Runtime Verification of Safety-Progress Properties

Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier
Firstname.Lastname@imag.fr

Verimag, Université Grenoble I

Abstract. The underlying property, its definition and representation play a major role when monitoring a system. Having a suitable and convenient framework to express properties is thus a concern for runtime analysis. It is desirable to delineate in this framework the spaces of properties for which runtime verification approaches can be applied to.
This paper presents a unified view of runtime verification and enforcement of properties in the safety-progress classification. Firstly, we characterize the set of properties which can be verified (monitorable properties) and enforced (enforceable properties) at runtime. We propose in particular an alternative definition of "property monitoring" to the one classically used in this context. Secondly, for the delineated spaces of properties, we obtain specialized verification and enforcement monitors.

1 Introduction

Runtime-verification [1–5] is an effective technique to ensure at execution time that a system meets a desirable behavior. It can be used in numerous application domains, and more particularly when integrating together untrusted software components. In runtime verification, a run of the system under scrutiny is analyzed incrementally using a decision procedure: a monitor. This monitor may be generated from a user-provided high level specification (*e.g.* a temporal property, an automaton). The primary goal of this monitor is to detect violation or validation wrt. the given specification. It is a state machine (with an output function) processing an execution sequence (step by step) of the monitored program, and producing a sequence of verdicts (truth values taken from a truth-domain) indicating specification fulfilment or violation. The major part of research endeavor was done on the monitoring of safety properties, as seen for example in [6, 7]. However, the authors of [8] show that safety properties are not the only monitorable properties. Recently, a new definition of monitorability was given by Pnueli in [2] and it has been proven in [4] that safety and co-safety properties represent only a proper subset of the space of the monitorable properties.

Runtime enforcement is an extension of runtime verification aiming to circumvent property violations. It was initiated by the work of Schneider [9] on what has been called *security automata*. In this work the monitors watch the current execution sequence and halt the underlying program whenever it deviates

from the desired property. Such security automata are able to enforce the class of safety properties [10] stating that *something bad can never happen*. Later, Viswanathan [11] noticed that the class of enforceable properties is impacted by the computational power of the enforcement monitor. As the enforcement mechanism can implement no more than computable functions, the enforceable properties are included in the decidable ones.

More recently, Ligatti and al. [12] showed that it is possible to enforce at runtime more than safety properties. Using a more powerful enforcement mechanism called *edit-automata*, it is possible to enforce the larger class of *infinite renewal properties*. Within the classical safety-liveness dichotomy, the renewal class is a super set of the safety class which contains some liveness properties (but not all). More than simply halting an underlying program, edit-automata can also "suppress" (i.e. freeze) and "insert" (frozen) actions in the current execution sequence.

Several tools have been proposed in this context, and in practice there is not always a clear distinction between runtime-verification and runtime-enforcement (for instance a verification monitor may execute an exception handler when detecting an error, hence modifying the initial program execution). The question we consider in this work is then the following: what are the classes of properties that can be handled at runtime, and is there a distinction between these two techniques ? This question is not original in itself, but we propose here to address it within a unified framework: the safety-progress (SP) classification of properties [13, 14]. The paper contributions are then the following:

- to improve some recent results related to property enforcement [15, 16], giving a more accurate classification of enforceable properties;
- to integrate in the same framework some existing results related to property monitoring [2–4], and to propose an alternative definition of property monitoring, leveraging the semantics of finite execution sequences;
- to get a generic monitor synthesis technique, allowing to produce either a verification or an enforcement monitor from a same property description.

Paper Organization. The remainder of this article is organized as follows. Sect. 2 introduces some preliminary notations used throughout this paper. Sect. 3 overviews related work on the issues addressed in this paper. In Sect. 4, we provide minimal background on the safety-progress classification of properties in a runtime verification context. Sect. 5 is dedicated to the study of the space of monitorable properties, while Sect. 6 studies the space of enforceable properties. In Sect. 7, we present the synthesis of runtime verification and enforcement monitors. We give some concluding remarks and future works in Sect. 8.

Complete proofs and more details are given in [17].

2 Preliminaries and notations

This section introduces some background, namely the notions of *program execution sequences* and *program properties*.

2.1 Sequences, and execution sequences

Sequences and execution sequences. Considering a finite set of elements E, we define notations about sequences of elements belonging to E. A sequence σ containing elements of E is formally defined by a total function $\sigma : I \rightarrow E$ where I is either the integer interval $[0, n]$ for some $n \in \mathbb{N}$, or $\mathbb{N}$ itself (the set of natural numbers). We denote by E^* the set of finite sequences over E (partial function from $\mathbb{N}$), by E^+ the set of non-empty finite sequences over E, and by E^ω the set of infinite sequences over E. The set $E^\infty = E^* \cup E^\omega$ is the set of all sequences over E. The empty sequence of E is denoted by ϵ_E or ϵ when clear from context. The length (number of elements) of a finite sequence σ is noted $|\sigma|$ and the $(i+1)$-th element of σ is denoted by σ_i. For two sequences $\sigma \in E^*, \sigma' \in E^\infty$, we denote by $\sigma \cdot \sigma'$ the concatenation of σ and σ', and by $\sigma \prec \sigma'$ the fact that σ is a strict prefix of σ' (resp. σ' is a strict suffix of σ). The sequence σ is said to be a strict prefix of $\sigma' \in \Sigma^\infty$ when $\forall i \in \{0, \ldots, |\sigma| - 1\} \cdot \sigma_i = \sigma'_i$ and $|\sigma| < |\sigma'|$. When $\sigma' \in E^*$, we note $\sigma \preceq \sigma' \stackrel{\text{def}}{=} \sigma \prec \sigma' \vee \sigma = \sigma'$. For $\sigma \in E^\infty$ and $n \in \mathbb{N}$, $\sigma_{\ldots n}$ is the sub-sequence containing the $n+1$ first elements of σ. Also, when $|\sigma| > n$, the subsequence $\sigma_{n\ldots}$ is the sequence containing all elements of σ but the n first ones.

A program $\mathcal{P}$ is considered as a generator of execution sequences. We are interested in a restricted set of operations the program can perform. These operations influence the truth value of properties the program is supposed to fulfill. Such execution sequences can be made of access events on a secure system to its ressources, or kernel operations on an operating system. In a software context, these events may be abstractions of relevant instructions such as variable modifications or procedure calls. We abstract these operations by a finite set of *events*, namely a vocabulary Σ. We denote by $\mathcal{P}_\Sigma$ a program for which the vocabulary is Σ. The set of execution sequences of $\mathcal{P}_\Sigma$ is denoted by $Exec(\mathcal{P}_\Sigma) \subseteq \Sigma^\infty$. This set is *prefix-closed*, that is $\forall \sigma \in Exec(\mathcal{P}_\Sigma), \sigma' \in \Sigma^* \cdot \sigma' \preceq \sigma \Rightarrow \sigma' \in Exec(\mathcal{P}_\Sigma)$. In the remainder of this article, we consider a vocabulary Σ.

2.2 Properties

Properties as sets of execution sequences. A *finitary property* (resp. an *infinitary property*, a *property*) is a subset of execution sequences of Σ^* (resp. Σ^ω, Σ^∞). Considering a given finite (resp. infinite, finite or infinite) execution sequence σ and a property ϕ (resp. φ, θ), when $\sigma \in \phi$, noted $\phi(\sigma)$ (resp. $\sigma \in \varphi$, noted $\varphi(\sigma)$, $\sigma \in \theta$, noted $\theta(\sigma)$), we say that σ *satisfies* ϕ (resp. φ, θ). A consequence of this definition is that properties we will consider are restricted to *single* execution sequences[1], excluding specific properties defined on powersets of execution sequences (like fairness, for instance).

[1] This is the distinction, made by Schneider [9], between properties and (general) policies. The set of properties (defined over single execution sequences) is a subset of the set of policies (defined over sets of execution sequences).

Runtime properties. In this paper we are interested in runtime properties. As stated in the introduction, we consider finite and infinite execution sequences (that a program may produce). Runtime verification properties should characterize satisfaction for both kinds of sequences in a uniform way. As so, we introduce r-properties (runtime properties) as pairs $(\phi, \varphi) \subseteq \Sigma^* \times \Sigma^\omega$. Intuitively, the finitary property ϕ represents the desirable property that finite execution sequences should fulfill, whereas the infinitary property φ is the expected property for infinite execution sequences. The definition of negation of a r-property follows from definition of negation for finitary and infinitary properties. For a r-property (ϕ, φ), we define $\overline{(\phi, \varphi)}$ as $(\overline{\phi}, \overline{\varphi})$. Boolean combinations of r-properties are defined in a natural way. For $* \in \{\cup, \cap\}$, $(\phi_1, \varphi_1) * (\phi_2, \varphi_2) = (\phi_1 * \phi_2, \varphi_1 * \varphi_2)$. Considering an execution sequence $\sigma \in \mathit{Exec}(\mathcal{P}_\Sigma)$, we say that σ satisfies (ϕ, φ) when $\sigma \in \Sigma^* \wedge \phi(\sigma) \vee \sigma \in \Sigma^\omega \wedge \varphi(\sigma)$. For a r-property $\Pi = (\phi, \varphi)$, we note $\Pi(\sigma)$ (resp. $\neg\Pi(\sigma)$) when σ satisfies (resp. does not satisfy) (ϕ, φ).

