Runtime Verification of Safety-Progress Properties

9^{th} International Workshop on Runtime Verification

Ylies Falcone Jean-Claude Fernandez Laurent Mounier

Verimag, Grenoble Universities

June 26 2009, Grenoble, France

"Classical" runtime validation method: monitoring

Runtime Verification [Havelund,Rosu]

- A lightweight verification technique "bridging the gap" between **testing** and **verification**
- **Checking** whether a run of the system under scrutiny satisfies a given correctness property

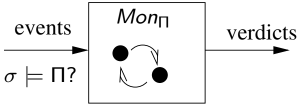
"Classical" runtime validation method: monitoring

Runtime Verification [Havelund,Rosu]

- A lightweight verification technique "bridging the gap" between **testing** and **verification**
- **Checking** whether a run of the system under scrutiny satisfies a given correctness property

Instrument the underlying program to observe relevant events

A monitor acts as an **oracle** for the property (validation/violation)



Enforcement Monitoring: extension of monitoring

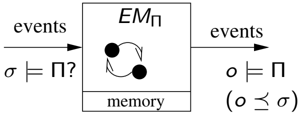
Gaining more confidence?

- 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?

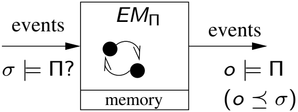


Underlying mechanism: enforcement monitor
$\hookrightarrow$ modifies the current execution sequence

Enforcement Monitoring: extension of monitoring

Gaining more confidence?

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



Underlying mechanism: enforcement monitor
$\hookrightarrow$ modifies the current execution sequence

Informal principle [Schneider, Ligatti and al.]

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

Previous Mechanisms and enforceable properties

- Schneider and al.: security-automata and safety properties
- Ligatti and al.: edit-automata and renewal properties

Our proposal

Which properties those techniques can address ?

Characterization of some "classes" of properties

Based on the Safety-Progress classification [Manna,Pnueli]
$\hookrightarrow$ Unified framework with several views of properties (logical, automata,. . .)

Revisiting and extending existing results in this uniform framework

(Simple) Synthesis techniques of monitors

Outline

1. The Safety-Progress Classification of Properties [Manna,Pnueli]
2. Monitorable Properties
3. Enforceable properties
4. Synthesis of Monitors from Streett Automata
5. Conclusion and future works

Outline

1. The Safety-Progress Classification of Properties [Manna,Pnueli]
   - Overview
   - The automata view
2. Monitorable Properties
3. Enforceable properties
4. Synthesis of Monitors from Streett Automata
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

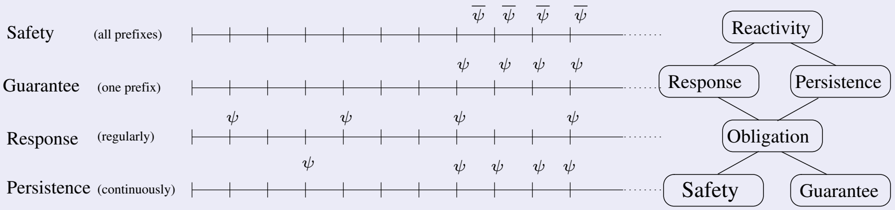
Overview (1)

General classification of linear temporal properties

Fine-grain definition of classes of properties

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

The intuitive/informal idea



Overview (2)

Characterization according to several views

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

Overview (2)

Characterization according to several views

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

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: (ϕ, φ)

- ϕ: the finitary property
- φ: the infinitary property

There should be a "link" between ϕ and φ

Outline

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

2 Monitorable Properties

3 Enforceable properties

4 Synthesis of Monitors from Streett Automata

5 Conclusion and future works

Finite state automata: Streett automata

Definition of a deterministic Streett m-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.

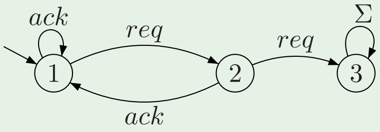
Finite state automata: Streett automata

Definition of a deterministic Streett m-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.

