Int J Softw Tools Technol Transfer
DOI 10.1007/s10009-011-0196-8

RUNTIME VERIFICATION

What can you verify and enforce at runtime?

Yliès Falcone · Jean-Claude Fernandez · Laurent Mounier

© Springer-Verlag 2011

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 sets of properties for which runtime analysis approaches can be applied to. This paper presents a unified view of runtime verification and enforcement of properties in the Safety-Progress classification. First, we extend the Safety-Progress classification of properties in a runtime context. Second, 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. Finally, for the delineated sets of properties, we define specialized verification and enforcement monitors.

Keywords Runtime verification · Property monitoring · Property enforcement · Monitorable properties · Enforceable properties · Safety-Progress characterization

1 Introduction

In the past decades, we have seen the emergence of a world in which information systems are ubiquitous. System dissemination entails a growing need of confidence. System failures in history showed limits of existing engineering methodologies and enabled the emergence of *formal methods* [9]. Ideally, one would like to validate a program prior to its execution. However, static validation methods such as model-checking [12] suffer from limits preventing their use in real large-scale applications. For instance, those techniques are often bound to the design stage of a system, and hence, they are not shaped to face-off specification evolution. Even when those techniques (e.g., static analysis [10]) do scale well, they are limited by the properties they can check and may not be able to check interesting behavioral properties. Thus, the verification of some properties and elimination of some faults have to be complemented using methods relying on dynamic analysis. In this paper, we are interested in runtime verification and runtime enforcement. These methods, said to be incomplete, operate on *one* execution of the system. Acknowledging the loss of completeness enables to face-off the limitations of static validation methods.

Runtime-verification [3,4,21,32,34] 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. A possible approach for runtime verification consists in analyzing a run of the system under scrutiny in an incremental way using a decision procedure called a *monitor*. This monitor may be generated from a user-provided high-level specification (consisting in e.g., a property expressed by temporal logic formula or an automaton). The primary goal of this monitor is to detect violation or satisfaction with respect to the given specification. It can be viewed as 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

Y. Falcone (✉)
INRIA, Rennes, Bretagne Atlantique, Rennes, France
e-mail: Ylies.Falcone@inria.fr

J.-C. Fernandez · L. Mounier
Verimag, Université Grenoble I, Grenoble, France
e-mail: Jean-Claude.Fernandez@imag.fr

L. Mounier
e-mail: Laurent.Mounier@imag.fr

Published online: 21 April 2011

satisfaction or violation. Most research endeavors focused on monitoring safety properties (stating that *something bad can never happen*), as seen for example in [22,33]. Moreover, it has been shown by Viswanathan and Kim [37] that some computability constraints apply to runtime monitors. Considering the monitoring of safety properties, the violation detection mechanism used in the runtime device needs to be effective. Thus, for a safety property to be monitorable, it has to be co-recursively enumerable. However, the authors of [11] show that, when monitoring is purposed to detect violations of a property, safety properties are not the only monitorable properties. Recently, a new definition of monitorability was given by Pnueli and Zaks [32] where monitoring not only detects violations but also satisfactions, and it is proved in [3] that safety and co-safety properties represent only a proper subset of the set 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 [35] on the so-called *security automata*. In this work, the enforcement monitor watches the current execution sequence and halts the underlying program whenever it deviates from the desired property. Schneider announced that security automata are able to enforce the whole class of safety properties. The results in [37], previously mentioned in Sect. 1, that impose computability constraints on monitors also apply to security automata. Thus, the known class of enforceable properties with security automata was refined into the class of co-recursively enumerable properties. Later, Hamlen et al. [20] addressed the question of determining, in general, the class of properties enforceable on programs seen as Turing machines. The authors showed that enforcement at runtime can be addressed for co-recursively enumerable properties.

More recently, Ligatti et al. [28] showed that it is possible to enforce at runtime more than safety properties. Using more powerful enforcement mechanisms 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.

Motivations and contributions Based on the amount of works published and existing tools now available within the runtime-validation community, it appears that this technique progressed a lot in the last decade and seems now mature enough to address concrete industrial challenges. However, some interesting questions remain about its expressiveness. More precisely, the main questions we consider in this work are as follows: what are the classes of properties that can be handled at runtime, and is there a distinct answer for runtime verification and runtime enforcement? These questions are not original in themselves, but we propose here to address them within a unified framework: the *Safety-Progress* (SP) classification of properties [6,29]. The contributions of the paper are then as follows:

1. to propose a suitable framework for specifying and reasoning about properties in a runtime context;
2. to integrate and improve within this framework some existing expressiveness results related to runtime monitoring [3,4,32], and to propose an alternative definition of the notion of *monitorability*, leveraging the semantics of finite execution sequences;
3. and to improve some recent results related to property enforcement [15,16], giving a more accurate classification of enforceable properties;
4. to get a generic monitor synthesis technique, allowing to produce either a verification or an enforcement monitor from the same property description.

Let us illustrate a bit more the second motivation. Consider a system on which it is possible to evaluate two atomic propositions called p and q. At system runtime, system events are fed to a monitor. Each event is a pair containing the truth-values of p and q. Now let us consider the following requirement: "Either p is always true or q is eventually true". This means that, for the observed sequence of events, either p is evaluated to true on every event, or there exists an event on which q is evaluated to true. Now consider the two following possible executions of the system, represented by their sequences of events of length 2:

- $\{p, \overline{q}\} \cdot \{p, \overline{q}\}$: on both events p is true, q is false;
- $\{p, \overline{q}\} \cdot \{\overline{p}, \overline{q}\}$: on the first event p is true and q is false, on the second event p and q are false.

After observation of the first sequence of events, one can reasonably state that the property is "currently" true. Thus, if the program execution stops after this observation, the requirement is satisfied. Indeed, p has been always true during the program execution. Conversely, after observing the second sequence of events, one can reasonably state that the property is "currently" false. Indeed, the last observed event does not fulfill the requirement (neither p nor q evaluate to true).

We will see in Sect. 5 that this kind of property is monitorable according to the classical definition of monitorability. Moreover, a monitor built following this definition of monitorability would produce the *same verdict* for those two

sequences, namely a *do not know* verdict. This situation is undesirable from our point of view. Thus, we will propose an alternative definition of monitorability able to better cope with these kinds of properties and to give more precise verdicts.

This paper is a revised and extended version of [17] which appeared in the 9th international workshop on runtime verification. This new version brings the following additional contributions. First, it contains a more comprehensive theoretical basis by revisiting and extending results about the Safety-Progress classification of properties. Moreover, we provide additional results on monitorability. Furthermore, the synthesis of verification and enforcement monitors is given with full details (it was previously sketched). Finally, the presentation has been improved by means of additional examples, corrected results, and complete proofs.

Paper Organization The remainder of this article is organized as follows. First, Sect. 2 introduces some preliminary notations used throughout this paper, and Sect. 3 overviews related work on the issues we address. In Sect. 4, we propose an extension of the Safety-Progress classification of properties in a runtime verification context. Section 5 is dedicated to runtime monitoring, whereas Sect. 6 is dedicated to runtime enforcement. In both sections, we provide some characterizations of the classes of properties that can be handled by these techniques, with respect to the Safety-Progress framework. Then, in Sect. 7, we show how to obtain runtime verification and enforcement monitors for the delineated sets of properties. Finally, we give some concluding remarks and future work in Sect. 8.

In order to facilitate the reading of this article, some of the proofs have been sketched. Complete proofs can be found in Appendix.

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

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 $\sigma : I \to E$ 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 $E^\infty \stackrel{\text{def}}{=} E^* \cup E^\omega$ 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 a finite sequence $\sigma \in E^*$, we may use $last(\sigma)$ to denote the last element of σ, i.e., $\sigma_{|\sigma|-1}$. 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'. The sequence σ is said to be a strict prefix of $\sigma' \in E^\infty$ when $\forall i \in [0, |\sigma| - 1], \sigma_i = \sigma'_i$ 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 $n \in \mathbb{N}$, $\sigma_{...n}$ is the subsequence containing the $n+1$ first elements of σ. Moreover, when $|\sigma| > n$, the subsequence $\sigma_{n...}$ is the sequence containing all elements of σ but the n first ones. The set of prefixes $pref(\sigma)$ of a sequence $\sigma \in E^\infty$ is defined as follows. If $\sigma \in E^*$, then $pref(\sigma) \stackrel{\text{def}}{=} \{\sigma' \in E^* \mid \sigma' \preceq \sigma\}$. If $\sigma \in E^\omega$, then $pref(\sigma) \stackrel{\text{def}}{=} \{\sigma' \in E^* \mid \sigma' \prec \sigma\}$. The set $Pref(X)$ of prefixes of a set of sequences X is the union of the sets of prefixes of X-sequences: $Pref(X) \stackrel{\text{def}}{=} \bigcup_{\sigma \in X} pref(\sigma)$. The set $Pref(X, \sigma)$ of prefixes of a set of sequences X which are also strict prefixes of a sequence $\sigma \in \Sigma^\infty$ is: $Pref_\prec(X, \sigma) \stackrel{\text{def}}{=} Pref(X) \cap pref(\sigma) \setminus \{\sigma\}$. The σ-continuations, i.e., the continuations of a sequence σ, are the finite and infinite sequences belonging to the set $\{\sigma' \in E^\infty \mid \sigma \prec \sigma'\}$. For $\sigma' \in E^\infty$ a σ-continuation, if $\sigma' = \sigma \cdot \sigma''$, then $\sigma'' \in E^\infty$ is called an extension of σ.

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 resources, 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. These events may also be fed from the underlying program and contain the evaluation of some propositions of the system under scrutiny. We abstract these operations by a finite set of *events*, namely an alphabet Σ. We denote by $\mathcal{P}_\Sigma$ a program for which the alphabet 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*, i.e., $\forall \sigma \in Exec(\mathcal{P}_\Sigma), \forall \sigma' \in \Sigma^*, \sigma' \preceq \sigma \Rightarrow \sigma' \in Exec(\mathcal{P}_\Sigma)$. In the remainder of this article, we use an alphabet 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. $\sigma \in \varphi$, denoted $\varphi(\sigma)$, $\sigma \in \theta$, 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 power-sets of execution sequences (like fairness, for instance).

Runtime properties In this paper, we will focus on properties to be evaluated at *runtime*. As stated in Sect. 1, this means that we would have to consider finite and infinite execution sequences (that a program may produce). A runtime verification technique should address both kinds of sequences in a uniform way. Hence, we introduce a notion of "runtime property" (*r*-property) as a pair of finite/infinite execution sequence sets[2]:

Definition 1 (*runtime properties*) An *r*-property is a pair $(\phi, \varphi) \subseteq \Sigma^* \times \Sigma^\omega$. The property ϕ is called the finitary part of the *r*-property, whereas φ is called the infinitary part of the *r*-property.

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. Notations for *r*-properties follow from the notations for finitary and infinitary properties. For an *r*-property (ϕ, φ), its negation, noted $\overline{(\phi, \varphi)}$, is defined as $(\Sigma^* \setminus \phi, \Sigma^\omega \setminus \varphi)$. Boolean combinations of *r*-properties are defined in a natural way: $(\phi_1, \varphi_1) \vee (\phi_2, \varphi_2) = (\phi_1 \cup \phi_2, \varphi_1 \cup \varphi_2)$, and $(\phi_1, \varphi_1) \wedge (\phi_2, \varphi_2) = (\phi_1 \cap \phi_2, \varphi_1 \cap \varphi_2)$. Considering an execution sequence $\sigma \in Exec(\mathcal{P}_\Sigma)$, we say that σ satisfies (ϕ, φ) when $\sigma \in \Sigma^* \wedge \phi(\sigma) \vee \sigma \in \Sigma^\omega \wedge \varphi(\sigma)$. For an *r*-property $\Pi = (\phi, \varphi)$, we note $\Pi(\sigma)$ (resp. $\neg\Pi(\sigma)$) when σ satisfies (resp. does not satisfy) (ϕ, φ). The set of prefixes of an *r*-property $\Pi = (\phi, \varphi)$ is defined as: $Pref(\Pi) = Pref(\phi) \cup Pref(\varphi)$. Intersection between finitary, infinitary properties and *r*-properties is straightforward and denoted using operator $\sqcap$, e.g., $\Sigma^* \sqcap (\phi, \varphi) = \phi$.

Evaluation of r-properties Monitorability, enforceability, and monitor synthesis are based on the evaluation of *r*-properties by a monitor. Evaluating an execution sequence σ w.r.t. an *r*-property consists in producing a verdict regarding the current property-satisfaction of σ or future satisfactions of the possible σ-continuations. As a matter of fact, the verdicts produced by monitors are not necessarily 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. Examples of truth-domains are the classical Boolean domain $\{true, false\}$ or the real-number interval $[0, 1]$, or any relevant set of values used for evaluating properties. Considering a truth-domain $\mathbb{B}$, an

Fig. 1 Principle of runtime verification

r-property Π and a finite execution sequence σ, the evaluation of $\sigma \in \Sigma^*$ w.r.t. Π in $\mathbb{B}$, noted $[\![\Pi]\!]_{\mathbb{B}}(\sigma)$, is an element of $\mathbb{B}$ depending on $\Pi(\sigma)$ and satisfaction of σ-continuations w.r.t. Π.

The sets of monitorable and enforceable properties (Sects. 5, 6) rely both upon the truth-domain and evaluation function we consider.

3 Related work

This section overviews some related work on the topics we will discuss in this paper.[3] In particular, we summarize the basic concepts used for runtime verification and runtime enforcement, and we recall the existing results in terms of sets of properties that can be addressed by each of these techniques.

