- $\mathbb{B}_4 : \bot < \bot_p < \top_p < \top$
- $Ops : halt < store < dump < off$
About composition (overview)

The idea: order $\mathbb{B}_4 = \{\bot, \bot_p, \top_p, \top\}$ and $Ops = \{halt, store, dump, off\}$

- $\mathbb{B}_4 : \bot < \bot_p < \top_p < \top$
- $Ops : halt < store < dump < off$

How to compose?

- Build the underlying cartesian product
- To determine the output (verdict or enforcement operations):
  - Union is “taking the max”
  - Intersection is “taking the min”
About composition (overview)

The idea: order $\mathbb{B}_4 = \{\bot, \bot_p, \top_p, \top\}$ and $\text{Ops} = \{\text{halt, store, dump, off}\}$

- $\mathbb{B}_4 : \bot < \bot_p < \top_p < \top$
- $\text{Ops} : \text{halt} < \text{store} < \text{dump} < \text{off}$

How to compose?

- Build the underlying cartesian product
- To determine the output (verdict or enforcement operations):
  - Union is “taking the max”
  - Intersection is “taking the min”

Example (Union of Enforcement Monitors: “always a or eventually b”)

```
always a
\[ a \]
\[ 1 \]
\[ \bar{a} \]
\[ \Sigma_1 \]
1: presumably good
2: bad

eventually b
\[ \bar{b} \]
\[ 1 \]
\[ b \]
\[ \Sigma_2 \]
1: presumably bad
2: good
```

“always a or eventually b”
About composition (overview)

The idea: order $\mathbb{B}_4 = \{\bot, \bot_p, \top_p, \top\}$ and $\text{Ops} = \{\text{halt, store, dump, off}\}$

- $\mathbb{B}_4 : \bot < \bot_p < \top_p < \top$
- $\text{Ops} : \text{halt} < \text{store} < \text{dump} < \text{off}$

How to compose?
- Build the underlying cartesian product
- To determine the output (verdict or enforcement operations):
  - Union is “taking the max”
  - Intersection is “taking the min”

Example (Union of Enforcement Monitors: “always $a$ or eventually $b$”)

```
Σ1   |   dump   |   Σ2   |   store   
  ▼    ▼       ▼     ▼
    a    b        b
Σ1   |   halt   |   Σ2   |   off     
  ▼    ▼       ▼     ▼
    a    b        b
```

“always $a$ or eventually $b$”
About composition (overview)

The idea: order $\mathbb{B}_4 = \{\bot, \bot_p, T_p, T\}$ and $Ops = \{halt, store, dump, off\}$

- $\mathbb{B}_4: \bot < \bot_p < T_p < T$
- $Ops: \text{halt} < \text{store} < \text{dump} < \text{off}$

How to compose?
- Build the underlying cartesian product
- To determine the output (verdict or enforcement operations):
  - Union is “taking the max”
  - Intersection is “taking the min”

Example (Union of Enforcement Monitors: “always a or eventually b”)

```
Σ1
   a
  —> dump
    b
  → halt
Σ2
   b
  —> store
    a
  → off
Σ_{12} = \{\overline{a}b; ab, \overline{a}b, ab\}
```

(L. Mounier - Verimag/UJF)
About composition (overview)

The idea: order $\mathbb{B}_4 = \{\bot, \bot_p, \top_p, \top\}$ and $\text{Ops} = \{\text{halt}, \text{store}, \text{dump}, \text{off}\}$

- $\mathbb{B}_4 : \bot < \bot_p < \top_p < \top$
- $\text{Ops} : \text{halt} < \text{store} < \text{dump} < \text{off}$

How to compose?