Evaluation of r-properties. Monitorability, enforceability, and monitor synthesis are based on the evaluation of r-properties. Evaluating an execution sequence σ wrt. a r-property consists in producing a verdict regarding the current property-satisfaction of σ or future satisfactions of the possible σ-continuations. The verdicts considered here are not usual boolean values: they are truth-values taken from a truth-domain. A truth-domain is a lattice, *i.e.* a partially ordered set with an upper-bound and a lower-bound. Considering a truth-domain $\mathbb{B}$, a r-property Π and an execution sequence σ, the evaluation of $\sigma \in \Sigma^*$ wrt. Π in $\mathbb{B}$, noted $[\![\Pi]\!]_{\mathbb{B}}(\sigma)$, is an element of $\mathbb{B}$ depending on $\Pi(\sigma)$ and satisfaction of σ-continuations (*i.e.* $\{\sigma' \in \Sigma^\infty \mid \sigma \prec \sigma'\}$) wrt. Π.

The sets of monitorable and enforceable properties (Sect. 5 and 6) relies upon the considered truth-domain and the chosen evaluation function.

3 Related Work

This section overviews related work in the topics addressed in this paper. First we recall previous characterizations on the properties that can be *verified* at runtime (monitorable properties). Then, we recall previous characterization for runtime enforcement (enforceable properties). Next, we overview previous work on the synthesis of monitors for runtime verification and enforcement.

3.1 Runtime verification (monitorable) properties

Monitorability in the sense of [2]: Pnueli and al. give a notion of monitorable properties relying on the notion of verdict determinacy for an infinite sequence. More precisely, considering a finite sequence $\sigma \in \Sigma^*$, a property $\theta \subseteq \Sigma^\infty$ is negatively determined (resp. positively determined) by an execution sequence σ if σ and all its extension do not satisfy (resp. satisfy) θ. Then, θ is σ-monitorable if σ has an extension *s.t.* θ is negatively or positively determined by this extension. Finally, θ is monitorable, if it is σ-monitorable for every σ. In Sect. 5, we give the formal definition in the context of r-properties.

The idea is that it becomes unnecessary to continue the execution of a θ-monitor after reading σ if θ is not σ-monitorable. The intent of [2] was to characterize when it is worth monitoring a property.

Monitorability in the sense of [4]: Bauer and al. inspired from Pnueli's definition of monitorable properties to propose a slightly different one based on the notion of good and bad prefix introduced in model-checking [18]. The intuitive idea is that with monitorable properties it is possible to "detect" a violation or validation of infinitary properties with finite sequences. Considering an infinitary property $\varphi \subseteq \Sigma^\omega$, a prefix σ is said to be a bad prefix, noted $\mathit{bad\_prefix}(\sigma, \varphi)$ (resp. good prefix, noted $\mathit{good\_prefix}(\sigma, \varphi)$) of φ if $\forall w \in \Sigma^\omega \cdot \neg\varphi(\sigma \cdot w)$ (resp. $\forall w \in \Sigma^\omega \cdot \varphi(\sigma \cdot w)$). Then, a prefix σ is said to be ugly if it has no good nor bad continuation, *i.e.* $\nexists v \in \Sigma^\omega \cdot \mathit{bad\_prefix}(\sigma \cdot v, \varphi) \vee \mathit{good\_prefix}(\sigma \cdot v, \varphi)$. Finally, a property is said to be monitorable if it does not have ugly prefix, formally: $\forall \sigma \in \Sigma^*, \exists v \in \Sigma^\omega \cdot \mathit{bad\_prefix}(\sigma \cdot v, \varphi) \vee \mathit{good\_prefix}(\sigma \cdot v, \varphi)$.

Previous characterization of monitorable properties: Bauer and al. have shown that, according to this definition, the set of monitorable properties is a strict super set of safety and co-safety properties. These classes of properties are taken from the classical safety-liveness classification of properties [19, 20]. They also gave an example of request/acknowledge property which is not monitorable. Such a property can be framed in the set of response properties (see Sect. 4) wrt. the SP classification (see Ex. 1 in Sect. 5).

3.2 Runtime enforcement (enforceable) properties

In [10], the authors proposed a classification of enforceable properties with the regard of a program as a Turing machine. Their purpose was to delineate the set of enforceable properties according to the mechanism used for the enforcement purpose. Properties are classified according to the modification the enforcement mechanism can perform on the underlying program. The mechanisms can be characterized as static analysis, runtime execution monitor and program rewriting. Other works [9, 11, 12, 21, 16] focused on particular runtime enforcement monitors and proposed a characterization of enforceable properties with those mechanisms.

Property enforcement by an enforcement monitor (EM) is usually defined as the conjunction of the two following constraints:

soundness: the output sequence should satisfy the underlying property
transparency: the input sequence should be modified *in a minimal way*, namely if it already verifies the property it should remain unchanged (up to a given equivalence relation), otherwise its *longest prefix* satisfying the property should be issued.

Security automata and decidable safety properties: Schneider introduced security automata (a variant of Büchi automata) as the first runtime mechanism for enforcing properties in [9]. The set of enforceable properties with this kind of security automata is the set of safety properties. Then [10] Schneider, Hamlen,

and Morisett refined the set of enforceable properties and show that these security automata were in fact restrained by some computational limits. Indeed, Viswanathan [11] noticed that the class of enforceable properties is impacted by the computational power of the enforcement monitor. As the enforcement mechanism can implement no more than computable functions, the enforceable properties are included in the decidable ones. Hence, they showed in [10] that the set of safety properties is a strict superior limit to the power of (execution) enforcement monitors defined as security automata.

Edit-automata and infinite renewal properties: Ligatti and al. [12, 21] introduced *edit-automata* as runtime monitors. Depending on the current input and its control state, an edit-automata can either insert a new action by replacing the current input, or suppress it. The properties enforced by edit-automata are called *infinite renewal* properties: it is a superset of safety properties and contains some liveness properties (but not all). Then a property θ is said to be an infinite renewal property iff $\forall \sigma \in \Sigma^{\infty}, \theta(\sigma) \Rightarrow \forall \sigma' \in \Sigma^{*}, \sigma' \prec \sigma \Rightarrow \exists \sigma'', \sigma' \preceq \sigma'' \prec \sigma \wedge \theta(\sigma'')$.

Generic runtime enforcers and response properties: In [16] we introduced a generic notion of EM encompassing previous mechanisms and gave a lower-bound on the space of properties they can enforce in the SP classification (see Sect. 4).

3.3 Synthesis of monitors

For runtime verification: Generally, runtime verification monitors are generated from LTL-based specifications, as seen recently in [4, 22]. Alternatively, ω-regular expressions have been used as a basis for generating monitors, as for example in [8]. An exhaustive list of works on monitor synthesis is far beyond the scope of this paper. We refer to [1, 23, 5] for a more exhaustive list.

For runtime enforcement: In [24] Martinelli and Matteucci tackle the synthesis of enforcement mechanism as defined by Ligatti. More generally the authors consider security automata and edit-automata. The monitor is modelled by an algebraic operator expressed in CCS. The program under scrutiny is then a term $Y \rhd_K X$ where X is the target program, Y the controller program and $\rhd_K$ the operator modeling the monitor where K is the kind of monitor (truncation, insertion, suppression or edit). The desired property for the underlying system is formalized using μ-calculus. In [25] Matteucci extends the approach in the context of realtime systems. In [15] we defined transformations for some classes of the safety-progress classification of properties. Those class-specific transformations take as input a Streett automaton recognizing a property and produce an enforcement monitor for this property.

4 The SP classification in a runtime context

This section presents minimal theoritical background on the safety-progress classification of properties, introduced by Manna and Pnueli in [13, 14], in a runtime

verification context. This classification introduced a hierarchy between properties defined as *infinite* execution sequences. We extend the classification to deal with finite-length execution sequences. As so, we consider r-properties which are suitable to express runtime properties. This hierarchy presents properties in a uniform way according to 4 views: a language (seeing properties as sets of sequences), a logical (seeing properties as LTL formulas), a topological (seeing properties as open or closed sets), and an automata view (seeing properties as Streett automata [26]).