3.1 Runtime verification

Basic concepts As stated in Sect. 1, the notion of runtime verification can be formalized by a *verification monitor* (see Fig. 1) whose behavior consists in translating a sequence of events $\sigma \in \Sigma^\infty$ into a sequence of verdicts $\omega \in \mathbb{B}^\infty$, where $\mathbb{B}$ is a given truth-domain. This monitor is defined with respect to an *r*-property Π, and the sequence of verdicts ω is expected to give some information on the evaluation of Π on σ with respect to $\mathbb{B}$. Thus, one of the problems to be addressed is that each evaluation $[\![\Pi]\!]_{\mathbb{B}}(\sigma_{\cdots n}) = \omega_{\cdots n}$ of a *finite* sequence should not only give some relevant information on $\Pi(\sigma_{\cdots n})$, but also possibly on $\Pi(\sigma)$. In this context, several notions of *monitorability* were proposed in the past, and we review below the most important results.

Monitorability in the sense of [37] Viswanathan and Kim gave the first characterization of monitorable properties in [37]. Monitorable properties were characterized as a strict subset of safety properties defined over infinite sequences. The authors show that, due to the undecidability of some problems, a verification monitor is limited by some computability constraints. Moreover, this definition of monitorability was specifically defined for the detection of errors. Thus, the mechanism used for the error detection needs to be effective. Consequently, a property $\varphi \subseteq \Sigma^\omega$ was defined

[1] This is the distinction, made by Schneider [35], 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).

[2] Using a pair of sets makes the distinction between the finitary and infinitary parts of the property more explicit.

[3] The interested reader may consult [21] (resp. [13]) for more information on runtime verification (resp. runtime enforcement).

to be monitorable if it is a safety property and $\Sigma^* \setminus \mathit{Pref}(\varphi)$ is recursively enumerable. The authors establish the equality between this set of properties and the class Π_1^0 of the arithmetical hierarchy which is the class of co-recursively enumerable properties.

Monitorability in the sense of [32] Pnueli et al. gave a more general notion of monitorable properties relying on the notion of verdict determinacy by a *finite* 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 *each of its possible continuations* does not satisfy (resp. does satisfy) θ. Then θ is σ-monitorable if σ has a continuation s.t. θ is negatively or positively determined by this continuation. Finally, θ is monitorable if it is σ-monitorable for every $\sigma \in \Sigma^*$. The idea is that it becomes unnecessary to continue the execution of a θ-monitor after reading σ if θ is not σ-monitorable.

In Sect. 5, we give the corresponding formal definition in the context of *r*-properties.

Monitorability in the sense of [3] Bauer et al., inspired from Pnueli's definition of monitorable properties, proposed a slightly different one based on the notion of good and bad prefix introduced in model-checking by Kupferman and Vardi [24]. The intuitive idea is that with monitorable properties it is possible to "detect" a violation or satisfaction of infinitary properties with finite sequences. More precisely, the definition of monitorable properties comes in the following way. 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 $\forall w \in \Sigma^\omega, \neg\varphi(\sigma \cdot w)$ (resp. $\forall w \in \Sigma^\omega, \varphi(\sigma \cdot w)$). Then a prefix σ is said to be ugly if it does neither have good nor bad continuation, i.e., $\neg\exists v \in \Sigma^\omega, \mathit{bad\_prefix}(\sigma \cdot v, \varphi) \vee \mathit{good\_prefix}(\sigma \cdot v, \varphi)$. Finally, a property is said to be monitorable if it has no ugly prefix, formally: $\forall\sigma \in \Sigma^*, \exists v \in \Sigma^*, \mathit{bad\_prefix}(\sigma \cdot v, \varphi) \vee \mathit{good\_prefix}(\sigma \cdot v, \varphi)$.

About previous characterizations of monitorable properties The first characterization of monitorable properties given in [37] may seem arbitrary. It characterizes the class of monitorable properties directly as a class of properties.

However, let us remark that, in this definition a monitor is dedicated to the detection of "bad behaviors" from finite observations. It seems reasonable that a verification monitor is used to detect "good" behaviors as well, e.g., the satisfaction of a desired property. This is actually the idea behind the definition given in [32]. The last definition, given in [3], is equivalent to the previous one on Σ^ω. We will refer to the definition given in [32] as the *classical definition* as it was enunciated before the definition in [3]. Furthermore, Bauer et al. have shown that, according to this definition, the set of monitorable properties is a strict superset of safety and co-safety properties. These classes of properties are taken from the classical Safety-Liveness classification of properties [1,25]. 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) w.r.t. the SP classification (see Example 5 in Sect. 5).

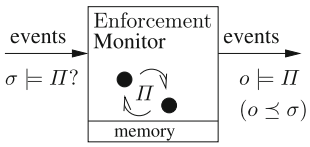

Fig. 2 Principle of runtime enforcement

3.2 Runtime enforcement

Basic concepts In *runtime enforcement*, the purpose of an *enforcement monitor* (see Fig. 2) used at runtime is to transform an input sequence $\sigma \in \Sigma^\infty$ into an output sequence $o \in \Sigma^\infty$ with respect to an *r*-property Π. The expected constraints on o are (usually) as follows:

> *soundness*: o should be a *correct* execution sequence, i.e., Π should evaluate to true on o;
> *transparency*: the enforcement operation should preserve as much as possible the initial program behavior by modifying the input sequence *in a minimal way*. A possible interpretation is that when σ does not satisfy Π then o should be the *longest correct prefix* of σ.

According to this definition, the set of properties that can be enforced at runtime clearly depends on the capabilities of the enforcement mechanism. For this purpose, the authors of [20] proposed a very general classification of enforceable properties: a program is viewed as a Turing machine, and the enforcement mechanisms they considered were based respectively on static analysis, program rewriting and runtime enforcement monitors. Other research efforts [16,27,28,35,37] focused more specifically on runtime enforcement monitors and proposed a characterization of enforceable properties in this context. We summarize these results below.

Security automata and co-recursively enumerable safety properties Schneider introduced security automata (a variant of Büchi automata) as the first runtime mechanism for enforcing properties in [35]. The announced set of enforceable properties with this kind of security automata is the set of safety properties. Then Schneider, Hamlen, and Morrisett refined the set of enforceable properties and showed that these security automata were actually restrained by the computational limits exhibited by Viswanathan and Kim in [37]. Hence, Schneider, Hamlen, and Morrisett showed that the

set of co-recursively enumerable safety properties is a strict upper limit of the power of (execution) enforcement monitors defined as security automata [20].

Edit-automata and infinite renewal properties Ligatti et al. [27,28] introduced *edit-automata* as runtime monitors. Depending on the current input and its control state, an edit-automaton 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). More precisely, a property θ is said to be an infinite renewal property if each valid infinite sequence σ has an infinite number of valid prefixes:

$$\forall \sigma \in \Sigma^{\infty}, \theta(\sigma) \Rightarrow \forall \sigma' \in \Sigma^{*}, \sigma' \prec \sigma \Rightarrow \exists \sigma'' \in \Sigma^{*}, \sigma' \preceq \sigma'' \prec \sigma \wedge \theta(\sigma'').$$

Shallow history automata and an information-based lattice of enforceable policies Fong [19] studied the effect of restraining the capacity of the runtime execution monitor and the effect on the enforcement power. Shallow history automata (SHA) keep as history a set of access events the underlying program made. Fong showed that these automata can enforce a set of properties strictly contained in the set of properties enforceable by Schneider's automata. The result has been generalized by using abstraction mechanisms on an equivalent variant of Schneider's automata. It raised up an information-based lattice of enforceable policies. At the top of this lattice is the set of properties enforceable by security automata (SHA keeps history of all events). At the bottom of this lattice is the set of policies prohibiting a set of events (SHA does not distinguish between prefixes of execution sequences made of the same events).

Fong's classification has a practical interest in the sense that it studies the effect of practical programming constraint (limited memory). It also shows that some classical security policies remain enforceable using such shallow automata.

Generic runtime enforcers and response properties In previous work [16,18], we introduced a generic notion of enforcement monitor encompassing previous mechanisms and gave a lower-bound on the set of properties they can enforce in the Safety-Progress classification (see Sect. 4). In this paper, we will show that this bound is tight. Furthermore, in [18], we have studied the question of enforcement monitor composition.

3.3 Synthesis of monitors

We give a short overview of the works related to the synthesis of monitors. An exhaustive list of works on monitor synthesis is far beyond the scope of this paper. We refer to [21,26,34] (resp. [13]) for more information on this topic in runtime verification (resp. runtime enforcement).

For runtime verification Generally, runtime verification monitors are generated from LTL-based specifications, as seen recently in [3,7]. Alternatively, ω-regular expressions have been used as a basis for generating monitors, as for example in [11].

For runtime enforcement In [30], Martinelli and Matteucci tackle the synthesis of enforcement mechanisms as defined by Ligatti. More generally, the authors consider security automata and edit-automata. The monitor is modeled 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 [31], Matteucci extends the approach in the context of real-time systems. In [15,18], 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. In this paper, we will provide a unified class-independent transformation.

4 The SP classification in a runtime context

This section recalls and extends some results about the Safety-Progress classification of properties [5,6,29]. In the original papers, this classification introduced a hierarchy between *regular* properties[4] defined as sets of *infinite* execution sequences. We extend the classification with finite-length execution sequences in a conservative way.

4.1 Informal description

The Safety-Progress classification is made of four basic classes over execution sequences. Informally, the classes were defined as follows:

- *safety* properties are the properties for which whenever a sequence satisfies a property, *all its prefixes* satisfy this property;
- *guarantee* properties are the properties for which whenever a sequence satisfies a property, *there are some prefixes* (at least one) satisfying this property;
- *response* properties are the properties for which whenever a sequence satisfies a property, *an infinite number of its prefixes* satisfy this property;

[4] In the following, the term property will stand for regular property.

– *persistence* properties are the properties for which whenever a sequence satisfies a property, *all but finitely many* of its prefixes satisfy this property; i.e., a finite number of its prefixes does not satisfy the property.

Furthermore, two extra classes can be defined as finite Boolean combinations (union and intersection) of basic classes.

– The *obligation class* can be defined as the class obtained by Boolean combinations of safety and guarantee properties.
– The *reactivity class* can be defined as the class obtained by Boolean combinations of response and persistence properties. This is the most general class containing all linear temporal properties [5].

An *r*-property of a given class is said to be *pure* when it is a property of none of the other sub-classes.

The Safety-Progress classification is an alternative to the more classical Safety-Liveness dichotomy [1,25]. Unlike this later, the Safety-Progress classification is a hierarchy and not a partition. It provides a finer-grain classification, and the properties of each class are characterized according to four *views* [5]: a language-theoretic view, a topological view, a temporal logic view, and an automata-based view. The language-theoretic view describes the hierarchy according to the way each class can be constructed from sets of finite sequences. The topological view characterizes the classes as sets with topological properties. The third vision links the classes to their expression in temporal logic. At last, the automata view gives syntactic characterizations on automata recognizing properties of a given class. We will consider here only the language-theoretic and the automata views dedicated to *r*-properties.

4.2 The language-theoretic view of *r*-properties

4.2.1 Construction of r-properties

The language-theoretic view of the Safety-Progress classification is based on the construction of infinitary properties and finitary properties from finitary ones. It relies on the use of four operators A, E, R, P (building infinitary properties) and four operators A_f, E_f, R_f, P_f (building finitary properties) applying to finitary properties. In the original classification of Manna and Pnueli, the operators A, E, R, P, A_f, E_f were introduced. In this paper, we add the operators R_f and P_f and give a formal definition of all operators. In these definitions, ψ is a finitary property over Σ.

Definition 2 (*Operators A, E, R, P*)

– $A(\psi) = \{\sigma \in \Sigma^\omega \mid \forall \sigma' \in \Sigma^*, \sigma' \prec \sigma \Rightarrow \psi(\sigma')\}$.
– $E(\psi) = \{\sigma \in \Sigma^\omega \mid \exists \sigma' \in \Sigma^*, \sigma' \prec \sigma \wedge \psi(\sigma')\}$.
– $R(\psi) = \{\sigma \in \Sigma^\omega \mid \forall \sigma' \in \Sigma^*, \sigma' \prec \sigma \Rightarrow \exists \sigma'' \in \Sigma^*, \sigma' \prec \sigma'' \prec \sigma \wedge \psi(\sigma'')\}$.
– $P(\psi) = \{\sigma \in \Sigma^\omega \mid \exists \sigma' \in \Sigma^*, \forall \sigma'' \in \Sigma^*, \sigma' \prec \sigma'' \prec \sigma \Rightarrow \psi(\sigma'')\}$.

$A(\psi)$ consists of all infinite words σ s.t. *all* prefixes of σ belong to ψ. $E(\psi)$ consists of all infinite words σ s.t. *some* prefixes of σ belong to ψ. $R(\psi)$ consists of all infinite words σ s.t. *infinitely many* prefixes of σ belong to ψ. $P(\psi)$ consists of all infinite words σ s.t. *all but finitely many* prefixes of σ belong to ψ.

The operators A_f, E_f, R_f, P_f build finitary properties from finitary ones.