Example (Streett automata)



$R = \{1\}$

"Every request is acknowledged, and never two successive requests"

Acceptance criteria

Acceptance condition for **Infinite sequences**

For $\sigma \in \Sigma^\omega$, $\mathcal{A}$ accepts σ if
$\forall i \in [1, m], vinf(\sigma, \mathcal{A}) \cap R_i \neq \emptyset \vee vinf(\sigma, \mathcal{A}) \subseteq P_i$
where $vinf(\sigma, \mathcal{A})$: set of states visited infinitely often

Acceptance criteria

Acceptance condition for **Infinite sequences**

For $\sigma \in \Sigma^\omega$, $\mathcal{A}$ accepts σ if
$\forall i \in [1, m], vinf(\sigma, \mathcal{A}) \cap R_i \neq \emptyset \vee vinf(\sigma, \mathcal{A}) \subseteq P_i$
where $vinf(\sigma, \mathcal{A})$: set of states visited infinitely often

Acceptance condition for **Finite sequences**

For $\sigma \in \Sigma^*$ s.t. $|\sigma| = n$, $\mathcal{A}$ accepts σ if $\exists q_0, \ldots, q_n \in Q^{\mathcal{A}}$,
$run(\sigma, \mathcal{A}) = q_0 \cdots q_n \wedge q_0 = {q_{\mathrm{init}}}^{\mathcal{A}}$ 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

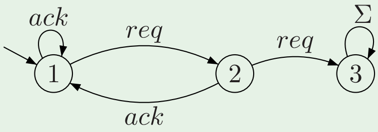
Acceptance condition for **Infinite sequences**

For $\sigma \in \Sigma^\omega$, $\mathcal{A}$ accepts σ if
$\forall i \in [1, m], vinf(\sigma, \mathcal{A}) \cap R_i \neq \emptyset \vee vinf(\sigma, \mathcal{A}) \subseteq P_i$
where $vinf(\sigma, \mathcal{A})$: set of states visited infinitely often

Acceptance condition for **Finite sequences**

For $\sigma \in \Sigma^*$ s.t. $|\sigma| = n$, $\mathcal{A}$ accepts σ if $\exists q_0, \ldots, q_n \in Q^{\mathcal{A}}$,
$run(\sigma, \mathcal{A}) = q_0 \cdots q_n \wedge q_0 = q_{\text{init}}{}^{\mathcal{A}}$ 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.])



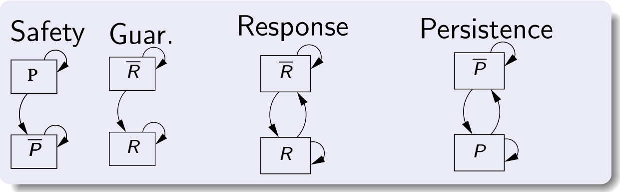
$R = \{1\}$

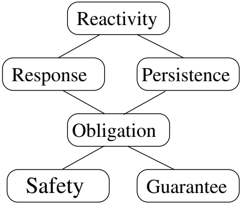
"Every request is acknowledged, and never two successive requests"

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 Monitorable Properties
- Classical definition of Monitorability
- Refinement of the notion of monitorability

3 Enforceable properties

4 Synthesis of Monitors from Streett Automata

5 Conclusion and future works

Classical definition of monitorability [Pnueli,Zaks]

"Determine verdict of infinite sequences with (finite) observations"

$\hookrightarrow$ 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"

$\hookrightarrow$ evaluation depends on the satisfaction of the current sequence and its continuations

Properties $\oplus$/$\ominus$-determined

Considering $\sigma \in \Sigma^*$, a r-property $\Pi = (\phi, \varphi)$ is said to be:

- $\ominus$-determined by σ, if $\neg\Pi(\sigma \cdot \mu)$ for all completions $\mu \in \Sigma^\infty$
  $\hookrightarrow$ verdict $\perp$
- $\oplus$-determined by σ, if $\Pi(\sigma \cdot \mu)$ for all completions $\mu \in \Sigma^\infty$
  $\hookrightarrow$ verdict $\top$

Classical definition of monitorability [Pnueli,Zaks]

"Determine verdict of infinite sequences with (finite) observations"

$\hookrightarrow$ evaluation depends on the satisfaction of the current sequence and its continuations

Properties $\oplus/\ominus$-determined

Considering $\sigma \in \Sigma^*$, a r-property $\Pi = (\phi, \varphi)$ is said to be:

- $\ominus$-determined by σ, if $\neg\Pi(\sigma \cdot \mu)$ for all completions $\mu \in \Sigma^\infty$
  $\hookrightarrow$ verdict $\perp$
- $\oplus$-determined by σ, if $\Pi(\sigma \cdot \mu)$ for all completions $\mu \in \Sigma^\infty$
  $\hookrightarrow$ verdict $\top$