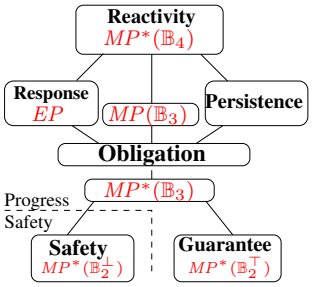


Fig. 1. The SP classification

We only present the results about the automata view as needed for ongoing discussions in this paper. A graphical representation of the safety-progress classification of properties is depicted in Fig. 1. Further details and results can be found in [27]. For each class of the SP classification it is possible to syntactically characterize a recognizing automaton. We define a variant of deterministic and complete Streett automata (introduced in [26]) for property recognition by adding to original Streett automata a finite-sequence recognizing criterion in such a way that these automata uniformly recognize r-properties.

Definition 1 (Streett m-automaton). *A deterministic Streett m-automaton is a tuple $\mathcal{A} = (Q, q_{\text{init}}, \Sigma, \longrightarrow, \{(R_1, P_1), \ldots, (R_m, P_m)\})$ defined relatively to a set of events Σ. The set Q is the set of automaton states, $q_{\text{init}} \in Q$ is the initial state. The function $\longrightarrow: Q \times \Sigma \to Q$ is the transition function. In the following, for $q, q' \in Q, e \in \Sigma$ we abbreviate $\longrightarrow (q, e) = q'$ by $q \xrightarrow{e} q'$. The set $\{(R_1, P_1), \ldots, (R_m, P_m)\}$ is the set of accepting pairs, for all $i \leq m$, $R_i \subseteq Q$ are the sets of recurrent states, and $P_i \subseteq Q$ are the sets of persistent states.*

We refer to an automaton with m accepting pairs as an m-automaton. When $m = 1$, a 1-automaton is also called a *plain*-automaton, and we refer to R_1 and P_1 as R and P.

For $\sigma \in \Sigma^\infty$, the *run* of σ on $\mathcal{A}$ is the sequence of states involved by the execution of σ on $\mathcal{A}$. It is formally defined as $run(\sigma, \mathcal{A}) = q_0 \cdot q_1 \cdots$ where $\forall i \cdot (q_i \in Q^{\mathcal{A}} \wedge q_i \xrightarrow{\sigma_i}_{\mathcal{A}} q_{i+1}) \wedge q_0 = {q_{\text{init}}}^{\mathcal{A}}$. The *trace* resulting in the execution of σ on $\mathcal{A}$ is the unique sequence (finite or not) of tuples $(q_0, \sigma_0, q_1) \cdot (q_1, \sigma_1, q_2) \cdots$ where $run(\sigma, \mathcal{A}) = q_0 \cdot q_1 \cdots$. The uniqueness of the trace is due to the fact that we consider only deterministic Streett automata.

Also we consider the notion of infinite visitation of an execution sequence $\sigma \in \Sigma^\omega$ on a Streett automaton $\mathcal{A}$, denoted $vinf(\sigma, \mathcal{A})$, as the set of states appearing infinitely often in $run(\sigma, \mathcal{A})$. It is formally defined as follows: $vinf(\sigma, \mathcal{A}) = \{q \in Q^{\mathcal{A}} \mid \forall n \in \mathbb{N}, \exists m \in \mathbb{N} \cdot m > n \wedge q = q_m\}$ with $run(\sigma, \mathcal{A}) = q_0 \cdot q_1 \cdots$.

Acceptance conditions (finite and infinite sequences) are defined using the accepting pairs.

Definition 2 (Acceptance condition (finite sequences)). *For $\sigma \in \Sigma^*$ s.t. $|\sigma| = n$, we say that the m-automaton $\mathcal{A}$ accepts σ if $(\exists q_0, \ldots, q_n \in Q^{\mathcal{A}} \cdot run(\sigma, \mathcal{A}) = q_0 \cdots q_n \wedge q_0 = {q_{\text{init}}}^{\mathcal{A}}$ and $\forall i \in [1, m] \cdot q_n \in P_i \cup R_i)$.*

Definition 3 (Acceptance condition (infinite sequences)). *For $\sigma \in \Sigma^\omega$, we say that $\mathcal{A}$ accepts σ if $\forall i \in [1,m] \cdot inf(\sigma,\mathcal{A}) \cap R_i \neq \emptyset \vee inf(\sigma,\mathcal{A}) \subseteq P_i$.*

Note that this notion of acceptance for finite sequences exactly coincides with the one proposed by [4] for the RV-LTL temporal logic.

The hierarchy of automata. By setting syntactic restrictions on a Streett automaton, we modify the kind of properties recognized by such an automaton. Each class is characterized by some conditions on the transition function and the accepting pairs.

A *safety automaton* is a plain automaton such that $R = \emptyset$ and there is no transition from a state $q \in \overline{P}$ to a state $q' \in P$. A *guarantee automaton* is a plain automaton such that $P = \emptyset$ and there is no transition from a state $q \in R$ to a state $q' \in \overline{R}$. An *m-obligation automaton* is an m-automaton such that for each i in $[1,m]$: there is no transition from $q \in \overline{P_i}$ to $q' \in P_i$ and there is no transition from $q \in R_i$ to $q' \in \overline{R_i}$. A *response automaton* is a plain automaton such that $P = \emptyset$, a *persistence automaton* is a plain automaton such that $R = \emptyset$. And a *reactivity automaton* is any unrestricted automaton.

It is possible to link the syntactic characterizations on automata to the semantic characterization of the properties they specify. As stated by the following definition (transposed from an initial theorem [13, 14]).

Definition 4. *A r-property (ϕ,φ) is a κ-r-property iff it is specifiable by a κ-automaton, where $\kappa \in$ {safety, guarantee, obligation, response, persistence, reactivity}*

We note Safety(Σ) (resp. Guarantee(Σ), Obligation(Σ), Response(Σ), Persistence(Σ), Reactivity(Σ)) the set of safety (resp. guarantee, obligation, response, persistence, reactivity) r-properties over Σ. Moreover, a r-property of a given class is *pure* when it is a property of none of others sub-classes.

5 Monitorability wrt. the SP classification

In this section we first revisit existing monitorability results in the safety-progress classification of properties. Second, we propose an alternative definition of monitorability. In fact, characterizing the space of "monitorable" properties depends on several parameters: the property semantics for *finite* sequence, the set of monitor verdicts we consider, and the exact definition of monitoring.

5.1 Classical definition of monitoring

The main objective of monitoring, in its classical definition, is to evaluate an (infinitary) property φ on a possibly infinite execution sequence from one of its *finite* prefix. This definition is formalized below.

Definition 5 (Positive/Negative determinacy [2]). *A r-property $\Pi \subseteq \Sigma^* \times \Sigma^\omega$ is said to be:*

- *negatively determined by $\sigma \in \Sigma^*$ if $\neg\Pi(\sigma) \wedge \forall \mu \in \Sigma^\infty \cdot \neg\Pi(\sigma \cdot \mu)$;*
- *positively determined by $\sigma \in \Sigma^*$ if $\Pi(\sigma) \wedge \forall \mu \in \Sigma^\infty \cdot \Pi(\sigma \cdot \mu)$.*

Definition 6 (Monitorable [2] r-properties). *A r-property Π is:*

- *σ-monitorable, if there exists a (finite) $\mu \in \Sigma^*$ s.t. Π is positively or negatively determined by $\sigma \cdot \mu$;*
- *monitorable, if it is monitorable for every $\sigma \in \Sigma^*$.*

The underlying assumed truth-domain is $\mathbb{B}_3 = \{\top, ?, \bot\}$. Value $\top$ is used to express property satisfaction when the property is positively determined. Value $\bot$ is used to express property violation when the property is negatively determined. Whereas value ? is used to express that no verdict can be produced. (See Def. 7). Within the $\mathbb{B}_3$ lattice, boolean operators $\vee$ and $\wedge$ are defined respectively as upper and lower bounds. In this context it can be shown that the set of monitorable properties with $\mathbb{B}_3$ strictly contains the set of obligation properties. In the following, for a truth-domain $\mathbb{B}$, we note $MP(\mathbb{B})$ the space of monitorable properties according to this definition.