Definition 3 (*Operators A_f, E_f, R_f, P_f*)

– $A_f(\psi) = \{\sigma \in \Sigma^* \mid \forall \sigma' \in \Sigma^*, \sigma' \preceq \sigma \Rightarrow \psi(\sigma')\}$.
– $E_f(\psi) = \{\sigma \in \Sigma^* \mid \exists \sigma' \in \Sigma^*, \sigma' \preceq \sigma \wedge \psi(\sigma')\}$.
– $R_f(\psi) = \{\sigma \in \Sigma^* \mid \psi(\sigma) \wedge \forall n \in \mathbb{N}, \exists \sigma' \in \Sigma^*, \sigma \prec \sigma' \wedge |\sigma'| \geq n \wedge \psi(\sigma')\}$.
– $P_f(\psi) = \{\sigma \in \Sigma^* \mid \psi(\sigma) \wedge \exists \sigma' \in \Sigma^*, \sigma \preceq \sigma' \wedge \forall n \in \mathbb{N}, \exists \sigma'' \in \Sigma^*, |\sigma''| = n \wedge \psi(\sigma' \cdot \sigma'')\}$.

$A_f(\psi)$ consists of all finite words σ s.t. *all* prefixes of σ belong to ψ. One can observe that $A_f(\psi)$ is the largest prefix-closed subset of ψ. $E_f(\psi)$ consists of all finite words σ s.t. *some* prefixes of σ belong to ψ. One can observe that $E_f(\psi) = \psi \cdot \Sigma^*$. $R_f(\psi)$ consists of all finite words σ s.t. $\psi(\sigma)$ and there exists an infinite number of continuations σ' of σ also belonging to ψ. $P_f(\psi)$ consists of all finite words σ belonging to ψ s.t. there exists a continuation σ' of σ s.t. σ' persistently has continuations σ'' staying in ψ (i.e., $\sigma' \cdot \sigma''$ belongs to ψ).

Based on these operators, each class can be seen from the language-theoretic view.

Definition 4 $\Pi = (\phi, \varphi)$ is defined to be:

– A *safety* *r*-property if $\Pi = (A_f(\psi), A(\psi))$ for some finitary property ψ, i.e., all prefixes of a finite word $\sigma \in \phi$ or of an infinite word $\sigma \in \varphi$ belong to ψ.
– A *guarantee* *r*-property if $\Pi = (E_f(\psi), E(\psi))$ for some finitary property ψ, i.e., each finite word $\sigma \in \phi$ or infinite word $\sigma \in \varphi$ is guaranteed to have some prefixes (at least one) belonging to ψ.
– A *response* *r*-property if $\Pi = (R_f(\psi), R(\psi))$ for some finitary property ψ, i.e., each finite word $\sigma \in \phi$ or infinite word $\sigma \in \varphi$ recurrently has (infinitely many) prefixes belonging to ψ.
– A *persistence* *r*-property if $\Pi = (P_f(\psi), P(\psi))$ for some finitary property ψ, i.e., each finite word $\sigma \in \phi$ or infinite word $\sigma \in \varphi$ persistently has (continuously from a certain point on) prefixes belonging to ψ.

In all cases, we say that Π is built over ψ. Furthermore, obligation (resp. reactivity) r-properties are obtained by Boolean combinations of safety and guarantee (resp. response and persistence) r-properties.

Given a set of events Σ, Safety(Σ) (resp. Guarantee(Σ), Obligation(Σ), Response(Σ), Persistence(Σ)) designates the set of safety (resp. guarantee, obligation, response, persistence) r-properties defined over Σ.

We illustrate in the following example the construction of infinitary properties from finitary ones (described as regular expressions) for each of the four operators.

Example 1 (*Construction of infinitary and finitary properties from finitary ones—r-properties*) We consider a client–server application, with a set of observable events $\Sigma \subseteq \{r, g, d\}$ where r denotes a client request of a given resource and g (resp. d) denotes a corresponding grant (resp. deny) of this resource provided by the server.

- For the finitary property $\psi = \epsilon + r^+ \cdot g^*$, $A_f(\psi) = \epsilon + r^+ \cdot g^*$, $A(\psi) = r^\omega + r^+ \cdot g^\omega$, $\Pi_1 = (A_f(\psi), A(\psi))$ is a safety r-property. This language contains all the words that have either only occurrences of r or a finite number of occurrences of r (at least one) followed only by occurrences of g. According to this property, a resource should be requested at least once to be granted, and, when granted once, it should not be requested anymore.
- For the finitary property $\psi = r^+ \cdot g$, $E_f(\psi) = r^+ \cdot g \cdot \Sigma^*$, $E(\psi) = r^+ \cdot g \cdot \Sigma^\omega$, $\Pi_2 = (E_f(\psi), E(\psi))$ is a guarantee r-property. This property tells that the client will issue some requests and will receive a positive answer later on.
- For the finitary property $\psi = g + (r \cdot g)^*$, $R_f(\psi) = (r \cdot g)^*$, $R(\psi) = (r \cdot g)^\omega$, $\Pi_3 = (R_f(\psi), R(\psi))$ is a response r-property. This language contains all the words that have infinitely many occurrences of $r \cdot g$. This property tells that clients will repeatedly send requests and receive back a positive answer (the pattern $r \cdot g$ can be seen here as a *transaction*).
- For the finitary property $\psi = g + r \cdot g \cdot (r + r \cdot g)^*$, $P_f(\psi) = r \cdot g \cdot (r + r \cdot g)^*$, $P(\psi) = r \cdot g \cdot (r + r \cdot g)^\omega$, $\Pi_4 = (P_f(\psi), P(\psi))$ is a persistence r-property. This language contains all the words starting with $r \cdot g \cdot r$ and ending with occurrences of $r + r \cdot g$. According to this property, after a first granted resource, at some point this resource should be granted forever.

4.2.2 Some useful facts about the language view

Now, we give some useful facts about r-properties in the language view. Those facts will be used in the remainder when characterizing the set of monitorable properties.

Basic classes were defined in a constructive fashion. It is sometimes interesting to have a direct characterization for the properties of those classes. The following property gives a characterization for safety and guarantee r-properties. The proof is a direct adaptation of the proof given in [5].

Property 1 (*Characterization of safety and guarantee r-properties*) An r-property $\Pi = (\phi, \varphi)$ is

- a safety iff $\Pi = (A_f(Pref(\phi)), A(Pref(\varphi)))$,
- a guarantee iff $\Pi = \left(E_f(\overline{Pref(\overline{\phi})}), E(\overline{Pref(\overline{\varphi})})\right)$.

We expose the closure of safety and guarantee r-properties as a straightforward consequence of definitions of safety and guarantee r-properties.

Property 2 (*Closure of r-properties*) Considering an r-property $\Pi = (\phi, \varphi)$ defined over an alphabet Σ built from a finitary property ψ, the following facts hold:

1. If Π is a safety r-property, all prefixes of a sequence belonging to Π also belong to Π, i.e., $\forall \sigma \in \Sigma^\infty, \Pi(\sigma) \Rightarrow \forall \sigma' \prec \sigma, \Pi(\sigma')$.
2. If Π is a guarantee r-property, all continuations of a finite sequence belonging to Π also belong to Π, i.e., $\forall \sigma \in \Sigma^*, \Pi(\sigma) \Rightarrow \forall \sigma' \in \Sigma^\infty, \Pi(\sigma \cdot \sigma')$.

Proof The proof can be found in Appendix A.1.1; it uses the definitions of the operators A_f, A, E_f, E. □

The following lemma (inspired from [5]) provides a decomposition of each obligation properties in a normal form.

Lemma 1 *Any obligation r-property can be represented as the intersection*

$$\bigcap_{i=1}^{k} (\text{Safety}_i \cup \text{Guarantee}_i)$$

for some $k > 0$, where Safety$_i$ *and* Guarantee$_i$ *are, respectively, safety and guarantee r-properties. We refer to this presentation as the conjunctive normal form of obligation r-properties.*

When an r-property Π is expressed as $\cap_{i=1}^{k}(\text{Safety}_i \cup \text{Guarantee}_i)$, Π is said to be a k-obligation r-property. The set of k-obligation r-properties (with $k \geq 1$) is denoted k–Obligation(Σ). Similar definitions and properties hold for reactivity r-properties which are expressed by Boolean combinations of response and persistence r-properties.

4.3 The automata view of r-properties

For the automata view of the Safety-Progress classification, we follow [5] and define r-properties using Streett automata.

Furthermore, for each class of the Safety-Progress classification, it is possible to syntactically characterize a recognizing finite-state automaton. Moreover, we introduce transformations that take a deterministic finite-state automaton and a "modification pattern" so as to obtain a Streett automaton. These transformations are the representatives in the automata view of the operators defined in the language view.

4.3.1 Streett automata

We define a variant of deterministic and complete Streett automata (introduced in [36] and used in [5]) for property recognition.[5] These automata process events and decide properties of interest. We add to original Streett automata a finite-sequence recognizing criterion in such a way that these automata uniformly recognize r-properties.

Definition 5 (*Streett automaton*) A deterministic finite-state Streett automaton is a tuple $(Q, q_{\text{init}}, \Sigma, \longrightarrow, \{(R_1, P_1), \ldots, (R_m, P_m)\})$ 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 $\longrightarrow\colon Q \times \Sigma \to Q$ is the (complete) 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 $i \leq n$; $R_i \subseteq Q$ 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. Moreover, for $\sigma = \sigma_0 \cdots \sigma_{n-1}$ a word over Σ of length n and $q, q' \in Q^{\mathcal{A}}$ two states, we note $q \xrightarrow{\sigma} q'$ when $\exists q_1, \ldots, q_{n-2} \in Q^{\mathcal{A}}, q \xrightarrow{\sigma_0} q_1 \wedge \cdots \wedge q_{n-2} \xrightarrow{\sigma_{n-1}} q'$. In the following $\mathcal{A} = (Q^{\mathcal{A}}, q^{\mathcal{A}}_{\text{init}}, \Sigma, \longrightarrow_{\mathcal{A}}, \{(R_1, P_1), \ldots, (R_m, P_m)\})$ designates a deterministic finite state Streett m-automaton.

For $q \in Q^{\mathcal{A}}$, $Reach_{\mathcal{A}}(q)$ is the set of reachable states from q with at least one transition in $\mathcal{A}$ (denoted $Reach(q)$ when clear from context), i.e., $Reach_{\mathcal{A}}(q) \stackrel{\text{def}}{=} \{q' \in Q^{\mathcal{A}} \mid \exists \sigma \in \Sigma^+, q \xrightarrow{\sigma}_{\mathcal{A}} q'\}$. 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 $\forall i, (q_i \in Q^{\mathcal{A}} \wedge q_i \xrightarrow{\sigma_i}_{\mathcal{A}} q_{i+1}) \wedge q_0 = q^{\mathcal{A}}_{\text{init}}$. 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 $run(\sigma, \mathcal{A}) = q_0 \cdot q_1 \cdots$.

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

For a Streett automaton, the notion of acceptance condition is defined using the accepting pairs.

Definition 6 (*Acceptance condition for infinite sequences*) For $\sigma \in \Sigma^\omega$, we say that $\mathcal{A}$ accepts σ if $\forall i \in [1, m]$, $vinf(\sigma, \mathcal{A}) \cap R_i \neq \emptyset \vee vinf(\sigma, \mathcal{A}) \subseteq P_i$.

To deal with r-properties, we need to also define an acceptance criterion for *finite* sequences: a finite sequence is accepted by a Streett automaton if and only if it terminates on a distinguished state R_i or P_i for each accepting pair i.

Definition 7 (*Acceptance condition for finite sequences*) For a finite-length execution sequence $\sigma \in \Sigma^*$ s.t. $|\sigma| = n$, we say that the m-automaton $\mathcal{A}$ accepts σ if $(\exists q_0, \ldots, q_n \in Q^{\mathcal{A}}, run(\sigma, \mathcal{A}) = q_0 \cdots q_n \wedge q_0 = q^{\mathcal{A}}_{\text{init}}$ and $\forall i \in [1, m]$, $q_n \in P_i \cup R_i)$.

4.3.2 The hierarchy of automata

An interesting feature of Streett automata is that the class of properties they recognize can be easily characterized by some syntactic considerations.

- A *safety automaton* is a plain-automaton s.t. $R = \emptyset$, 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 s.t. $P = \emptyset$, 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 s.t. for each i in $[1, m]$:
  - there is no transition from $q \in \overline{P_i}$ to $q' \in P_i$,
  - there is no transition from $q \in R_i$ to $q' \in \overline{R_i}$.
- A *response automaton* is a plain-automaton s.t. $P = \emptyset$.
- A *persistence automaton* is a plain-automaton s.t. $R = \emptyset$.
- A *reactivity automaton* is any unrestricted automaton.

The syntactic restrictions are illustrated in Fig. 4: shapes of Streett automata for basic classes are depicted. One may remark that these syntactic restrictions hold for the automata represented in Fig. 3.

Automata and properties We now link Streett automata to r-properties.

Definition 8 (*Automata and r-properties*) We say that a Streett automaton $\mathcal{A}$ *defines* an r-property $(\phi, \varphi) \in 2^{\Sigma^* \times \Sigma^\omega}$ if and only if the set of finite (resp. infinite) execution sequences accepted by $\mathcal{A}$ is equal to ϕ (resp. φ), which is noted $\mathcal{L}(\mathcal{A}) = (\phi, \varphi)$. Conversely, an r-property $(\phi, \varphi) \in 2^{\Sigma^* \times \Sigma^\omega}$ is said to be *specifiable* by an automaton $\mathcal{A}$ if the set

[5] There exist several equivalent definitions of Streett automata dedicated to the recognition of infinite sequences. We choose here to follow the definition used in [5].