Truth $\mathbb{B}$ domain determines the class of monitorable properties
$\hookrightarrow$ $MP(\mathbb{B})$: the set of monitorable properties with $\mathbb{B}$

Characterization of monitorability

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

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

(Exact characterization on a Streett automaton)

Characterization of monitorability

Truth-domain of cardinality 3: *Obligation* $\subset MP(\{?, \perp, \top\})$

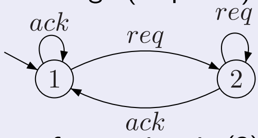
- Safety properties are monitorable
- Guarantee properties are monitorable
- Union and intersection of monitorable properties is monitorable

(Exact characterization on a Streett automaton)

Non-monitorable properties

- (some) Response, Persistence, and Reactivity properties
- Impossible to detect $\top$ or $\perp$
- Example: request/acknowledge (response) properties [Bauer and al.]

in LTL: $\Box(r \Rightarrow \Diamond a)$



$\hookrightarrow$ the output sequence of a monitor is $(?)^*$

Refinement of the notion of monitorability

Following [Bauer and al.] and the motivations of RV-LTL:

- Trying to 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

Refinement of the notion of monitorability

Following [Bauer and al.] and the motivations of RV-LTL:

- Trying to 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

Definition (Refinement of monitorability)

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

Refinement of the notion of monitorability

Following [Bauer and al.] and the motivations of RV-LTL:

- Trying to 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

Definition (Refinement of monitorability)

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

Theorem ($MP(\mathbb{B}_4) = Reactivity(\Sigma)$)

Outline

1. The Safety-Progress Classification of Properties [Manna,Pnueli]
2. Monitorable Properties
3. Enforceable properties
4. Synthesis of Monitors from Streett Automata
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

Consequence: input sequence σ 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 Π 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

Consequence: input sequence σ 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 Π 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,
$\hookrightarrow$ *i.e.* a finite number of correct prefixes.

Enforcement Criterion and response properties

Definition (Enforcement criterion)

A r-property $\Pi = (\phi, \varphi)$ is said to be enforceable iff

$$\forall \sigma \in \Sigma^\omega, \left(\neg\varphi(\sigma) \Rightarrow (\exists \sigma' \in \Sigma^*, \sigma' \prec \sigma, \forall \sigma'' \in \Sigma^* \cdot \sigma' \prec \sigma'' \Rightarrow \neg\phi(\sigma''))\right)$$

Enforcement Criterion and response properties

Definition (Enforcement criterion)

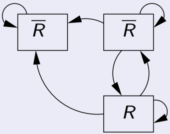
A r-property $\Pi = (\phi, \varphi)$ is said to be enforceable iff

$$\forall \sigma \in \Sigma^\omega, (\neg\varphi(\sigma) \Rightarrow (\exists \sigma' \in \Sigma^*, \sigma' \prec \sigma, \forall \sigma'' \in \Sigma^* \cdot \sigma' \prec \sigma'' \Rightarrow \neg\phi(\sigma'')))$$

Theorem (Response are enforceable: $Response(\Sigma) \subseteq EP$)

Sketch of proof

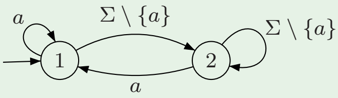
- Consider $\sigma_{bad} \in \Sigma^\omega$
- $\neg\varphi(\sigma) \Rightarrow vinf(\sigma_{bad}) \cap R = \emptyset$
- run "stays" in $\overline{R}$ from a certain point



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

Pure persistence properties are not enforceable

Example

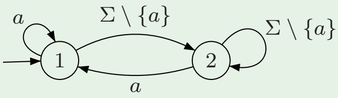


$\Pi = (\Sigma^* \cdot a^+, \Sigma^* \cdot a^\omega)$: "it will be eventually true that a always occur"
$vinf(\sigma, \mathcal{A}_\Pi) \subseteq P$ and $P = \{1\}$

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

Pure persistence properties are not enforceable

Example



$\Pi = (\Sigma^* \cdot a^+, \Sigma^* \cdot a^\omega)$: "it will be eventually true that a always occur"

$vinf(\sigma, \mathcal{A}_\Pi) \subseteq P$ and $P = \{1\}$

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

In other words:

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

Outline

1. The Safety-Progress Classification of Properties [Manna,Pnueli]
2. Monitorable Properties
3. Enforceable properties
4. Synthesis of Monitors from Streett Automata
   - Preliminaries on Streett Automata
   - Runtime Verification and Enforcement Monitors
5. Conclusion and future works