Theorem 1 ($Obligation(\Sigma) \subset MP(\mathbb{B}_3)$). *The obligation properties are strictly contained in the set of monitorable properties with $\mathbb{B}_3$.*

Proof. Obligation r-properties are obtained by boolean combinations of safety and guarantee r-properties. For $k \in \mathbb{N}$, a k-obligation r-property is a r-property $\bigcap_{i=1}^{k}(\text{Safety}_i \cup \text{Guarantee}_i)$, where Safety_i and Guarantee_i are safety and guarantee r-properties. The set of all k-obligation r-properties for $k \in \mathbb{N}$ is the set of obligation r-properties.

Let $\Pi \in \text{Obligation}(\Sigma)$, there exists $k \in \mathbb{N}$ s.t. $\Pi \in k\text{–Obligation}(\Sigma)$. The proof relies on an easy induction on k and uses the following facts:

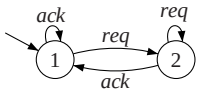
- Safety and guarantee properties are monitorable. By examining the syntactic restrictions of an automaton recognizing a safety or a guarantee property, we have: for all $\sigma \in \Sigma^*$ there exists a continuation μ s.t. this property is positively or negatively determined by $\sigma \cdot \mu$.
- Union and intersection of two monitorable properties is monitorable.
- Ex. 1 show that the inclusion is strict.

Thus, we have extended the previous bound established by Bauer and al. in [4] stating that $\text{Safety}(\Sigma) \cup \text{Guarantee}(\Sigma) \subset MP(\mathbb{B}_3)$[2]. Indeed, the set of obligation properties is a strict super set of the union of safety and guarantee properties.

Beyond Obligation properties. Following the classical definition of monitorability, it is possible to show that there exist non-monitorable and monitorable properties for super-classes of the Obligation class. The above two properties are pure response properties, one is not monitorable, the other one is.

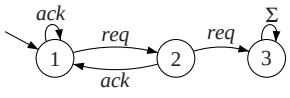
Example 1 (Non-monitorable response property [4]). The (response) property "Every request should be acknowledged" is not monitorable.

[2] In [4], guarantee properties are named co-safety properties.



This property is represented by the Streett (response) automaton on the left with $R = \{1\}$. For this property, there are two limitations for monitoring with the considered truth-domain and definition of monitorability. First, it is impossible to distinguish correct (ending in state 1) and incorrect finite sequences (ending in state 2): both evaluate to "?". Second, for all finite sequences, it is never possible to decide $\top$ or $\bot$ since every finite sequence can be extended to correct or incorrect infinite continuations.

Example 2 (Monitorable response property). The (response) property "Every request should be acknowledged and two successive requests (without acknowledgement) is forbidden" is monitorable.



This property is represented by the Streett (response) automaton on the left with $R = \{1\}$. Intuitively, given an execution sequence, this r-property can always be negatively determined by one of its extension.

Monitorability with $\mathbb{B}_2$. Restraining $\mathbb{B}_3$ to a truth-domain of cardinality 2 allows only either positive or negative determinacy, and hence reduces the set of monitorable properties. However, there is no simple characterization of these properties in the safety-progress hierarchy. Intuitively one may think that with $\mathbb{B}_2^{\bot} = \{?, \bot\}$, the set of monitorable properties would be the set of safety properties. But in fact, there are numerous safety properties which can never be negatively determined. For example, the r-property $true = (\Sigma^*, \Sigma^\omega)$ cannot be negatively determined nor falsified. Moreover all safety properties which are valid forever for execution sequences longer than a given k are not $\sigma - \mathbb{B}_2^{\bot}$-monitorable when $|\sigma| > k$. For those kind of properties a monitor would produce only sequences of "?" when evaluating an execution sequence. Similarly, there exist many guarantee properties that cannot be positively determined, and therefore not monitorable with $\mathbb{B}_2^{\top} = \{?, \top\}$.

However, in Sect. 7, we give a syntactic criterion on Streett automata to determine whether a r-property (recognized by a Streett automaton) is monitorable or not under these conditions.

5.2 An alternative definition of monitoring

The interest of previous definitions of monitorability is due to two facts: the underlying truth-domain is 2-valued or 3-valued and the aim is the detection of verdict of infinitary properties. Although it is possible to give a semantics to all reactive properties with either a 2-valued or 3-valued truth-domain, the question is whether those values make sense for some properties in a monitoring context.

As noticed in [4, 23], it seems interesting to investigate further the space of monitorable properties, and to answer more precisely questions like "what verdict to issue if the program execution stops here". This means a better distinction between finite sequences which evaluate to ? in a 2-valued or 3-valued truth-domain.

Hence, the authors proposed to consider a 4-valued truth-domain $\mathbb{B}_4 = \{\top, \top_p, \bot_p, \bot\}$. The truth-value $\top_p$ (resp. $\bot_p$) denotes "presumably true" (resp. "presumably false") and it express "Π-satisfaction (resp. Π-violation) if the program execution stops here". Boolean operators $\vee$ and $\wedge$ are defined in [4]. Using $\mathbb{B}_4$ leads to an alternative definition of monitoring. This new definition leverages the evaluation of finite sequences in the Safety-Progress classification framework.

Property evaluation in a truth-domain. We first introduce how, given a r-property, we evaluate an execution sequence in the truth-domains we considered so far.

Definition 7 (Property evaluation wrt. a truth-domain). *For each of the possible truth-domain $\mathbb{B}$, we define the evaluation functions $[\![\cdot]\!]_{\mathbb{B}}(\cdot) : (\Sigma^* \times \Sigma^\omega) \times \Sigma^* \to \mathbb{B}$ as follows:*

For $\mathbb{B}_2^\bot$:
$[\![\Pi]\!]_{\mathbb{B}_2^\bot}(\sigma) = \bot$ *if* $\neg\Pi(\sigma) \wedge \forall \mu \in \Sigma^\infty \cdot \neg\Pi(\sigma \cdot \mu)$,
$[\![\Pi]\!]_{\mathbb{B}_2^\bot}(\sigma) = ?$ *otherwise.*

For $\mathbb{B}_2^\top$:
$[\![\Pi]\!]_{\mathbb{B}_2^\top}(\sigma) = \top$ *if* $\Pi(\sigma) \wedge \forall \mu \in \Sigma^\infty \cdot \Pi(\sigma \cdot \mu)$,
$[\![\Pi]\!]_{\mathbb{B}_2^\top}(\sigma) = ?$ *otherwise.*

For $\mathbb{B}_3$:
$[\![\Pi]\!]_{\mathbb{B}_3}(\sigma) = \bot$ *if* $\neg\Pi(\sigma) \wedge \forall \mu \in \Sigma^\infty \cdot \neg\Pi(\sigma \cdot \mu)$,
$[\![\Pi]\!]_{\mathbb{B}_3}(\sigma) = \top$ *if* $\Pi(\sigma) \wedge \forall \mu \in \Sigma^\infty \cdot \Pi(\sigma \cdot \mu)$,
$[\![\Pi]\!]_{\mathbb{B}_3}(\sigma) = ?$ *otherwise.*

For $\mathbb{B}_4$:
$[\![\Pi]\!]_{\mathbb{B}_4}(\sigma) = [\![\Pi]\!]_{\mathbb{B}_3}(\sigma)$ *if* $[\![\Pi]\!]_{\mathbb{B}_3}(\sigma) = \bot$ *or* $[\![\Pi]\!]_{\mathbb{B}_3}(\sigma) = \top$,
$[\![\Pi]\!]_{\mathbb{B}_4}(\sigma) = \top_p$ *if* $[\![\Pi]\!]_{\mathbb{B}_3}(\sigma) = ?$ *and* $\Pi(\sigma)$
$[\![\Pi]\!]_{\mathbb{B}_4}(\sigma) = \bot_p$ *if* $[\![\Pi]\!]_{\mathbb{B}_3}(\sigma) = ?$ *and* $\neg\Pi(\sigma)$

An alternative definition of monitorability. Intuitively, the monitorability notion we propose relies on the ability of a given monitor to distinguish between *good* and *bad* finite execution sequences with respect to a property Π.

Definition 8 (Monitorability). *A r-property $\Pi = (\phi, \varphi)$ is said to be monitorable with the truth-domain $\mathbb{B}$, or $\mathbb{B}$-monitorable 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 $MP^*(\mathbb{B})$, the set of monitorable properties with truth domain $\mathbb{B}$ according to this definition.