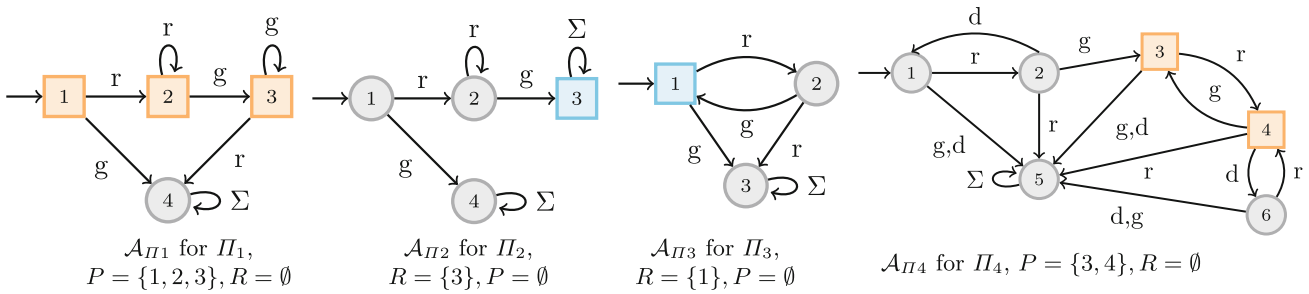

Fig. 3 Examples of Streett automata

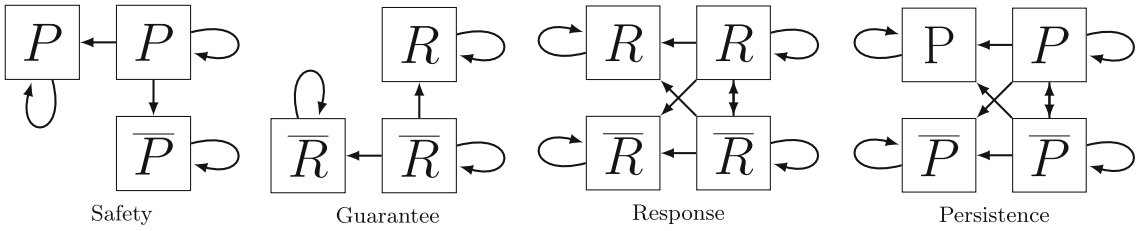

Fig. 4 Schematic illustrations of the shapes of Streett automata for basic classes

of finite (resp. infinite) execution sequences accepted by the automaton $\mathcal{A}$ is ϕ (resp. φ).

Example 2 (*Streett Automata*) In Fig. 3 are represented Streett plain-automata for the properties presented in Example 1.

- $\mathcal{A}_{\Pi 1}$ is a safety automaton; its set of recurrent states is empty; its set of persistent states is $P = \{1, 2, 3\}$. A finite sequence is accepted if its run ends in either states 1, 2 or 3, meaning that, if a grant happened there was at least one request previously. An infinite sequence is accepted if the only states visited infinitely often are states 1, 2, or 3, meaning that requests have been made, and they were followed by only grants.
- $\mathcal{A}_{\Pi 2}$ is a guarantee automaton; its set of persistent states is empty; its set of recurrent states is $R = \{3\}$. A finite sequence is accepted if its run ends in state 3. An infinite sequence is accepted if the state 3 is visited infinitely often. In both cases, it means that requests have been issued and then have been granted.
- $\mathcal{A}_{\Pi 3}$ is a response automaton; its set of persistent states is empty; its set of recurrent states is $R = \{1\}$. A finite sequence is accepted if its run ends in state 1, meaning that every request has been followed by a grant (in this order). An infinite sequence is accepted if it visits the state 1 infinitely often, meaning that this infinite sequence contains a succession of action sequences "one request followed by one grant".
- $\mathcal{A}_{\Pi 4}$ is a persistence automaton; its set of recurrent states is empty; its set of persistent states is $P = \{3, 4\}$. A finite sequence is accepted if its run ends in state 3 or 4, meaning that a first successful request has been made and, after that, the user performs only successful requests (if he makes a requests, this request is granted). An infinite sequence is accepted if it visits infinitely often only states 3 and 4, meaning that after a first successful request all user's requests have been granted.

In Sect. 4.5 we link the syntactic characterizations on the automata to the semantic characterization of the properties they specify.

4.3.3 From a DFA to a Streett automaton

We now introduce four transformations allowing to obtain a Streett automaton, given a deterministic finite-state automaton and a "pattern" for this underlying property. These patterns are inspired from the different classes of the Safety-Progress hierarchy. These simple transformations correspond, in the automata view, to the operators in the language view and the temporal modalities in the logical view.[6] We start by first defining those transformations and then prove their soundness.

A deterministic finite-state automaton (DFA) [23], is given relatively to an alphabet Σ and is here formally defined as a tuple $(Q, q_{\text{init}}, \longrightarrow, F)$, where Q is a finite set of states, $q_{\text{init}} \in Q$ is the initial state, $\longrightarrow : Q \times \Sigma \to Q$ is the transition function, and $F \subseteq Q$ is the set of accepting states.

Definitions of the transformations In the following definitions, $\mathcal{A}_\psi = (Q^{\mathcal{A}_\psi}, q_{\text{init}}^{\mathcal{A}_\psi}, \longrightarrow_{\mathcal{A}_\psi}, F^{\mathcal{A}_\psi})$ designates a

[6] i.e., operators A, E, R, P (and their finitary versions) of the language view and the temporal modalities $\Box$, $\Diamond$, $\Box\Diamond$, $\Diamond\Box$ of the logical view.

complete DFA recognizing a finitary regular property ψ. We define a transformation for each basic class of the hierarchy.

Synthesis of safety automata For this class of r-properties, the transformation is defined as follows:

Definition 9 (*DFA to Streett safety automaton*) The transformation of $\mathcal{A}_\psi$ into a Streett safety automaton is DFA2S_Saf $(\mathcal{A}_\psi) = (Q^{\mathcal{A}_\Pi}, q_{\text{init}}^{\mathcal{A}_\Pi}, \rightarrow_{\mathcal{A}_\Pi}, \{(\emptyset, P)\})$ and defined by:

- $Q^{\mathcal{A}_\Pi} = F^{\mathcal{A}_\psi} \cup \{sink\}$, where $sink \notin Q^{\mathcal{A}_\psi}$,
- $q_{\text{init}}^{\mathcal{A}_\Pi} = q_{\text{init}}^{\mathcal{A}_\psi}$ if $q_{\text{init}}^{\mathcal{A}_\psi} \in F^{\mathcal{A}_\psi}$, and $sink$ otherwise,
- $\rightarrow_{\mathcal{A}_\Pi}$ is defined as the smallest relation verifying:
  - $q \xrightarrow{e}_{\mathcal{A}_\Pi} q'$ if $q \in F^{\mathcal{A}_\psi} \wedge q' \in F^{\mathcal{A}_\psi} \wedge q \xrightarrow{e}_{\mathcal{A}_\psi} q'$ (TSAFE1),
  - $q \xrightarrow{e}_{\mathcal{A}_\Pi} sink$ if $\exists q' \in Q^{\mathcal{A}_\psi}, q' \notin F^{\mathcal{A}_\psi} \wedge q \xrightarrow{e}_{\mathcal{A}_\psi} q'$ (TSAFE2),
  - $\forall e \in \Sigma, sink \xrightarrow{e}_{\mathcal{A}_\Pi} sink$ (TSAFE3),
- $P = Reach_{\mathcal{A}_\Pi}(q_{\text{init}}^{\mathcal{A}_\Pi}) \setminus \{sink\}$, $(m = 1)$.

One can remark that the resulting automaton is indeed a Streett safety automaton since $R = \emptyset$, and there is no transition from the states in $\overline{P}$ to the states in P. This transformation adds a sink state and modifies the transition function in order to redispatch the transitions outgoing from accepting states to the sink state. Furthermore, the transitions outgoing from a non-accepting state have been removed. The set of persistent states is the set of accepting states of the DFA.

Moreover, according to the syntactic restrictions of the obtained Streett safety automata, the following property holds: for a sequence $\sigma \in \Sigma^\omega$ and an automaton resulting of the transformation $\mathcal{A}_\Pi$, if $sink \in vinf(\sigma, \mathcal{A}_\Pi)$, then $vinf(\sigma, \mathcal{A}_\Pi) = \{sink\}$; else $vinf(\sigma, \mathcal{A}_\Pi) \subseteq P$.

Synthesis of Streett guarantee automata For this class of r-properties, the transformation is defined as follows:

Definition 10 (*DFA to Streett guarantee automaton*) The transformation of $\mathcal{A}_\psi$ into a Streett guarantee automaton is DFA2S_Guar$(\mathcal{A}_\psi) = (Q^{\mathcal{A}_\Pi}, q_{\text{init}}^{\mathcal{A}_\Pi}, \rightarrow_{\mathcal{A}_\Pi}, \{(R, \emptyset)\})$ and defined by:

- $Q^{\mathcal{A}_\Pi}$ is the smallest subset of $Q^{\mathcal{A}_\psi}$ containing the reachable states from the initial state $q_{\text{init}}^{\mathcal{A}_\Pi}$ with $\longrightarrow_{\mathcal{A}_\Pi}$ (defined below),
- $q_{\text{init}}^{\mathcal{A}_\Pi} = q_{\text{init}}^{\mathcal{A}_\psi}$,
- $\rightarrow_{\mathcal{A}_\Pi}$ is defined as the smallest relation verifying:
  - $q \xrightarrow{e}_{\mathcal{A}_\Pi} q$ if $\exists q' \in Q^{\mathcal{A}_\psi}, q \xrightarrow{e}_{\mathcal{A}_\psi} q' \wedge q \in F^{\mathcal{A}_\psi}$ (TGUAR1),
  - $q \xrightarrow{e}_{\mathcal{A}_\Pi} q'$ if $q \notin F^{\mathcal{A}_\psi} \wedge q \xrightarrow{e}_{\mathcal{A}_\psi} q'$ (TGUAR2),
- $R = F^{\mathcal{A}_\psi}$, $(m = 1)$.

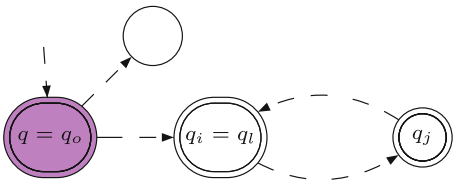

Fig. 5 Principle for tagging recurrent states in DFA2S_Res

One may remark that the resulting automaton is indeed a Streett guarantee automaton since $P = \emptyset$ and there is no transition from the R-states to the $\overline{R}$-states. This automaton may not be minimal regarding the number of R-states. They can be merged into one unique state since they are all equivalent w.r.t. property recognition. This transformation modifies the transition function in the following manner: outgoing transitions from the accepting states (to an accepting state or not) are modified into a loop on the same state. Indeed, when a run reaches a state in F, this suffix suffices in order to satisfy the guarantee property. The initial state is not modified, and the set of states of the Streett automaton is defined as the smallest set of reachable states from the initial state with the new transition function.

Synthesis of Streett response automata For this class of r-properties, the transformation is defined as follows:

Definition 11 (*DFA to Streett response automaton*) The transformation of $\mathcal{A}_\psi$ into a Streett response automaton is DFA2S_Resp$(\mathcal{A}_\psi) = (Q^{\mathcal{A}_\Pi}, q_{\text{init}}^{\mathcal{A}_\Pi}, \rightarrow_{\mathcal{A}_\Pi}, \{(R, \emptyset)\})$ and defined by:

- $Q^{\mathcal{A}_\Pi} = Q^{\mathcal{A}_\psi}$,
- $q_{\text{init}}^{\mathcal{A}_\Pi} = q_{\text{init}}^{\mathcal{A}_\psi}$,
- $\rightarrow_{\mathcal{A}_\Pi} = \rightarrow_{\mathcal{A}_\psi}$,
- $R = \{q \in F^{\mathcal{A}_\psi} \mid \exists l > 0, \exists q_0, \ldots, q_l \in Q^{\mathcal{A}_\psi}, \ (1) \wedge (2)\ \} \cup \{q \in F^{\mathcal{A}_\psi} \mid q \longrightarrow_{\mathcal{A}_\psi} q\}$, where

$$\forall j \in [0, l-1], q_j \longrightarrow_{\mathcal{A}_\psi} q_{j+1} \tag{1}$$

$$\exists i \in [0, l], \exists j \in [i, l-1], q_j \in F^{\mathcal{A}_\psi} \wedge q_i = q_l \wedge q_0 = q. \tag{2}$$

The resulting automaton is indeed a Streett response automaton since $P = \emptyset$. This transformation does neither modify the set of states nor the transition function. It marks as recurrent states (cf. Fig. 5) every accepting state of the DFA s.t. it is possible from this state to reach a cycle containing at least one accepting state.