- Build the underlying cartesian product
- To determine the output (verdict or enforcement operations):
  - Union is “taking the max”
  - Intersection is “taking the min”

Example (Union of Enforcement Monitors: “always $a$ or eventually $b$”)

```
always $a$

```

![Diagram](always_a.png)

```
eventually $b$

```

![Diagram](eventually_b.png)

```
always $a$ or eventually $b$

```

![Diagram](always_a_or_eventually_b.png)

$\Sigma_{12} = \{\overline{a}b; ab, \overline{a}b, ab\}$
Outline

1. The Safety-Progress Classification of Properties [Manna, Pnueli]
2. Applicability of Runtime Validation Techniques
3. Runtime Verification and Enforcement Monitors
4. j-VETO: A Java Verification and Enforcement Tool Box
   - Overview and property design
   - Monitor Synthesis
   - Monitor Integration
5. Conclusion and future works
About j-VETO

**Toolbox** for this approach: 2 main stages and additional features

- **Monitor synthesis**: Property → verification or enforcement Monitor  
  ↪ XSLT transformation (XML to XML)

- **Monitor integration**:  
  ↪ using program-transformation frameworks (here AOP)

- Additional features: monitor composition (boolean operations), graphic representation of monitors, ...
About j-VETO

**Toolbox** for this approach: 2 main stages and additional features

- **Monitor synthesis**: Property $\rightarrow$ verification or enforcement Monitor $\leftarrow$ XSLT transformation (XML to XML)
- **Monitor integration**: $\leftarrow$ using program-transformation frameworks (here AOP)
- Additional features: monitor composition (boolean operations), graphic representation of monitors, ...  

**Experimental evaluation**

- DaCapo benchmark with properties about the use of Java data structures e.g., “Do not update a Collection if an iterator has been created from it”
- Comparison with Java-MOP
Overview

Pattern

Regular Property

$X \in \{A, E, R, P\}$

$\psi(\Sigma_a)$

Monitor Synthesis

Monitor for

$\Pi = (X_f(\psi_a), X(\psi_a))$

Monitor Integration

j-VETO

Mapping

Integration Program

Directives

Program

$P'_{\Sigma_c}$

Automaton–like Representations

(Java Objects, XML Automata)

Graphic Representations

(png, jpg, . . . )

Automaton Model and Utilities
Property design

Pattern
\[ X \in \{A, E, R, P\} \]

Regular Property
\[ \psi(\Sigma_a) \]

Mapping
\[ \Sigma_a \leftrightarrow \Sigma_c \]

Informal Property

Program \( P_{\Sigma_c} \)

Property Design

(L. Mounier - Verimag/UJF)

Runtime Verif. of SP properties

24/11/09 - Prague 37 / 43
Monitor Synthesis

Pattern
\[ X \in \{ A, E, R, P \} \]

Regular Property \( \psi(\Sigma_a) \)
DFA, \( A_\psi \)

\[ r\text{-property} \quad \Pi = (X_f(\psi), X(\psi)) \]
Streett Automaton \( A_\Pi \)

DFA2Streett

Streett2Monitor

Verification Monitor \( A_{?\Pi} \)
Enforcement Monitor \( A_{!\Pi} \)

(L. Mounier - Verimag/UJF)
Monitor Integration

\[
\Pi = (X_f(\psi), X(\psi))
\]

\[
\Sigma_a \leftrightarrow \Sigma_c
\]
Outline

1. The Safety-Progress Classification of Properties [Manna, Pnueli]
2. Applicability of Runtime Validation Techniques
3. Runtime Verification and Enforcement Monitors
4. j-VETO: A Java Verification and Enforcement TOOLbox
5. Conclusion and future works
Study & implementations of runtime techniques using a general framework
Conclusion

Study & implementations of runtime techniques using a general framework

Applicability of Runtime Techniques

- Characterization of monitorable properties
  - parameterized according to a truth-domain
  - classical definition ($MP(B)$)
  - alternative definition ($MP^*(B)$)
- Characterization of enforceable properties (EP)
Conclusion

Study & implementations of runtime techniques using a general framework

Applicability of Runtime Techniques

- Characterization of monitorable properties
  - parameterized according to a truth-domain
  - classical definition ($MP(B)$)
  - alternative definition ($MP^*(B)$)
- Characterization of enforceable properties (EP)

Synthesis procedures to generate runtime and enforcement monitors

Prototype tools implementing the aforementioned features
(some) Future Works

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

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

→ How the sets of monitorable and enforceable properties are impacted?

**Application to specific domains:**

- web service orchestration/choreography
- continuous lifetime verification
- non-functionnal properties (e.g., power management)
- etc.
Some publications


- Falcone, Y., Mounier, L., Fernandez, J-C., Richier, J-L.: **Runtime Enforcement Monitors: composition, synthesis, and enforcement abilities** under review at FMSD

Thank you for your attention