Theorem 2 (Multi-valued characterization of monitorability). *The sets of monitorable properties according to the truth domains considered so far are the following:*

$$MP^*(\mathbb{B}_2^\bot) = \text{Safety}(\Sigma)$$

$MP^*(\mathbb{B}_2^\top) = \text{Guarantee}(\Sigma)$
$MP^*(\mathbb{B}_3) \subset \text{Obligation}(\Sigma)$ *and* $\text{Safety}(\Sigma) \cup \text{Guarantee}(\Sigma) \subset MP^*(\mathbb{B}_3)$
$MP^*(\mathbb{B}_4) = \text{Reactivity}(\Sigma)$

Example 3 (Monitoring of an obligation property). Let consider the LTL property $\Pi = \Box p \vee \Diamond q$, stating that the state-predicate p should *always* hold or q should *eventually* hold. This is an obligation property. Let consider the following execution sequences: $\sigma_{good} = \{p\} \cdot \{p\}$ and $\sigma_{bad} = \emptyset \cdot \{p\}$. In $\mathbb{B}_3$, we have $[\![\Pi]\!]_{\mathbb{B}_3}(\sigma_{good}) = [\![\Pi]\!]_{\mathbb{B}_3}(\sigma_{bad}) =?$. Thus, Π is not $\mathbb{B}_3$-monitorable. However, Π is $\mathbb{B}_4$-monitorable and $[\![\Pi]\!]_{\mathbb{B}_4}(\sigma_{good}) = \top_p$ and $[\![\Pi]\!]_{\mathbb{B}_4}(\sigma_{bad}) = \bot_p$.

We will show in Sect. 7 that, for a given finite sequence σ, $[\![\Pi]\!]_{\mathbb{B}_4}(\sigma)$ is easy to compute from a Streett automaton recognizing Π.

Remark 1. It is worth noticing that property interpretation of finite sequences with "weak verdicts" ($\bot_p$, $\top_p$) extends to infinite sequences in a consistent way, depending on the class of properties under consideration:

- for a safety property Π, $(\forall i \in \mathbb{N}, [\![\Pi]\!](\sigma_{...i}) = \top_p) \Rightarrow \Pi(\sigma)$
- for a guarantee property Π, $(\forall i \in \mathbb{N}, [\![\Pi]\!](\sigma_{...i}) = \bot_p) \Rightarrow \neg\Pi(\sigma)$
- for a response property Π, $(\overset{\infty}{\exists}\, i \in \mathbb{N}, [\![\Pi]\!](\sigma_{...i}) = \top_p) \Rightarrow \Pi(\sigma)$
- for a persistence property Π, $(\overset{\infty}{\exists}\, i \in \mathbb{N}, [\![\Pi]\!](\sigma_{...i}) = \bot_p) \Rightarrow \neg\Pi(\sigma)$

6 Enforceability wrt. the SP classification

In Sect. 3, we have seen that the previous proposed spaces of enforceable properties were delineated according to the mechanism used to enforce the properties. Such mechanisms should obey the soundness and transparency constraints. We choose here to take an alternative approach. Indeed we believe that the set of enforceable properties can be characterized independently from any enforcement mechanism complying to these constraints. This will give us an upper-bound of the set of enforceable properties.

6.1 Enforcement criteria

A consequence of transparency is that a r-property (ϕ, φ) will be considered as *enforceable* only if each incorrect infinite sequence has a *longest* correct prefix. This means that any infinite incorrect sequence should have only a finite number of correct prefixes. This transparency demand can be seen from the language and automata views of r-properties. Thus we give two equivalent enforcement criteria for r-properties for each view of r-properties[3].

Definition 9 (Enforcement criterion (language view)). *A r-property (ϕ, φ) is said to be enforceable iff*

$$\forall \sigma \in \Sigma^\omega, \neg\varphi(\sigma) \Rightarrow (\exists \sigma' \in \Sigma^*, \sigma' \prec \sigma, \forall \sigma'' \in \Sigma^* \cdot \sigma' \prec \sigma'' \Rightarrow \neg\phi(\sigma'')) \qquad (1)$$

[3] Note that those (equivalent) criteria differ from the existence of bad prefixes. Bad prefixes are sequences which cannot be extended to correct (finite or infinite) ones

A r-property Π recognized by a Streett automaton $\mathcal{A}_\Pi$ is said to be enforceable iff every maximal strongly-connected component (SCC) of $\overline{R}$-states contain (only) either P-states or $\overline{P}$-states.

Definition 10 (Enforcement criterion (automata view)). *Denoting $\mathcal{S}(\mathcal{A}_\Pi)$ the set of SCC of $\mathcal{A}_\Pi$, an m-automaton, recognizing Π, Π is said to be enforceable iff*

$$\forall i \in [1,m], \forall s \in \mathcal{S}(\mathcal{A}_\Pi), s \subseteq \overline{R_i} \Rightarrow (s \subseteq \overline{P_i} \vee s \subseteq P_i) \tag{2}$$

Enforcement criteria of Def. 9 and 10 are equivalent for basic classes of properties, as stated below.

Property 1 (Equivalence between enforcement criteria (basic classes)). Considering a r-property $\Pi = (\phi, \varphi)$ of a basic class, recognized by a Streett automaton $(Q^{\mathcal{A}_\Pi}, {q_{\text{init}}}^{\mathcal{A}_\Pi}, \Sigma, \rightarrow^{\mathcal{A}_\Pi}, \{(R,P)\}$, we have that:

$$(1) \Leftrightarrow \forall s \in \mathcal{S}(\mathcal{A}_\Pi), s \subseteq \overline{R} \Rightarrow (s \subseteq \overline{P} \vee s \subseteq P).$$

Proof. This proof relies on the computation of maximal strongly connected components [28] of a Streett automaton (SCC). The proof is in two stages by proving implications in both ways.

(1) $\Rightarrow$ (2) Let consider a SCC of $\mathcal{A}_\Pi$ containing only $\overline{R}$-states. Suppose that there exists two states q, q' in this SCC s.t. $q \in P$ and $q' \notin P$. As q and q' are in a SCC, there exists a path from q to q' and from q' to q in $\mathcal{A}_\Pi$. Then there would exist an infinite execution sequence σ s.t. the run of σ on $\mathcal{A}_\Pi$ contains infinite occurrences of q and q'. As this SCC is made of $\overline{R}$-states, σ is not accepted by $\mathcal{A}_\Pi$ (since $inf(\sigma, \mathcal{A}_\Pi) \not\subseteq P$), *i.e.* $\neg\varphi(\sigma)$. However σ has an infinite number of "good" prefixes: all prefixes s.t. the run ends in a R-state. This is contradictory with our initial assumption.

(2) $\Rightarrow$ (1) Let consider $\sigma \in \Sigma^\omega$ s.t. $\neg\varphi(\sigma)$. As $\mathcal{A}_\Pi$ recognizes Π, σ is not accepted by $\mathcal{A}_\Pi$. As $\mathcal{A}_\Pi$ is a finite state automaton, the run of σ on $\mathcal{A}_\Pi$ visits a SCC infinitely often and can be expressed:
$run(\sigma, \mathcal{A}_\Pi) = q_0 \cdots q_{k-1} \cdot (q_i \cdots q_{i+l})^n$
with $k \leq i \wedge l \leq |Q| \wedge i = l * n + k, n \in \mathbb{N}$.
Moreover, we know that $\forall i \leq j \leq i + l \cdot q_j \in P \vee \forall i \leq j \leq i + l \cdot q_j \in \overline{P}$.
- In the first case, the sequence σ is accepted by $\mathcal{A}_\Pi$ (Def. 3):
  $inf(\sigma, \mathcal{A}_\Pi) \subseteq \{q_i, \ldots, q_{i+l}\} \subseteq P$. This is contradictory with $\neg\varphi(\sigma)$.
- In the second case, the sequence σ is not accepted by $\mathcal{A}_\Pi$:
  $inf(\sigma, \mathcal{A}_\Pi) \subseteq \{q_i, \ldots, q_{i+l}\} \not\subseteq P$. According to the finite-sequence acceptance criterion (Def. 2) and since $\forall c \in \mathbb{N}, c \geq k \Rightarrow q_c \notin P$, we obtain $\forall c \in \mathbb{N}, c \geq k \Rightarrow \neg\Pi(\sigma_{\ldots c})$.

□

Property 2 (Comparing enforcement criteria for compound classes). Considering a r-property $\Pi = (\phi, \varphi)$, recognized by a Streett automaton $(Q^{\mathcal{A}_\Pi}, {q_{\text{init}}}^{\mathcal{A}_\Pi}, \Sigma, \rightarrow^{\mathcal{A}_\Pi}, \{(R,P)\}$, we have that:

(1) $\Leftrightarrow$ (2), for Obligation properties
(1) $\Leftarrow$ (2), for Reactivity properties

Proof. We sketch the proof for those classes of properties.

For Obligation properties. Similarly to the proof of Prop. 1, the proof relies on the fact that in a m-obligation automaton, for $i \in [1, m]$, there is no transition from R_i-states to $\overline{R_i}$-states, and no transition from $\overline{P_i}$-states to P_i-states.