Synthesis of Streett persistence automata For this class of r-properties, the transformation is defined as follows:

Definition 12 (*DFA to Streett persistence automaton*) The transformation of $\mathcal{A}_\psi$ into a Streett persistence automaton

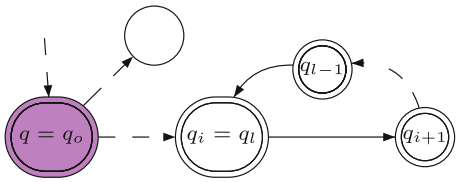

Fig. 6 Principle for tagging persistent states in DFA2S_Per

is DFA2S_Per$(\mathcal{A}_\psi) = (Q^{\mathcal{A}_\Pi}, q_{\text{init}}^{\mathcal{A}_\Pi}, \rightarrow_{\mathcal{A}_\Pi}, \{(\emptyset, P)\})$ and defined by:

- $Q^{\mathcal{A}_\Pi} = Q^{\mathcal{A}_\psi}$,
- $q_{\text{init}}^{\mathcal{A}_\Pi} = q_{\text{init}}^{\mathcal{A}_\psi}$,
- $\rightarrow_{\mathcal{A}_\Pi} = \rightarrow_{\mathcal{A}_\psi}$,
- $P = \{q \in F^{\mathcal{A}_\psi} \mid \exists l > 0, \exists q_0, \ldots, q_l \in Q^{\mathcal{A}_\psi}, \ (1) \wedge (3)\ \} \cup \{q \in F^{\mathcal{A}_\psi} \mid q \longrightarrow_{\mathcal{A}_\psi} q\}$, where

$$\exists i \in [0, l], \forall j \in [i, l-1], q_j \in F^{\mathcal{A}_\psi} \quad \wedge q_i = q_l \wedge q_0 = q. \tag{3}$$

The resulting automation is indeed a Streett persistence automaton since $R = \emptyset$. This transformation does neither modify the set of states nor the transition function. It marks (cf. Fig. 6) as persistent state every accepting state of the DFA from which it is possible to reach a cycle of accepting states.

Soundness of the transformations Given a finitary property ψ, defining a regular language over an alphabet Σ and specified by a DFA $\mathcal{A}_\psi$, the safety (resp. guarantee, response, persistence) r-property $(X_f(\psi), X(\psi))$ where $X \in \{A, E, R, P\}$ is specified by the Streett automaton obtained by the transformation *DFA2S* specific to safety (resp. guarantee, response, persistence) properties. This is stated formally by the following theorem:

Theorem 1 (Soundness of the transformations of DFAs to Streett automata) *The transformation DFA2S_Saf (resp. DFA2S_Guar, DFA2S_Resp, DFA2S_Per) in the automata view "corresponds" to the operator A_f and A (resp. E_f and E, R_f and R, P_f and P) in the language view. Formally, when $\mathcal{L}(\mathcal{A}_\psi) = \psi$,*

$$\begin{aligned}
\mathcal{A}_\Pi = \mathit{DFA2S_Saf}(\mathcal{A}_\psi) &\Rightarrow \mathcal{L}(\mathcal{A}_\Pi) &&= (A_f(\psi), A(\psi)) \\
\mathcal{A}_\Pi = \mathit{DFA2S_Guar}(\mathcal{A}_\psi) &\Rightarrow \mathcal{L}(\mathcal{A}_\Pi) &&= (E_f(\psi), E(\psi)) \\
\mathcal{A}_\Pi = \mathit{DFA2S_Resp}(\mathcal{A}_\psi) &\Rightarrow \mathcal{L}(\mathcal{A}_\Pi) &&= (R_f(\psi), R(\psi)) \\
\mathcal{A}_\Pi = \mathit{DFA2S_Per}(\mathcal{A}_\psi) &\Rightarrow \mathcal{L}(\mathcal{A}_\Pi) &&= (P_f(\psi), P(\psi))
\end{aligned}$$

Proof Proofs are conducted for each class of properties and the associated transformation by using the acceptance criteria and examining runs of accepted sequences. The complete proof can be found in Appendix A.1.2. □

Remark 1 Let us remark that these transformations may entail a loss of information. It is in general not possible to find again the finitary language from which a Streett automaton has been built. Consider for example the Streett guarantee automaton represented on Fig. 3. There exists an infinite number of finitary languages from which this automaton can be constructed. Indeed, to obtain them, it suffices to re-transform this Streett automaton into a minimal DFA by forgetting accepting pairs and changing the R-state into an accepting state. Then, from this accepting state, we can add arbitrary transitions. The automata produced by doing so will always be transformed by *DFA2S_Guar* into $\mathcal{A}_{\Pi 2}$.

4.4 Characterizing states of Streett automata

To better identify particular execution sequences on a Streett automaton we characterize some subsets of its states in terms of reachability of distinguished states. More precisely, the set $\mathbb{P}^{\mathcal{A}} = \{Good^{\mathcal{A}}, Good_c^{\mathcal{A}}, Bad_c^{\mathcal{A}}, Bad^{\mathcal{A}}\}$ is a partition of $Q^{\mathcal{A}}$, s.t. $Good^{\mathcal{A}}, Good_c^{\mathcal{A}}, Bad_c^{\mathcal{A}}, Bad^{\mathcal{A}}$ designate respectively the good (resp. currently good, currently bad, bad) states. The set $\mathbb{P}^{\mathcal{A}}$ is defined as follows:

- q is in $Good^{\mathcal{A}}$ iff it terminates an accepted sequence and every sequence starting from q is accepted: $Good^{\mathcal{A}} \stackrel{\text{def}}{=} \{q \in \bigcap_{i=1}^{m}(R_i \cup P_i) \mid Reach_{\mathcal{A}}(q) \subseteq \bigcap_{i=1}^{m}(R_i \cup P_i)\}$;
- q is in $Good_c^{\mathcal{A}}$ iff it terminates an accepted sequence and there exist non accepted sequences starting from q: $Good_c^{\mathcal{A}} \stackrel{\text{def}}{=} \{q \in \bigcap_{i=1}^{m}(R_i \cup P_i) \mid Reach_{\mathcal{A}}(q) \not\subseteq \bigcap_{i=1}^{m}(R_i \cup P_i)\}$;
- q is in $Bad_c^{\mathcal{A}}$ iff it terminates a non accepted sequence and there exist accepted sequences starting from q: $Bad_c^{\mathcal{A}} \stackrel{\text{def}}{=} \{q \in \bigcup_{i=1}^{m}(\overline{R_i} \cap \overline{P_i}) \mid Reach_{\mathcal{A}}(q) \not\subseteq \bigcup_{i=1}^{m}(\overline{R_i} \cap \overline{P_i})\}$;
- q is in $Bad^{\mathcal{A}}$ iff it terminates a non accepted sequence and every sequence starting from q is not accepted: $Bad^{\mathcal{A}} \stackrel{\text{def}}{=} \{q \in \bigcup_{i=1}^{m}(\overline{R_i} \cap \overline{P_i}) \mid Reach_{\mathcal{A}}(q) \subseteq \bigcup_{i=1}^{m}(\overline{R_i} \cap \overline{P_i})\}$.

The subsets are illustrated for basic classes in Fig. 7.

Example 3 (*Characterization of Streett automata states*) We illustrate the characterization on the states of the Streett automata presented in Example 2:

- $Bad^{\mathcal{A}_{\Pi 1}} = \{4\}, Good_c^{\mathcal{A}_{\Pi 1}} = \{1, 2, 3\}$,
- $Bad^{\mathcal{A}_{\Pi 2}} = \{4\}, Bad_c^{\mathcal{A}_{\Pi 2}} = \{1, 2\}, Good^{\mathcal{A}_{\Pi 2}} = \{3\}$,
- $Bad^{\mathcal{A}_{\Pi 3}} = \{3\}, Bad_c^{\mathcal{A}_{\Pi 2}} = \{2\}, Good_c^{\mathcal{A}_{\Pi 3}} = \{1\}$,
- $Bad^{\mathcal{A}_{\Pi 4}} = \{5\}, Bad_c^{\mathcal{A}_{\Pi 4}} = \{1, 2, 6\}, Good_c^{\mathcal{A}_{\Pi 4}} = \{3, 4\}$.

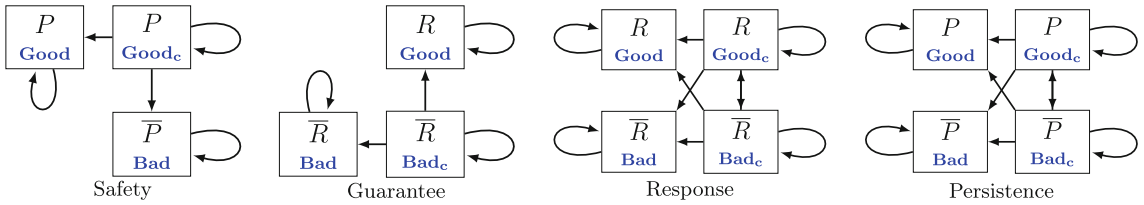

Fig. 7 Characterization of states for basic classes

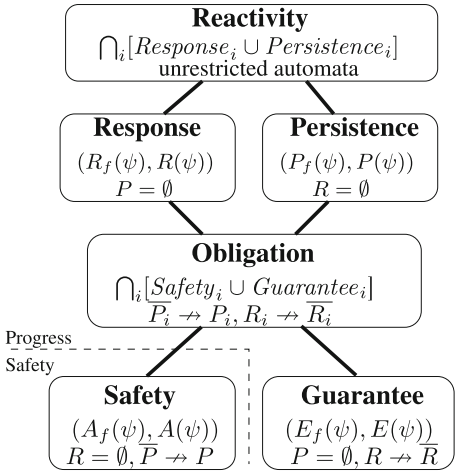

Fig. 8 The Safety-Progress classification of r-properties

Remark 2 For a Streett automaton $\mathcal{A}_\Pi$, all states in $Bad^{\mathcal{A}_\Pi}$ (resp. $Good^{\mathcal{A}_\Pi}$) are equivalent w.r.t. property recognition—they can be merged into one single state.

This characterization of states will be useful in the following sections when characterizing monitorable properties and when synthesizing monitors.

4.5 Summary

A graphical representation of the Safety-Progress hierarchy of properties is depicted in Fig. 8. A link between two classes means that the higher class contains strictly the lower one. Furthermore, for each class, we have recalled and uniformly extended the characterizations in the language-theoretic and automata views.

In Table 1 is represented each "basic block", i.e., the element used to build an r-property. In the language view, r-properties are built from a finitary language ψ, and using operators X_f, and X, with $X \in \{A, E, R, P\}$. In the automata view, a finite-state automaton is transformed, by one of the transformations DFA2S specific to a class of properties, into a Streett automaton which recognizes $(X_f(\psi), X(\psi))$ according to the class of properties.

Table 1 Ways to specify properties according to the views

Basic block	Language view $\psi \subseteq \Sigma^*$	Automata view $\mathcal{A}$ (DFA)
r-property	$(X_f(\psi), X(\psi))$	$DFA2S\_X(\mathcal{A})$
	$X \in \{A, E, R, P\}$	$X \in \{Saf., Guar., Resp., Persit.\}$

Table 2 Recognizing criteria according to the considered view

	Language view	Automata view
Finite seq	$\in X_f(\psi)$ (Definition 2)	Finite seq criterion (Definition 7)
Infinite seq	$\in X(\psi)$ (Definition 3)	Infinite seq criterion (Definition 6)

Remark 3 It is worth noticing that property interpretation of finite sequences extends to infinite sequences in a consistent way, depending on the class of properties under consideration:

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

5 Monitorability w.r.t. the SP classification

As stated in Sect. 1, studying the question of monitorability amounts to studying the expressiveness of runtime verification, i.e., characterizing the classes of properties that can be verified at runtime. In this section, we first recall and extend existing monitorability results in the Safety-Progress classification of properties. Second, we propose to parameterize the classical definition with a truth-domain. Third, we propose an alternative definition of monitorability and characterize monitorable properties according to this new definition.

In fact, characterizing the set of "monitorable" properties depends on several parameters: the property semantics for

finite sequences, the set of monitor verdicts we consider, and the exact definition of monitoring.

5.1 Monitorable properties according to the classical definition of monitorability

We express the classical definition of monitorability given by Pnueli and Zaks in the SP framework introduced in the previous section. Then we characterize the set of monitorable properties according to this classical definition.

5.1.1 The classical definition of monitorability

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. Intuitively, the idea is to be able to detect verdicts, i.e., find an evaluation, w.r.t. an infinitary property, from a finite observation of a system behavior. This is formalized as follows for r-properties:

Definition 13 (*Positive/negative determinacy of an r-property* [32]) Let $\sigma \in \Sigma^*$, an r-property $\Pi \subseteq \Sigma^* \times \Sigma^\omega$ is said to be:

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

An r-property is negatively (resp. positively) determined if every possible future continuation (finite or infinite) does not (resp. does) satisfy the property. The practical meaning is as follows: when a monitor observes a system in order to check a property, if this property is negatively or positively determined, then the observation of the system can be stopped. In this case, a monitor emits the verdict $\bot$ (resp. $\top$) after reading σ if the property is negatively (resp. positively determined) by σ. The obtained verdict is definitive. In other cases, a monitor issues the value "?", meaning that no definitive verdict can be produced.

Definition 14 (*Monitorable r-properties, "classical" definition* [32]) An 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 σ-monitorable for every $\sigma \in \Sigma^*$.

The set of monitorable properties, according to the classical definition is noted MP_c. An r-property is monitorable if, for any execution sequence that can be observed, a possible continuation of this sequence determines negatively or positively the property. One can notice that the classical definition of monitorability is bound to an implicit 3-valued truth-domain $\{\bot, ?, \top\}$ where the truth-values are issued by a monitor as described previously.

Remark that the classical definition of monitorability is hard to use in practice. Hence, a characterization of monitorable properties is needed in practice.

5.1.2 Characterization of monitorable properties according to the classical definition

One of our first objectives is to characterize the subset of *monitorable properties* within the Safety-Progress classification.

We first enunciate a lemma that will be used later on. This lemma states that the set of MP_c-monitorable properties is closed under Boolean operations.

Lemma 2 (Closure of monitorable properties under boolean operations) *Given two r-properties* Π_1, Π_2, *we have:*

$$\Pi_1, \Pi_2 \in MP_c \Rightarrow \Pi_1 \wedge \Pi_2 \in MP_c,$$
$$\Pi_1, \Pi_2 \in MP_c \Rightarrow \Pi_1 \vee \Pi_2 \in MP_c,$$
$$\Pi_1 \in MP_c \Rightarrow \neg\Pi_1 \in MP_c.$$

Proof The complete proof is given in Appendix A.2.1. Let us consider two r-properties $\Pi_1, \Pi_2 \in MP_c$.