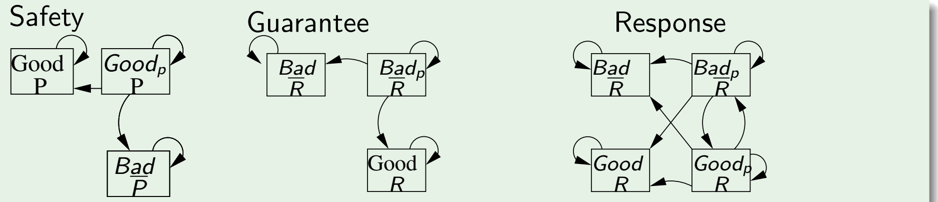
Streett m-automaton $\mathcal{A} = (Q^{\mathcal{A}}, {q_{\mathrm{init}}}^{\mathcal{A}}, \longrightarrow_{\mathcal{A}}, \{(R_1, P_1), \ldots, (R_m, P_m)\})$.

General Idea: (syntactic) characterization of the states according to the verdict to be produced

Definition ($\mathbb{P}^{\mathcal{A}}$ (good, presumably good, presumably bad, bad states))

- $Good^{\mathcal{A}} = \{q \in Q^{\mathcal{A}} \cap \bigcap_{i=1}^{m}(R_i \cup P_i) \mid Reach_{\mathcal{A}}(q) \subseteq \bigcap_{i=1}^{m}(R_i \cup P_i)\}$
- $Good_p^{\mathcal{A}} = \{q \in Q^{\mathcal{A}} \cap \bigcap_{i=1}^{m}(R_i \cup P_i) \mid Reach_{\mathcal{A}}(q) \not\subseteq \bigcap_{i=1}^{m}(R_i \cup P_i)\}$
- $Bad_p^{\mathcal{A}} = \{q \in Q^{\mathcal{A}} \cap \bigcup_{i=1}^{m}(\overline{R_i} \cap \overline{P_i}) \mid Reach_{\mathcal{A}}(q) \not\subseteq \bigcup_{i=1}^{m}(\overline{R_i} \cap \overline{P_i})\}$
- $Bad^{\mathcal{A}} = \{q \in Q^{\mathcal{A}} \cap \bigcup_{i=1}^{m}(\overline{R_i} \cap \overline{P_i}) \mid Reach_{\mathcal{A}}(q) \subseteq \bigcup_{i=1}^{m}(\overline{R_i} \cap \overline{P_i})\}$

Note that $Q^{\mathcal{A}} = Good^{\mathcal{A}} \cup Good_p^{\mathcal{A}} \cup Bad_p^{\mathcal{A}} \cup Bad^{\mathcal{A}}$.



Runtime verification and enforcement Monitors

Definition (Monitor)

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

- 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)

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

- 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

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

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

Outline

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

2 Monitorable Properties

3 Enforceable properties

4 Synthesis of Monitors from Streett Automata

5 Conclusion and future works

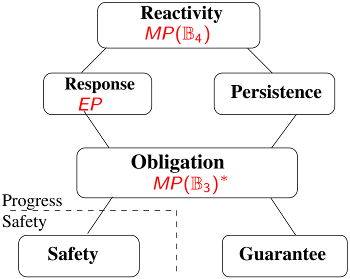
Conclusion

Monitorability and enforceability at runtime using a general framework

Conclusion

Monitorability and enforceability at runtime using a general framework

- Characterization of monitorable and enforceable properties in a unified way.
- Encompassing previous definitions of monitorability
(previous definitions can be derived by reducing the truth-domain $\mathbb{B}_4$)



Synthesis procedures to generate runtime and enforcement monitors

Future works

The testing perspective:

- Differences:
    - A monitor (passively) observes the execution of the program
    - notion of *controlable* event is introduced
- Characterize the set of testable properties in a similar fashion
  $\hookrightarrow$ deal with a reduced observability on the system under scrutiny

Future works

The testing perspective:

- Differences:
  - A monitor (passively) observes the execution of the program
  - notion of *controlable* event is introduced
- Characterize the set of testable properties in a similar fashion
  $\hookrightarrow$ deal with a reduced observability on the system under scrutiny

Space of properties for which others RV-like techniques can be applied (*e.g.* runtime reflection)

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

- data dependency between events
- Memory limitation for the EM
- Influence on the enforcement ability: how the set of enforceable properties is impacted?