For Reactivity properties. Let consider $\sigma \in \Sigma^\omega$ s.t. $\neg\varphi(\sigma)$. Similarly to the proof of Prop. 1 ($\Leftarrow$ direction), the run of σ on $\mathcal{A}_\Pi$ visits a SCC infinitely often and can be expressed:

$run(\sigma, \mathcal{A}_\Pi) = q_0 \cdots q_{k-1} \cdot (q_i \cdots q_{i+l})^n$ with $k \leq i \wedge l \leq |Q| \wedge i = l * n + k, n \in \mathbb{N}$. Moreover, we know that $\forall i \leq j \leq i + l \cdot q_j \in P \vee \forall i \leq j \leq i + l \cdot q_j \in \overline{P}$. We have that $\forall \sigma' \in \Sigma^*, \sigma_{\ldots k} \preceq \sigma', \neg\Pi(\sigma')$. Indeed, otherwise it would have mean that $\forall i \in [1, m], \forall j \geq k, q_j \in P_i$. Which would have lead to $\varphi(\sigma)$ using the infinite-sequence acceptance condition of Streett automata.

The set of enforceable r-properties is denoted EP. We will now characterize this set wrt. the SP classification. Though, we will prove that the class of enforceable properties is exactly the class of response properties. The enforcement criterion in the automata view is still usefull as it provides a sufficient condition on automata. Thus, given any automaton, this gives syntactic procedure to determine whether the recognized property is enforceable.

6.2 Enforceable properties

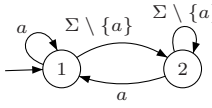
We start first by proving that response properties (defined in Sect. 4) are enforceable and give an example of persistence properties not enforceable. Then we find that the set of response properties is exactly the set of enforceable ones.

Theorem 3 (Response are enforceable). $Response(\Sigma) \subseteq EP$.

Proof. Indeed consider a response r-property $\Pi = (\phi, \varphi)$ and an execution sequence $\sigma \in \Sigma^\omega$. Π can be expressed $(R_f(\psi), R(\psi))$ ($\Pi \in Response(\Sigma)$). Let suppose that $\neg\varphi(\sigma)$. It means that $\sigma \notin R(\psi)$, *i.e.* σ has finitely many prefixes belonging to ψ. Consider the set $S = \{\sigma' \in \Sigma^* \mid \forall \sigma'' \in \Sigma^*, \sigma' \prec \sigma'' \prec \sigma \wedge \neg\psi(\sigma'')\}$ of finite sequences from which all finite continuations do not satisfy ψ. As $\neg R(\psi)$, this set is not empty. Let note σ_0 the smallest element of S regarding $\prec$. We have $\forall \sigma' \in \Sigma^*, \sigma_0 \prec \sigma' \Rightarrow \neg\psi(\sigma')$. Since $\forall \psi \subseteq \Sigma^*, R_f(\psi) \subseteq \psi$, it implies that $\forall \sigma' \in \Sigma^*, \sigma_0 \prec \sigma' \Rightarrow \neg\phi(\sigma')$. □

A straightforward consequence is that safety, guarantee and obligation r-properties are enforceable. We prove that, in fact, pure persistence properties are not enforceable.

An example of pure persistence r-property is $\Pi = (\Sigma^* \cdot a^+, \Sigma^* \cdot a^\omega)$ stating that "it will be eventually true that a always occurs". One can notice that this property is neither a safety, guarantee nor obligation property.



Π is recognized by the Streett automaton $\mathcal{A}_\Pi$ depicted on the left (with acceptance criterion $vinf(\sigma, \mathcal{A}_\Pi) \subseteq P$ and $P = \{1\}$). One can understand the enforcement limitation intuitively with the following argument: if this property was

enforceable it would imply that an EM can decide from a certain point that the underlying program will always produce the event a. However such a decision can never be taken by a monitor without memorizing the entire execution sequence beforehand. This is unrealistic for an infinite sequence. More formally, as stated in the previous section, a r-property (ϕ, φ) is enforceable if for all infinite execution sequences σ when $\neg\varphi(\sigma)$, the longest prefix of σ satisfying ϕ always exists. For the above automaton, the execution sequence $\sigma'_{bad} = (a \cdot b)^\omega$ exhibits the same issue. Indeed, the infinite sequence does not satisfy the property whereas an infinite number of its prefixes do (prefixes ending with a).
Applying enforcement criteria (Def. 9 and 10) on persistence properties, it turns out that the enforceable persistence properties are in fact response properties.

Theorem 4 (Enforceable persistence properties are response properties). $\text{Persistence}(\Sigma) \cap EP \subseteq \text{Response}(\Sigma)$.

Proof. A r-property becomes non-enforceable as soon as there exists a SCC of $\overline{R}$-states containing a P-state and a $\overline{P}$-state on its recognizing automaton (see Def. 10). Indeed, on a Streett automaton it allows infinite invalid execution sequences with an infinite number of valid prefixes. When removing this possibility on a Streett automaton, the constrained automaton can be easily translated to a response automaton. Indeed, on this constrained automaton, the states visited infinitely often are either all in P or $\overline{P}$, that is: $\forall\sigma \in \Sigma^\omega \cdot \mathit{vinf}(\sigma) \cap P \neq \emptyset \Leftrightarrow \mathit{vinf}(\sigma) \subseteq P$. On such automaton there is no difference between R-states and P-states. Consequently by retagging P-states to R, this automaton recognizes the same property. The retagged automaton is a response automaton. □

Corollary 1. *Pure persistence are not enforceable:*
$(\text{Persistence}(\Sigma) \setminus \text{Response}(\Sigma)) \cap EP = \emptyset$.

Proof. This is a direct consequence of Theorem 4. □

Corollary 2. *Pure reactivity are not enforceable:*
$\text{Reactivity}(\Sigma) \not\subseteq EP \wedge \text{Reactivity}(\Sigma) \setminus (\text{Persistence}(\Sigma) \cup \text{Response}(\Sigma)) \cap EP = \emptyset$.

Proof. This is a direct consequence of Corollary 1. A general reactivity property can be expressed as the composition of response and persistence properties. As a consequence, pure persistence properties are included in the set of reactivity properties. And consequently, the persistence part of a reactivity property is not enforceable. □

Corollary 3. *Enforceable properties are exactly response properties:*
$EP = \text{Response}(\Sigma)$.

Proof. It remains to be proven that the set of enforceable properties is included in the set of response one. Suppose that there exists an enforceable property which is not a response one. Then, according to the definition of the safety-progress hierarchy, this property would be a pure persistence or reactivity property. Consequently this property would not be enforceable. □

7 Monitor synthesis

Now we show how it is possible to obtain a monitor either for verifying or enforcing a property. Generally speaking, a monitor is a device processing an input sequence of events or states in an incremental fashion. It is purposed to yield a property-specific decision according to its goal. In (classic) runtime verification such a decision is a truth-value taken from a truth-domain. This truth-value states an appraisal of property satisfaction or violation by the input sequence. For runtime enforcement, the monitor produces a sequence of enforcement operations. The monitor uses an internal memory and applies enforcement operations to the input event and its current memory so as to modify input sequence and produce an output sequence. The relation between input and output sequence should follow enforcement monitoring constraints: soundness and transparency (Sect. 3.2). In the following we consider two Streett m-automata $\mathcal{A} = (Q^{\mathcal{A}}, {q_{\text{init}}}^{\mathcal{A}}, \longrightarrow_{\mathcal{A}}, \{(R_1, P_1), \ldots, (R_m, P_m)\})$ and $\mathcal{A}_{\Pi} = (Q^{\mathcal{A}_{\Pi}}, {q_{\text{init}}}^{\mathcal{A}_{\Pi}}, \rightarrow_{\mathcal{A}_{\Pi}}, \{(R_1, P_1), \ldots, (R_m, P_m)\}$, Π the r-property recognized by $\mathcal{A}_{\Pi}$. Also we evaluate properties only in $\mathbb{B}_4$, and consequently we abbreviate $[\![\Pi]\!]_{\mathbb{B}_4}(\cdot)$ by $[\![\Pi]\!](\cdot)$.