- The proof of $\Pi_1 \wedge \Pi_2 \in MP_c$ consists in showing that $\Pi_1 \wedge \Pi_2$ is σ-monitorable for any sequence $\sigma \in \Sigma^*$. Let $\sigma \in \Sigma^*$, let us exhibit an extension $\mu \in \Sigma^*$ s.t. $\Pi_1 \wedge \Pi_2$ is negatively or positively determined by $\sigma \cdot \mu$. As Π_1 is monitorable, there exists a sequence μ_1 s.t. Π_1 is positively or negatively determined by $\sigma \cdot \mu_1$. Then, as Π_2 is monitorable, there exists a sequence μ_2 s.t. $\Pi_1 \wedge \Pi_2$ is negatively or positively determined by $\sigma \cdot \mu_1 \cdot \mu_2$. Then one has to analyze the different Boolean combinations to obtain the expected result.
- The proof of $\Pi_1 \vee \Pi_2 \in MP_c$ is similar.
- The proof of $\neg\Pi_1 \in MP_c$ is straightforward by noticing that for any sequence $\sigma \in \Sigma^*$, if Π_1 is positively (resp. negatively) determined by σ, then $\neg\Pi$ is negatively (resp. positively) determined by σ. □

We are now able to establish that the set of monitorable properties according to the classical definition strictly contains the set of obligation properties.

Theorem 2 (Obligation(Σ) $\subset$ MPc) *The obligation properties are strictly contained in the set of monitorable properties.*

Proof The formal proof can be found in Appendix A.2.2 and uses the following facts:

- Safety and guarantee properties are monitorable.
- The set of obligation properties is the union of the sets of k-obligation properties for $k \geq 1$ (Lemma 1).
- Union and intersection of two monitorable properties are monitorable (Lemma 2).
- Example 6 shows that the inclusion is strict. □

Thus, we have extended the previous bound established by Bauer et al. in [3][7] stating that

$$\text{Safety}(\Sigma) \cup \text{Guarantee}(\Sigma) \subset MP_c.$$

Indeed, the set of obligation properties is a strict super set of the union of safety and guarantee properties.

Example 4 (*Classical monitoring of an obligation property*) We go back to the example presented in Sect. 1, defined using two atomic propositions p or q, stating that p should always hold or q should eventually hold. This is a 1-obligation r-property,[8] defined as the disjunction of a safety r-property ("p should always hold") and a guarantee r-property ("q should eventually hold"). According to the classical definition of monitorability, this property is monitorable. Indeed, for any finite sequence σ, this property can be positively determined by $\sigma \cdot \{\overline{p}, q\}$ or by $\sigma \cdot \{p, q\}$, i.e., by completing σ with an event in which q is true.

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, whereas the other one is monitorable.

Example 5 (*Non-monitorable response property* [3]) The (response) property "Every request is eventually followed by an acknowledgement"[9] is not monitorable. This property is represented by the Streett (response) automaton depicted in Fig. 9 with $R = \{1\}$. Its alphabet is $\Sigma = \{req, ack, oth\}$ where *req* (resp. *ack*, *oth*) denotes the request (resp. the acknowledgment, any other event). Using the acceptance criteria for finite and infinite sequences, one can reasonably be convinced that this automaton defines the considered property. Indeed, a finite sequence is accepted if and only if previous requests have been acknowledged. An infinite sequence is accepted if and only if state 1 is visited infinitely often which means for an infinite sequence that requests have been acknowledged.

For this property, there are two limitations for monitoring using the classical 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. In other words, it is never possible to satisfy or falsify this property with a finite observation.

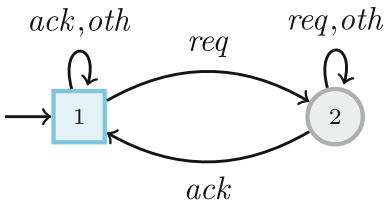

Fig. 9 Non-monitorable response property—$R = \{1\}$, $P = \emptyset$

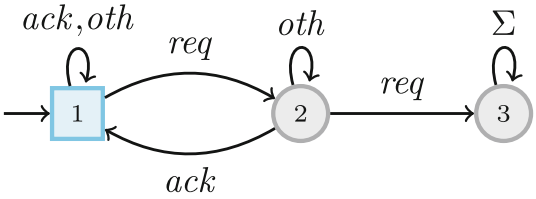

Fig. 10 Monitorable response property—$R = \{1\}$, $P = \emptyset$

Example 6 (*Monitorable response property*) The (response) property "Every request should be acknowledged, and it is forbidden to send two successive requests (without acknowledgment)" is monitorable. This property is represented by the Streett response automaton depicted in Fig. 10 with $R = \{1\}$ and the same alphabet Σ as in Example 5. Notice that this property is indeed a pure response property: it cannot be represented by a *finite-state* Streett obligation automaton. Intuitively, given an execution sequence, this r-property can always be negatively determined by one of its continuations. Indeed, for any $\sigma \in \Sigma^*$, the property is negatively determined by $\sigma \cdot req \cdot req$ and is thus σ-monitorable.

Thus, there exist monitorable (pure) response properties. Consequently, using Lemma 2, there exist also monitorable pure persistence and reactivity properties. Indeed, monitorable properties are closed under Boolean operations.

5.2 Parametrization of the classical definition of monitorability

As we will see, the characterization of monitorable properties may also depend on a truth-domain $\mathbb{B}$ we consider when evaluating an execution sequence. Thus, we parameterize the classical definition of monitorable properties with a truth-domain.

The first truth-domain we have studied is a 3-valued truth-domain $\mathbb{B}_3 \stackrel{\text{def}}{=} \{\bot, ?, \top\}$. This truth-domain is inherent in the classical definition. The value "$\bot$" is used to express property violation when the property is negatively determined. The value "$\top$" is used to express property satisfaction when

[7] In [3], guarantee properties are named co-safety properties.

[8] Seen in the logical view, this property can be defined by the temporal logic formula $\Box p \vee \Diamond q$.

[9] This property can be expressed in an event-based LTL as $\Box(req \Rightarrow \Diamond ack)$.

the property is positively determined. The value "?" is used to express that no verdict can be produced. $\mathbb{B}_3$ can be viewed as a complete lattice, whose minimal value is $\bot$ and maximal value is $\top$. Boolean operators $\vee$ and $\wedge$ are then defined respectively as upper and lower bounds.

The classical definition of monitorability is purposed to the detection of verdicts for infinitary properties, i.e., to detect either $\top$ or $\bot$. This definition can be parameterized by a truth-domain $\mathbb{B}$ containing at least one of these two elements:

Definition 15 (*Parameterized classical monitorability*) For a truth-domain $\mathbb{B}$ s.t. $\mathbb{B} \cap \{\bot, \top\} \neq \emptyset$, an r-property Π is:

- σ-monitorable with $\mathbb{B}$, if there exist $b \in \mathbb{B}$ and a (finite) $\mu \in \Sigma^*$ s.t.
  - $b = \bot$ and Π is negatively determined by $\sigma \cdot \mu$, or
  - $b = \top$ and Π is positively determined by $\sigma \cdot \mu$;
- monitorable with $\mathbb{B}$ if it is σ-monitorable with $\mathbb{B}$ for every $\sigma \in \Sigma^*$.

For a truth-domain $\mathbb{B}$, we will note $MP(\mathbb{B})$ the set of monitorable properties, according to the definition of parameterized classical monitorability. We now tackle the question of how the underlying considered truth-domain may influence the class of monitorable properties, according to a truth-domain derived from $\mathbb{B}_3$.

Remark 4 ($MP_c = MP(\mathbb{B}_3)$) The previous parameterized definition of classical monitorability amounts to the classical definition given by Pnueli and Zaks when instantiated with $\mathbb{B}_3$.

Remark 5 (*Finer truth-domains*) According to Definition 15, we can notice that adding further truth-values to $\mathbb{B}_3$ has no influence on the set of monitorable properties, i.e., $\forall \mathbb{B} : \mathbb{B} \supset \mathbb{B}_3 \Rightarrow MP(\mathbb{B}) = MP(\mathbb{B}_3)$.

This latest remark shows a first limitation of the definition of (parameterized) classical monitorability: considering finer truth-domains, as it could be required in specific application domains, will not increase the set of monitorable properties. This is one of the motivations to introduce an alternative definition of monitorability (in Sect. 5.3).

Monitorability with a truth-domain of cardinality 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. Indeed, the purpose of the monitor is then either to detect only bad behaviors *or* only good behaviors (but not both). In the sequel, we consider two subsets of $\mathbb{B}_3$, namely $\mathbb{B}_2^{\bot} \stackrel{\text{def}}{=} \{\bot, ?\}$ and $\mathbb{B}_2^{\top} \stackrel{\text{def}}{=} \{?, \top\}$.

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. Actually, there are numerous safety properties which can never be negatively determined. For example, the r-property $true = (\Sigma^*, \Sigma^\omega) = (A_f(\Sigma^*), A(\Sigma^*))$ can neither be negatively determined nor falsified. Moreover, all safety properties which are valid forever for execution sequences longer than a given $k \in \mathbb{N}$ are not $\sigma - \mathbb{B}_2^{\bot}$-monitorable when $|\sigma| > k$. For these kinds of properties, a monitor would produce only verdict sequences containing "?" when evaluating an execution sequence. Similarly, there exist many guarantee properties that cannot be positively determined and therefore are not monitorable with $\mathbb{B}_2^{\top} = \{?, \top\}$.

Regarding these sets of monitorable properties, it appears that there is no simple characterization, in terms of classes of the Safety-Progress classification. However, in Sect. 5.4, we will provide a syntactic criterion on Streett automata in order to decide whether the r-property recognized by a given automaton is monitorable according to the mentioned truth-domains.

5.3 Monitorable properties according to an alternative definition of monitorability

The interest of previous definitions of monitorability is due to two facts: the underlying truth-domain is 2- 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- or 3-valued truth-domain, the question is whether those values make sense for some properties in a monitoring context.

As noticed in [3,26], it seems interesting to investigate further the set 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- or a 3-valued truth-domain.

Hence, the authors of [3,26] proposed to consider a 4-valued truth-domain $\mathbb{B}_4 = \{\bot, \bot_c, \top_c, \top\}$. The truth-value $\top_c$ (resp. $\bot_c$) denotes "currently true" (resp. "currently false") and it expresses "Π-satisfaction (resp. Π-violation) if the program execution stops here". Boolean operators $\vee$ and $\wedge$ are defined in [3]. 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.

5.3.1 Property evaluation in a truth-domain

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

Definition 16 (*Property evaluation w.r.t. a truth-domain*) For each of the possible truth-domain $\mathbb{B}$, we define the evaluation functions $[\![\cdot]\!]_{\mathbb{B}}(\cdot) : 2^{\Sigma^* \times \Sigma^\omega} \times \Sigma^* \to \mathbb{B}$ as follows:

For $\mathbb{B}_2^\perp$:

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

For $\mathbb{B}_2^\top$:

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

For $\mathbb{B}_3$:

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

For $\mathbb{B}_4$:

$[\![\Pi]\!]_{\mathbb{B}_4}(\sigma) = [\![\Pi]\!]_{\mathbb{B}_3}(\sigma)$ if $[\![\Pi]\!]_{\mathbb{B}_3}(\sigma) = \perp \vee [\![\Pi]\!]_{\mathbb{B}_3}(\sigma) = \top$,
$[\![\Pi]\!]_{\mathbb{B}_4}(\sigma) = \top_c$ if $[\![\Pi]\!]_{\mathbb{B}_3}(\sigma) = ? \wedge \Pi(\sigma)$,
$[\![\Pi]\!]_{\mathbb{B}_4}(\sigma) = \perp_c$ if $[\![\Pi]\!]_{\mathbb{B}_3}(\sigma) = ? \wedge \neg\Pi(\sigma)$.

Remark 6 The defined property evaluation w.r.t. $\mathbb{B}_4$ is similar to the semantics of the LTL variant RV-LTL defined in [4].

5.3.2 An alternative definition of monitorability

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

Definition 17 (*Alternative monitorability*) An r-property $\Pi = (\phi, \varphi)$ is said to be monitorable with the truth-domain $\mathbb{B}$, or $\mathbb{B}$-monitorable if

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

We note $MP^*(\mathbb{B})$, the set of monitorable properties with truth-domain $\mathbb{B}$ according to this definition.

Thus, an r-property is monitorable with a given truth-domain $\mathbb{B}$ if and only if evaluations of good and bad finite execution sequences lead to *distinct* values. Remark that this definition seemingly does not rely on the infinitary part of the r-property (although this infinitary part is taken into account in the evaluation function).

5.3.3 Characterization of monitorable properties

Lemma 3 ($MP^*(\mathbb{B}_3)$, safety, and guarantee properties) *The set of monitorable properties (according to Definition 17) with $\mathbb{B}_3$ is included in the union of safety and guarantee properties. Formally:*

$$MP^*(\mathbb{B}_3) \subseteq \text{Safety}(\Sigma) \cup \text{Guarantee}(\Sigma).$$

Proof The formal proof can be found in Appendix A.2.3. It is done by *reductio ad absurdum* and supposing the existence of an r-property $\Pi = (\phi, \varphi)$ defined on Σ which is neither a safety nor a guarantee r-property. The proof shows the existence of two execution sequences, one good, the other one bad, for Π s.t. these sequences are evaluated to "?". □

Theorem 3 (Multi-valued characterization of alternative monitorability) *The sets of monitorable properties according to the truth-domains considered so far are as follows:*

(i) $MP^*(\mathbb{B}_2^\perp) = \text{Safety}(\Sigma)$,
(ii) $MP^*(\mathbb{B}_2^\top) = \text{Guarantee}(\Sigma)$,
(iii) $MP^*(\mathbb{B}_3) = \text{Safety}(\Sigma) \cup \text{Guarantee}(\Sigma)$,
(iv) $MP^*(\mathbb{B}_4) = \text{Reactivity}(\Sigma)$.

The proof can be found in Appendix A.2.4.

Example 7 (*Alternative monitoring of an obligation property*) Let us go back again to the property considered in Sect. 1, stating that "p should *always* hold or q should *eventually* hold". We examine again the execution sequences: $\sigma_{good} = \{p, \overline{q}\} \cdot \{p, \overline{q}\}$ and $\sigma_{bad} = \{p, \overline{q}\} \cdot \{\overline{p}, \overline{q}\}$. Considering $\mathbb{B}_3$, 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 $[\![\Pi]\!]_{\mathbb{B}_4}(\sigma_{good}) = \top_c$ and $[\![\Pi]\!]_{\mathbb{B}_4}(\sigma_{bad}) = \perp_c$.

This example shows how the finite sequence semantics leverages the interest of monitoring certain properties. Furthermore, it shows that under our definition of monitoring, ambiguous situations, such as those encountered with the classical definition, are avoided.

Our definition of monitorability has the advantage of being able to identify the properties which should not be monitored with a truth-domain "not fine enough". Indeed, the last property shows that if we build a monitor for such a property with the truth-domain $\mathbb{B}_3$, this monitor would produce an evaluation "?" for correct and incorrect execution sequences w.r.t. the property. This seems not desirable to us.

Furthermore, we have shown in Sect. 4.4 that, for a given finite sequence σ and an r-property Π, $[\![\Pi]\!]_{\mathbb{B}_4}(\sigma)$ is easy to compute from the set of states of a Streett automaton recognizing Π.

5.4 Characterizations in the automata view

Although some sets of monitorable properties we considered cannot be precisely expressed in terms of Safety-Progress classes, it is still possible to characterize them with some syntactic criteria on Streett automata. It relies on the characterization of the states of Streett automata introduced in Sect. 4.4.

Property 3 (*Correspondence between Streett automata states and* $\mathbb{B}_4$) Given a Streett m-automaton recognizing an r-property Π and a sequence $\sigma \in \Sigma^*$ of length n s.t. $run(\sigma, \mathcal{A}_\Pi) = q_0 \cdots q_n$, we have:

$$
\begin{aligned}
q_n \in Good^{\mathcal{A}_\Pi} &\Leftrightarrow [\![\Pi]\!]_{\mathbb{B}_4}(\sigma) = \top,\\
q_n \in Good_c^{\mathcal{A}_\Pi} &\Leftrightarrow [\![\Pi]\!]_{\mathbb{B}_4}(\sigma) = \top_c,\\
q_n \in Bad_c^{\mathcal{A}_\Pi} &\Leftrightarrow [\![\Pi]\!]_{\mathbb{B}_4}(\sigma) = \bot_c,\\
q_n \in Bad^{\mathcal{A}_\Pi} &\Leftrightarrow [\![\Pi]\!]_{\mathbb{B}_4}(\sigma) = \bot.
\end{aligned}
$$

Proof The proof is given in Appendix A.2; it uses the acceptance criteria of Streett automata to establish the correspondence. □

Remark 7 (*Correspondence with* $\mathbb{B}_3$, $\mathbb{B}_2^\bot$, *and* $\mathbb{B}_2^\top$) From Property 3 and Definition 16, one can easily deduce a correspondence between the set of states and the evaluation in the truth-domain of a lower cardinality:

- For $\mathbb{B}_3$:
  - $q_n \in Good^{\mathcal{A}_\Pi} \Leftrightarrow [\![\Pi]\!]_{\mathbb{B}_3}(\sigma) = \top$,
  - $q_n \in Good_c^{\mathcal{A}_\Pi} \cup Bad_c^{\mathcal{A}_\Pi} \Leftrightarrow [\![\Pi]\!]_{\mathbb{B}_3}(\sigma) = ?$
  - $q_n \in Bad^{\mathcal{A}_\Pi} \Leftrightarrow [\![\Pi]\!]_{\mathbb{B}_3}(\sigma) = \bot$.
- For $\mathbb{B}_2^\top$:
  - $q_n \in Good^{\mathcal{A}_\Pi} \Leftrightarrow [\![\Pi]\!]_{\mathbb{B}_2^\top}(\sigma) = \top$,
  - $q_n \in Good_c^{\mathcal{A}_\Pi} \cup Bad_c^{\mathcal{A}_\Pi} \cup Bad^{\mathcal{A}_\Pi} \Leftrightarrow [\![\Pi]\!]_{\mathbb{B}_2^\top}(\sigma) = ?$.
- For $\mathbb{B}_2^\bot$:
  - $q_n \in Bad^{\mathcal{A}_\Pi} \Leftrightarrow [\![\Pi]\!]_{\mathbb{B}_2^\bot}(\sigma) = \bot$,
  - $q_n \in Bad^{\mathcal{A}_\Pi} \cup Good_c^{\mathcal{A}_\Pi} \cup Bad_c^{\mathcal{A}_\Pi} \Leftrightarrow [\![\Pi]\!]_{\mathbb{B}_2^\bot}(\sigma) = ?$.

Now we are able to give an exact characterization of monitorable properties in the automata view.

Theorem 4 (Automata view of classical monitorability) *The r-property* Π *recognized by the Streett m-automaton* $\mathcal{A}_\Pi = (Q^{\mathcal{A}_\Pi}, q_{\text{init}}^{\mathcal{A}_\Pi}, \rightarrow_{\mathcal{A}_\Pi}, \{(R_1, P_1), \cdots, (R_m, P_m)\})$ *is*

(i) $MP(\mathbb{B}_2^\bot)$*-monitorable iff*
$\forall q \in Reach(q_{\text{init}}^{\mathcal{A}_\Pi}), Reach(q) \cap Bad^{\mathcal{A}_\Pi} \neq \emptyset$;

(ii) $MP(\mathbb{B}_2^\top)$*-monitorable iff*
$\forall q \in Reach(q_{\text{init}}^{\mathcal{A}_\Pi}), Reach(q) \cap Good^{\mathcal{A}_\Pi} \neq \emptyset$;

(iii) $MP(\mathbb{B}_3)$*-monitorable iff*
$\forall q \in Reach(q_{\text{init}}^{\mathcal{A}_\Pi}), Reach(q) \cap (Bad^{\mathcal{A}_\Pi} \cup Good^{\mathcal{A}_\Pi}) \neq \emptyset$.

Proof This property is established by noticing, first, that it is a consequence of Property 3 and, second, that we are considering deterministic and complete Streett automata. Thus, the two following facts are equivalent:

- from every accessible state, a bad (resp. good, bad or good) is reachable;
- every finite sequence has a continuation that determines negatively (resp. positively, negatively or positively) the underlying property. □

Example 8 (*Automata view of classical monitorability*) We illustrate the use of the previous theorem to state whether the properties of Example 2 (Fig. 3), with their provided automata, are monitorable according to the classical definition:

- The property Π_1 specified by $\mathcal{A}_{\Pi 1}$ is $\mathbb{B}_2^\bot$-monitorable, and thus $\mathbb{B}_3$-monitorable; but not $\mathbb{B}_2^\top$-monitorable.
- The property Π_2 specified by $\mathcal{A}_{\Pi 2}$ is $\mathbb{B}_2^\top$-monitorable and $\mathbb{B}_2^\bot$-monitorable, and thus $\mathbb{B}_3$-monitorable.
- The property Π_3 specified by $\mathcal{A}_{\Pi 3}$ is $\mathbb{B}_2^\bot$-monitorable, and thus $\mathbb{B}_3$-monitorable, but not $\mathbb{B}_2^\top$-monitorable.
- The property Π_4 specified by $\mathcal{A}_{\Pi 4}$ is $\mathbb{B}_2^\bot$-monitorable, and thus $\mathbb{B}_3$-monitorable, but not $\mathbb{B}_2^\top$-monitorable.

Theorem 5 (Automata view of alternative monitorability) *The r-property* Π *recognized by the Streett m-automaton* $\mathcal{A}_\Pi = (Q^{\mathcal{A}_\Pi}, q_{\text{init}}^{\mathcal{A}_\Pi}, \rightarrow_{\mathcal{A}_\Pi}, \{(R_1, P_1), \cdots, (R_m, P_m)\})$ *is*

(i) $MP^*(\mathbb{B}_2^\bot)$*-monitorable iff* $Bad^{\mathcal{A}_\Pi} = \bigcup_{i=1}^m \overline{R_i} \cap \overline{P_i}$;

(ii) $MP^*(\mathbb{B}_2^\top)$*-monitorable iff* $Good^{\mathcal{A}_\Pi} = \bigcap_{i=1}^m R_i \cup P_i$;

(iii) $MP^*(\mathbb{B}_3)$*-monitorable iff*

$$
\begin{aligned}
&\nexists q \in Reach(q_{\text{init}}^{\mathcal{A}_\Pi}) \cap \textstyle\bigcap_{i=1}^m R_i \cup P_i,\\
&\nexists q' \in Reach(q_{\text{init}}^{\mathcal{A}_\Pi}) \cap \textstyle\bigcup_{i=1}^m \overline{R_i} \cap \overline{P_i},\\
&q \in Good_c^{\mathcal{A}_\Pi} \wedge q' \in Bad_c^{\mathcal{A}_\Pi}.
\end{aligned}
$$

Proof The proof is conducted in three steps:

(i) The expressed condition generalizes the syntactic restriction of Streett safety automata and is also a condition when a given not minimal Streett m-automaton is recognizing a safety property and can be minimized so as to be represented as a Streett safety automaton.

(ii) The expressed condition generalizes the syntactic restriction of Streett guarantee automata and is also a condition when a given not minimal Streett m-automaton is recognizing a guarantee property and can be

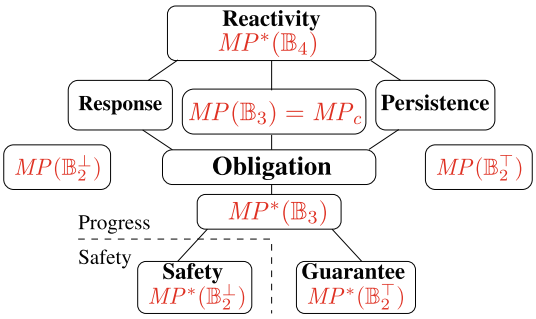

Fig. 11 Monitorable r-properties in the Safety-Progress classification

minimized so as to be represented as a Streett safety automaton.

(iii) The third condition can be established using the two following facts:

- An r-property is not monitorable according to this theorem if and only if for two sequences, a good and a bad sequences evaluate to "?". Other evaluations are not simultaneously possible for a bad and good sequences.
- We are considering deterministic and complete Streett automata. □

5.5 Summary

We depict in Fig. 11 the main results obtained in this section, which can be summarized as follows:

- The classes of monitorable properties, according to the classical definition, are:
  - $MP(\mathbb{B}_2^{\top})$, which cannot be compared directly with any other class;
  - $MP(\mathbb{B}_2^{\perp})$, which cannot be compared directly with any other class;
  - and $MP(\mathbb{B}_3) = MP_c$ which contains strictly the class of obligation properties.
- The classes of monitorable properties, according to the new definition we introduced are:
  - $MP^*(\mathbb{B}_2^{\perp})$ is the set of safety r-properties;
  - $MP^*(\mathbb{B}_2^{\top})$ is the set of guarantee r-properties;
  - $MP^*(\mathbb{B}_3)$ is strictly contained in the class of obligation r-properties and is the union of the sets of safety and guarantee properties;
  - and $MP^*(\mathbb{B}_4)$ is the set of reactivity r-properties.

Remark that using the truth-domain $\mathbb{B}_4$ does not add any expressiveness to the classical definition of monitorability (i.e., $MP(\mathbb{B}_4) = MP(\mathbb{B}_3)$). Indeed, this definition is bound to the notion of positive and negative determinacy. However, using this domain would permit to better distinguish execution sequences and to avoid ambiguity exposed in Example 4. Note also that some obligation properties (between $MP(\mathbb{B}_3)$ and $MP^*(\mathbb{B}_3)$) should not be monitored unless with a truth-domain equipped with an interpretation of finite sequences allowing to distinguish good and bad finite sequences (e.g., with truth-values $\perp_c$ and $\top_c$).

Remark 8 One can notice that the monitorability results expressed in the automata view (i.e., Theorems 4 and 5) hold only for regular r-properties. Whereas the results that were not expressed in the automata view (i.e., Lemma 2, Theorems 2 and 3) hold for all r-properties (e.g., regular, context-free, context-sensitive, recursively enumerable), independently from any computability constraint on their recognizing mechanisms.

6 Enforceability w.r.t. the SP classification

Now we address the question of the expressiveness of runtime enforcement, i.e., we characterize the class of enforceable properties. In Sect. 3, we have seen that the previous proposed classes 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, provided that this memory is unbounded but finite. This will give us an upper-bound of the set of enforceable properties with any enforcement mechanism.