7.1 Characterizing states of Streett automata

We will define monitors (for verification and enforcement) from Streett automata. To do so, we will define a set of subsets of Streett automaton states. The set $\mathbb{P}^{\mathcal{A}} = \{Good^{\mathcal{A}}, Good_p^{\mathcal{A}}, Bad_p^{\mathcal{A}}, Bad^{\mathcal{A}}\}$ is a set of subsets of $Q^{\mathcal{A}}$, s.t. $Good^{\mathcal{A}}$, $Good_p^{\mathcal{A}}$, $Bad_p^{\mathcal{A}}$, $Bad^{\mathcal{A}}$ designate respectively the good (resp. presumably good, presumably bad, bad) states. The set $\mathbb{P}^{\mathcal{A}}$ is defined as follows:

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

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

Property 3 (Correspondence between $\mathbb{P}$ and $\mathbb{B}_4$). Given an m-automaton $\mathcal{A}_{\Pi}$, Π, and an execution sequence $\sigma \in \Sigma^*$ of length n s.t. $run(\sigma, \mathcal{A}_{\Pi}) = q_0 \cdots q_{n-1}$, we have that:

$q_{n-1} \in Good^{\mathcal{A}_{\Pi}} \Leftrightarrow [\![\Pi]\!](\sigma) = \top$, $\quad q_{n-1} \in Bad_p^{\mathcal{A}_{\Pi}} \Leftrightarrow [\![\Pi]\!](\sigma) = \bot_p$,
$q_{n-1} \in Good_p^{\mathcal{A}_{\Pi}} \Leftrightarrow [\![\Pi]\!](\sigma) = \top_p$, $\quad q_{n-1} \in Bad^{\mathcal{A}_{\Pi}} \Leftrightarrow [\![\Pi]\!](\sigma) = \bot$.

Proof. This proof is naturally done in four steps. Let consider an execution sequence $\sigma \in \Sigma^*$ of length n.

- Proof of $q_{n-1} \in Good^{\mathcal{A}_{\Pi}} \Leftrightarrow [\![\Pi]\!](\sigma) = \top$.
  - Suppose that $q_{n-1} \in Good^{\mathcal{A}_{\Pi}}$. Using the acceptance criterion on finite sequences, we have that σ is accepted by $\mathcal{A}_{\Pi}$. Moreover, as $\mathcal{A}_{\Pi}$ recognizes Π, we have that $\Pi(\sigma)$. Now, let consider $\mu \in \Sigma^+$ s.t. $|\sigma| + |\mu| = n' > n$ and $run(\sigma \cdot \mu, \mathcal{A}_{\Pi}) = q_0 \cdots q_{n'-1}$. We have that $\forall k \in \mathbb{N}, n \leq k \leq n'-1 \Rightarrow$

$q_k \in \bigcap_{i=0}^{m} R_i \cup P_i$ and consequently $\Pi(\sigma \cdot \mu)$. Moreover, consider $\mu \in \Sigma^\omega$, we remark that $inf(\sigma \cdot \mu, \mathcal{A}_\Pi) \subseteq \bigcap_{i=0}^{m} R_i \cup P_i$. Then, we obtain that $\forall i \in [1, m], inf(\sigma \cdot \mu, \mathcal{A}_\Pi) \cap R_i \neq \emptyset \vee inf(\sigma \cdot \mu, \mathcal{A}_\Pi) \subseteq P_i$ implying that $\Pi(\sigma \cdot \mu)$. We have $\Pi(\sigma) \wedge \forall \mu \in \Sigma^\infty, \Pi(\sigma \cdot \mu)$, *i.e.* $[\![\Pi]\!](\sigma) = \top$.

- Conversely, suppose that $[\![\Pi]\!](\sigma) = \top$. By definition, it means that $\forall \mu \in \Sigma^\infty, \Pi(\sigma \cdot \mu)$. According to the acceptance criterion of a Streett automaton, we deduce that $\forall k \geq n, \forall \mu \in \Sigma^*, run(\sigma \cdot \mu, \mathcal{A}_\Pi) = q_0 \cdots q_{n-1} \cdots q_k \Rightarrow q_k \in \bigcap_{i=0}^{m} R_i \cup P_i$. That is $Reach_{\mathcal{A}_\Pi}(q_{n-1}) \subseteq \bigcap_{i=1}^{m} (R_i \cup P_i)$, *i.e.* $q_{n-1} \in Good^{\mathcal{A}_\Pi}$.

– Proof of $q_{n-1} \in Good_p^{\mathcal{A}_\Pi} \Leftrightarrow [\![\Pi]\!](\sigma) = \top_p$. Proving that $q_{n-1} \in Good_p^{\mathcal{A}_\Pi} \Leftrightarrow [\![\Pi]\!](\sigma) = \top_p$ is straightforward by examining the finite-sequence acceptance criterion of Streett automata.

- Suppose that $q_{n-1} \in Good_p^{\mathcal{A}_\Pi}$. Using the acceptance criterion on finite sequences, we have that σ is accepted by $\mathcal{A}_\Pi$. Moreover, as $\mathcal{A}_\Pi$ recognizes Π, we have that $\Pi(\sigma)$. Now, as $Reach_{\mathcal{A}}(q) \not\subseteq \bigcup_{i=1}^{m} (\overline{R_i} \cap \overline{P_i})$, there exists a state q' of $\mathcal{A}_\Pi$ reachable from q and belonging to $\bigcup_{i=1}^{m} (\overline{R_i} \cap \overline{P_i})$. As a consequence, there exists $\mu \in \Sigma^*$ s.t. $run(\sigma \cdot \mu) = q_0 \cdots q_{n-1} \cdots q'$. With the acceptance criterion of finite sequences, we deduce that $\neg\Pi(\sigma \cdot \mu)$, *i.e.* $[\![\Pi]\!](\sigma) = \top_p$.
- Conversely, the same reasoning using the finite sequence acceptance criterion can be used to prove the desired result.

– Proof of $q_{n-1} \in Bad_p^{\mathcal{A}_\Pi} \Leftrightarrow [\![\Pi]\!](\sigma) = \bot_p$. Similarly, proving that $q_{n-1} \in Bad_p^{\mathcal{A}_\Pi} \Leftrightarrow [\![\Pi]\!](\sigma) = \bot_p$ is straightforward by examining the finite acceptance criterion of Streett automata.

– Proof of $q_{n-1} \in Bad^{\mathcal{A}_\Pi} \Leftrightarrow [\![\Pi]\!](\sigma) = \bot$. Proving that $q_{n-1} \in Bad^{\mathcal{A}_\Pi} \Leftrightarrow [\![\Pi]\!](\sigma) = \bot$ can be done following the same proof as for $q_{n-1} \in Good^{\mathcal{A}_\Pi} \Leftrightarrow [\![\Pi]\!](\sigma) = \top$.

7.2 Back to the notion of monitorability

We have seen in Sect. 5.1 that there is no exact characterization (in terms of a specific class of the SP classification) of monitorable properties in this its classical definition. It is possible to determine whether the property is monitorable by a syntactic analysis of the automaton states.

Definition 11 (Monitorability (automata view)). *The r-property* Π *recognized by the Streett m-automaton* $\mathcal{A}_\Pi = (Q^{\mathcal{A}_\Pi}, {q_{\text{init}}}^{\mathcal{A}_\Pi}, \rightarrow_{\mathcal{A}_\Pi}, \{(R_1, P_1), \ldots, (R_m, P_m)\})$ *is*

- $MP(\mathbb{B}_2^\bot)$*-monitorable iff*
  $\forall q \in Q^{\mathcal{A}_\Pi}, q_{\text{init}} \rightarrow_{\mathcal{A}_\Pi}^* q \Rightarrow \exists q' \in Bad^{\mathcal{A}_\Pi}, q \rightarrow_{\mathcal{A}_\Pi}^* q'$
- $MP(\mathbb{B}_2^\top)$*-monitorable iff*
  $\forall q \in Q^{\mathcal{A}_\Pi}, q_{\text{init}} \rightarrow_{\mathcal{A}_\Pi}^* q \Rightarrow \exists q' \in Good^{\mathcal{A}_\Pi}, q \rightarrow_{\mathcal{A}_\Pi}^* q'$
- $MP(\mathbb{B}_3)$*-monitorable iff*
  $\forall q \in Q^{\mathcal{A}_\Pi}, q_{\text{init}} \rightarrow_{\mathcal{A}_\Pi}^* q \Rightarrow \exists q' \in Bad^{\mathcal{A}_\Pi} \cup Good^{\mathcal{A}_\Pi}, q \rightarrow_{\mathcal{A}_\Pi}^* q'$

7.3 Verification and enforcement monitor synthesis

A monitor is a procedure consuming events fed by an underlying program and producing an appraisal in the current state depending on the sequence read so far. Considered monitors are deterministic finite-state machines producing an output in a relevant domain. This domain will be refined for special-purpose monitors (verification and enforcement). For verification monitors, this output function gives a truth-value (a verdict) in $\mathbb{B}_4$ regarding the evaluation of the current sequence relatively to the desired property. For enforcement monitors (EMs), this output function gives an enforcement operation inducing a modification on the input sequence so as to enforce the desired property.

Definition 12 (Monitor). *A* 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 a set of events* Σ*. The finite set* $Q^{\mathcal{A}}$ *denotes the control states and* ${q_{\text{init}}}^{\mathcal{A}} \in Q^{\mathcal{A}}$ *is the initial state. The complete function* $\longrightarrow_{\mathcal{A}}: Q^{\mathcal{A}} \times \Sigma \to Q^{\mathcal{A}}$ *is the transition function. In the following we abbreviate* $\longrightarrow_{\mathcal{A}} (q, a) = q'$ *by* $q \xrightarrow{a}_{\mathcal{A}} q'$*. The set of values* $X^{\mathcal{A}}$ *depends on the purpose of the monitor (verification or enforcement). The function* $\Gamma^{\mathcal{A}} : Q^{\mathcal{A}} \to X^{\mathcal{A}}$ *is an output function, producing values in* $X^{\mathcal{A}}$ *from states.*

Starting from this general definition of monitor, it is possible to synthesize dedicated monitors for runtime verification and enforcement. The synthesis are based on the definition of $\mathbb{P}$. For example, a verification monitor outputs a $\top_p$ when the current sequence presumably satisfies the property, *i.e.* when the run of the monitor reaches a state in $Good_p^{\mathcal{A}}$ in the corresponding Streett automaton $\mathcal{A}$. An enforcement monitor produces a *store* operation, when the current sequence does not satisfy the property and this execution sequence still has some "good" continuations (at least one). It switches off (*off* operation), when the run reaches a *Good* state. More details are given in [17]

Furthermore, in [17], we explain in details the synthesis procedure. Also, we formally define the notion of property verification and enforcement. For runtime verification, we show how an execution sequence is processed and verified by a verification monitor. For runtime enforcement, we describe how a given input sequence is transformed using the enforcement operations produced by the monitor. Besides, we prove that our synthesis procedures of verification and enforcement monitors are correct.

8 Conclusion and future works

Conclusion. We addressed the problem of monitorability and enforceability of properties at runtime using a general framework. In this framework, we characterized the sets of monitorable and enforceable properties in a unified way. We introduced a new definition of monitorability based on distinguishability of good and bad execution sequences. This definition is weaker than the classical one (based on positive and negative determinacy) and we believe that it better corresponds to practical needs and tool implementations. Fig. 1 summarizes

the results of this paper, depicting the set of monitorable and enforceable properties wrt. the SP classification. Furthermore, we have given general synthesis procedures to generate runtime and enforcement monitors in this framework.

Future works. The proposed approach raises new research perspectives and open questions. First, it seems interesting to consider this approach in the *testing* perspective. A monitor (passively) observes the execution of the program. Notably it has no control on the produced events and their sequencing. In a testing context, the notion of controllable event is introduced. An interesting issue would be to characterize the set of testable properties in the SP framework. Note that the classical definition of monitoring could be rather appropriate in this context. An additional issue to take into consideration is to deal with a reduced observability on the system under scrutiny. In practical situations, the desired property may refer to events out of the observation scope of a monitor. Similarly, it seems interesting to see how it is possible to characterize the space of properties for which others runtime-verification derived techniques can be applied (*e.g.* runtime reflection [23]). Another research perspective is to add expressiveness to EMs. Such augmented enforcers may enjoy more handling abilities on the sequences produced by the program. It seems interesting to see the impact on the set of enforceable properties. Also it seems relevant to study and compare complexity of the proposed monitors, notably with monitors defined in [3] for RV-LTL. To the authors' knowledge these are the only (runtime) monitors endowed with a 4-valued truth-domain.

Acknowledgement. The authors would like to thank Susanne Graf, Yassine Lakhnech, and the referees for their helpful remarks.

References

1. Runtime Verification. http://www.runtime-verification.org (2001-2009)
2. Pnueli, A., Zaks, A.: PSL Model Checking and Run-Time Verification Via Testers. In: FM. (2006) 573–586
3. Bauer, A., Leucker, M., Schallhart, C.: Comparing LTL semantics for runtime verification. Journal of Logic and Computation (2008) accepted for publication.
4. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. Technical Report TUM-I0724, Institut für Informatik, Technische Universität München (2007)
5. Havelund, K., Goldberg, A.: Verify your runs. In: Verified Software: Theories, Tools, Experiments: First IFIP TC 2/WG 2.3 Conference, VSTTE 2005, Zurich, Switzerland, October 10-13, 2005, Revised Selected Papers and Discussions, Berlin, Heidelberg, Springer-Verlag (2008) 374–383
6. Roşu, G., Chen, F., Ball, T.: Synthesizing monitors for safety properties – this time with calls and returns –. In: Workshop on Runtime Verification (RV'08). Volume 5289 of Lecture Notes in Computer Science., Springer (2008) 51–68
7. Havelund, K., Rosu, G.: Efficient monitoring of safety properties. Software Tools and Technology Transfer (2002)

8. d'Amorim, M., Roşu, G.: Efficient monitoring of ω-languages. In: Proceedings of 17th International Conference on Computer-aided Verification (CAV'05). Volume 3576 of Lecture Notes in Computer Science., Springer (2005) 364 – 378
9. Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. **3** (2000) 30–50
10. Hamlen, K.W., Morrisett, G., Schneider, F.B.: Computability classes for enforcement mechanisms. ACM Trans. Program. Lang. Syst. **28** (2006) 175–205
11. Viswanathan, M.: Foundations for the run-time analysis of software systems. PhD thesis, University of Pennsylvania, Philadelphia, PA, USA (2000)
12. Ligatti, J., Bauer, L., Walker, D.: Run-time enforcement of nonsafety policies. ACM Transactions on Information and System Security **12** (2009) 1–41
13. Manna, Z., Pnueli, A.: A hierarchy of temporal properties (invited paper, 1989). In: PODC '90: Proceedings of the ninth annual ACM symposium on Principles of distributed computing, New York, NY, USA, ACM (1990) 377–410
14. Chang, E.Y., Manna, Z., Pnueli, A.: Characterization of temporal property classes. In: Automata, Languages and Programming. (1992) 474–486
15. Falcone, Y., Fernandez, J.C., Mounier, L.: Synthesizing Enforcement Monitors wrt. the Safety-Progress Classification of Properties. In: ICISS '08: Proceedings of the 4th International Conference on Information Systems Security, Berlin, Heidelberg, Springer-Verlag (2008) 41–55
16. Falcone, Y., Fernandez, J.C., Mounier, L.: Enforcement Monitoring wrt. the Safety-Progress Classification of Properties. In: SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing, New York, NY, USA, ACM (2009) 593–600
17. Falcone, Y., Fernandez, J.C., Mounier, L.: Runtime Verification of Safety-Progress Properties. Technical Report TR-2009-6, Verimag Research Report (2009)
18. Kupferman, O., Y. Vardi, M.: Model checking of safety properties. Form. Methods Syst. Des. **19** (2001) 291–314
19. Lamport, L.: Proving the correctness of multiprocess programs. IEEE Trans. Softw. Eng. **3** (1977) 125–143
20. Alpern, B., Schneider, F.B.: Defining liveness. Technical report, Cornell University, Ithaca, NY, USA (1984)
21. Ligatti, J., Bauer, L., Walker, D.: Enforcing Non-safety Security Policies with Program Monitors. In: ESORICS. (2005) 355–373
22. Chen, F., Roşu, G.: MOP: An Efficient and Generic Runtime Verification Framework. In: Object-Oriented Programming, Systems, Languages and Applications(OOPSLA'07), ACM press (2007) 569–588
23. Leucker, M., Schallhart, C.: A brief account of runtime verification. Journal of Logic and Algebraic Programming **78** (2008) 293–303
24. Martinell, F., Matteucci, I.: Through modeling to synthesis of security automata. Electron. Notes Theor. Comput. Sci. **179** (2007) 31–46
25. Matteucci, I.: Automated synthesis of enforcing mechanisms for security properties in a timed setting. Electron. Notes Theor. Comput. Sci. **186** (2007) 101–120
26. Streett, R.S.: Propositional dynamic logic of looping and converse. In: STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing, New York, NY, USA, ACM (1981) 375–383
27. Falcone, Y., Fernandez, J.C., Mounier, L.: Specifying Properties for Runtime Verification in the Safety-Progress Classification. Technical Report TR-2009-5, Verimag Research Report (2009)
28. Tarjan, R.: Depth-first search and linear graph algorithms. SIAM Journal on Computing **1** (1972) 146–160