6.1 Enforcement criteria

The enforcement constraints exposed in Sect. 3, namely *soundness* and *transparency*, express a relation between the input sequence (submitted to an enforcement monitor) and an output sequence (produced by this monitor). We interpret these constraints in the following way: if the input sequence already verifies the property, then it should remain unchanged (up to a given equivalence relation); otherwise, its *longest prefix* satisfying the property should be issued.[10]

A consequence is that an 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

[10] An alternative interpretation could consist in correcting an erroneous sequence by adding extra events.

prefixes.[11] We give two enforcement criteria in the language and automata views.

Definition 18 (*Enforcement criterion in the language view*) An r-property (ϕ, φ) is said to be enforceable if $\forall \sigma \in \Sigma^\omega$,

$$\neg\varphi(\sigma) \Rightarrow \exists \sigma' \in \Sigma^*, \sigma' \prec \sigma \wedge \forall \sigma'' \in \Sigma^*, \sigma' \prec \sigma'' \prec \sigma \Rightarrow \neg\phi(\sigma'') \quad (4)$$

Alternatively, it is possible to express an enforcement criterion for an r-property Π on its recognizing Streett m-automaton $\mathcal{A}_\Pi$. This criterion uses the set $\mathcal{S}(\mathcal{A}_\Pi)$ of (maximal and non-maximal) Strongly Connected Components of $\mathcal{A}_\Pi$. For an m-automaton $\mathcal{A}_\Pi$ and s an SCC of $\mathcal{A}_\Pi$, we define $I_s \stackrel{\text{def}}{=} \{i \in [1, m] \mid s \subseteq \overline{R_i} \wedge s \cap \overline{P_i} \neq \emptyset\}$. Intuitively, I_s is the set of accepting pairs of $\mathcal{A}_\Pi$ that leads the infinite sequences, visiting infinitely often the SCC s, to be rejected. The criterion is formally expressed as follows:

Definition 19 (*Enforcement criterion in the automata view*) The r-property Π recognized by an m-automaton $\mathcal{A}_\Pi$ is said to be enforceable if

$$\forall s \in \mathcal{S}(\mathcal{A}_\Pi),\ I_s \neq \emptyset \Rightarrow \exists i \in I_s, s \subseteq \overline{P_i} \quad (5)$$

Intuitively, this later enforcement criterion states that for every non-accepting SCC s (s is non-accepting because it contains at least one pair in $[1, m]$ for which the infinite acceptance criterion is not satisfied), then, among these indexes, there is at least one index i s.t. all states in the SCC s are in $\overline{P_i}$.

Enforcement criteria of Definitions 18 and 19 are equivalent, as stated below.

Property 4 (*Equivalence between enforcement criteria*) Considering an r-property $\Pi = (\phi, \varphi)$, recognized by a Streett m-automaton $(Q^{\mathcal{A}_\Pi}, q_{\text{init}}^{\mathcal{A}_\Pi}, \Sigma, \rightarrow_{\mathcal{A}_\Pi}, \{(R_1, P_1), \ldots, (R_m, P_m)\})$, we have:

(4) $\Leftrightarrow$ (5).

Proof We just give a sketch of the proof, the formal proof can be found in Appendix A.3.1. The proof shows the implications in both ways and uses the definitions of the acceptance criteria of Streett automata for finite and infinite sequences. Then one has to remark that:

- (4) expresses a condition on the set of infinite sequences that do not satisfy the r-property;
- (5) expresses a condition on the set of SCC s that reject the sequences visiting s infinitely often.

As we are dealing with complete and deterministic automata, these conditions are equivalent. □

The set of enforceable r-properties, equivalently defined by Definition 4 or Definition 5, is denoted EP. We will now characterize EP w.r.t. the SP classification. We will prove that EP is *exactly* the class of response properties. Note that the enforcement criterion in the automata view is still useful as it provides a syntactic procedure to determine whether a property is enforceable or not.

6.2 Enforceable properties

We start first by proving that response properties are enforceable. Then we give the intuition on the non-enforceability of persistence properties by providing an illustrative example. Then we prove that the set of response properties is exactly the set EP.

Theorem 6 (Response properties 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 as $(R_f(\psi), R(\psi))$ for a given finitary language ψ. Let us 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 $\neg R(\psi)(\sigma)$, this set is not empty. Let us note σ_0 the smallest element of S regarding $\prec$. We have $\forall \sigma' \in \Sigma^*, \sigma_0 \prec \sigma' \prec \sigma \Rightarrow \neg\psi(\sigma')$. Since $\forall \psi' \subseteq \Sigma^*, R_f(\psi') \subseteq \psi'$ (cf. the definition of operators building finitary properties), it implies that $\forall \sigma' \in \Sigma^*, \sigma_0 \prec \sigma' \prec \sigma \Rightarrow \neg\phi(\sigma')$. Thus, Π is enforceable according to Definition 18. □

A straightforward consequence is that safety, guarantee and obligation r-properties are enforceable. We prove that, in fact, pure persistence properties are not enforceable. Let us explain the intuition through an example.

For $\Sigma \supseteq \{a, b\}$, 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 in Fig. 12 (with $P = \{1\}$). One can understand the enforcement limitation intuitively with the following argument: if this property was enforceable it would imply that an enforcement monitor 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 Definition 18, an

[11] Note that those criteria differ from the existence of bad prefixes. Bad prefixes are sequences which cannot be extended to correct (finite or infinite) ones, i.e., sequences that determine negatively the underlying property.

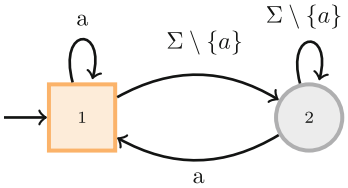

Fig. 12 Non-enforceable persistence r-property

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 $\sigma'_{bad} = (a \cdot b)^\omega$ does not satisfy the property whereas an infinite number of its prefixes do (prefixes ending with a).

Applying enforcement criteria on persistence properties, it turns out that the enforceable persistence properties are in fact response properties.

Theorem 7 (Enforceable persistence properties are response properties)

$$\text{Persistence}(\Sigma) \cap EP \subseteq \text{Response}(\Sigma).$$

Proof A persistence r-property Π 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 $\mathcal{A}_\Pi$ (see Definition 19). 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, the states visited infinitely often are then either all in P or $\overline{P}$, i.e., $\forall\sigma \in \Sigma^\omega$, $\mathit{vinf}(\sigma, \mathcal{A}_\Pi) \cap P \neq \emptyset \Leftrightarrow \mathit{vinf}(\sigma, \mathcal{A}_\Pi) \subseteq P$. On such automaton, there is no difference between R-states and P-states. Consequently, by re-tagging P-states to R-states, this automaton recognizes the same property. The re-tagged automaton is a response automaton. □

Corollary 1 *Pure persistence properties are not enforceable:*

$$(\text{Persistence}(\Sigma) \setminus \text{Response}(\Sigma)) \cap EP = \emptyset.$$

Proof This is a direct consequence of Theorem 7. □

Theorem 8 (Enforceable m-reactivity properties are response properties)

$$\text{Reactivity}(\Sigma) \cap EP \subseteq \text{Response}(\Sigma).$$

Proof The formal proof can be found in Appendix A.3.2. The proof shows that an m-reactivity automaton which is constrained by the enforcement criterion (Definition 19) can be translated to an m-response automaton accepting exactly the same sequences. Thus, the only enforceable reactivity properties are the response ones. □

Corollary 2 *Pure reactivity properties are not enforceable:*

$$\text{Reactivity}(\Sigma) \not\subseteq EP$$
$$\text{Reactivity}(\Sigma) \setminus (\text{Persistence}(\Sigma) \cup \text{Response}(\Sigma)) \cap EP = \emptyset.$$

Proof This is a direct consequence of Theorem 8. □

Corollary 3 *Enforceable properties are exactly response properties:*

$$EP = \text{Response}(\Sigma).$$

Proof It remains to prove that the set of enforceable properties is included in the set of response properties. 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 either a pure persistence or a pure reactivity property. Consequently, this property would not be enforceable. □

Example 9 (*Enforceable and not enforceable properties*) We illustrate the enforcement criterion on the properties introduced in Example 1 and represented by their Streett automata described in Example 2 and depicted in Fig. 3.

- The properties Π_1, Π_2, Π_3 are enforceable.
- The property Π_4 is not enforceable; e.g., the infinite sequence $r \cdot g \cdot (r \cdot d \cdot r \cdot g)^\omega$ is not accepted while this sequence has an infinite number of correct prefixes: e.g., all sequences belonging to $r \cdot g \cdot (r \cdot d \cdot r \cdot g)^*$.

Being enforceable or not can be determined rather easily either by observing the automata and using the acceptance criteria for finite and infinite sequences or by using the criterion in the automata view.

7 Monitor synthesis

Now we show how it is possible to obtain easily a monitor either for verifying or enforcing a property thanks to the framework introduced in Sect. 4. 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 the input sequence and produce an output sequence. The relation between the input and output sequences should follow enforcement monitoring constraints: soundness and transparency (Sect. 3.2).

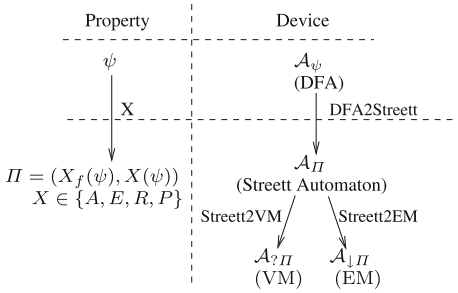

Fig. 13 Automaton transformations

In the following, we consider a Streett m-automaton $\mathcal{A}_\Pi = (Q^{\mathcal{A}_\Pi}, q_{\text{init}}^{\mathcal{A}_\Pi}, \rightarrow_{\mathcal{A}_\Pi}, \{(R_1, P_1), \ldots, (R_m, P_m)\})$ and Π the r-property recognized by $\mathcal{A}_\Pi$. Moreover, we evaluate properties only in $\mathbb{B}_4$, and consequently, we abbreviate $[\![\Pi]\!]_{\mathbb{B}_4}(\cdot)$ by $[\![\Pi]\!](\cdot)$.

The general monitor synthesis procedure is depicted in Fig. 13. From a "pattern" X corresponding to one of the basic classes of the hierarchy and a DFA $\mathcal{A}_\psi$ recognizing a finitary property $\psi \subseteq \Sigma^*$, DFA2S_X yields a Streett automaton recognizing the r-property $(X_f(\psi), X(\psi))$. Then using *Streett2VM* (resp. *Streett2EM*) one is able to obtain a verification (resp. enforcement) monitor for the r-property $(X_f(\psi), X(\psi))$.

7.1 Monitor: a general definition

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 20 (*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 $\longrightarrow_{\mathcal{A}}: Q^{\mathcal{A}} \times \Sigma \rightarrow Q^{\mathcal{A}}$ 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 $\Gamma^{\mathcal{A}} : Q^{\mathcal{A}} \rightarrow X^{\mathcal{A}}$ is an output function, producing values in $X^{\mathcal{A}}$ from states.

7.2 Synthesizing monitors for runtime verification

In the following,[12] we consider monitorable r-properties Π and (ϕ, φ) in $MP^*(\mathbb{B}_4)$.

Definition 21 (*Verification monitor*) A *verification monitor (VM)* $\mathcal{A}_?$ is a monitor with $X^{\mathcal{A}} = \mathbb{B}_4$.

Such monitors are independent from any specification formalism and can be easily adapted to the specification formalism from which they are generated. We define the notion of *verification sequence* produced by a monitor and what it means to verify a property for a monitor. It amounts to define the verification performed by a VM $\mathcal{A}_?$ while reading an input $\sigma \in \Sigma^*$ (produced by $\mathcal{P}_\Sigma$) and producing a sequence $b \in {\mathbb{B}_4}^+$.

Definition 22 (*Sequence verification*) The verification function $[\![\mathcal{A}_?]\!](\cdot) : \Sigma^* \rightarrow {\mathbb{B}_4}^+$, defining the verification performed by $\mathcal{A}_?$, produces a verification sequence while reading σ. This verification sequence is defined as follows:

$$\forall \sigma \in \Sigma^*, [\![\mathcal{A}_?]\!](\sigma) = \Gamma^{\mathcal{A}_?}(q_{\text{init}}^{\mathcal{A}_?}) \cdots \Gamma^{\mathcal{A}_?}(q_n) \tag{6}$$

with $q_{\text{init}}^{\mathcal{A}_?} \xrightarrow{\sigma_0}_{\mathcal{A}_?} q_1 \xrightarrow{\sigma_1}_{\mathcal{A}_?} \cdots q_{n-1} \xrightarrow{\sigma_{n-1}}_{\mathcal{A}_?} q_n$ and $|\sigma| = n$.

Definition 23 (*Monitor soundness*) A monitor $\mathcal{A}_?$ is sound w.r.t. $\Pi = (\phi, \varphi) \in MP^*(\mathbb{B}_4)$ on $\mathcal{P}_\Sigma$, noted $Ver(\mathcal{A}_?, \Pi, \mathcal{P}_\Sigma)$, if

$$\forall \sigma \in Exec(\mathcal{P}_\Sigma) \cap \Sigma^*, last([\![\mathcal{A}_?]\!](\sigma)) = [\![\Pi]\!]_{\mathbb{B}_4}(\sigma).$$

where $[\![\cdot]\!]_{\mathbb{B}_4}$ is defined in Definition 16.

This definition states that the verification sequence produced by $\mathcal{A}_?$ matches the evaluation function of a sequences w.r.t. an r-property.

Using the set $\mathbb{P}^{\mathcal{A}_\Pi}$ of a Streett automaton $\mathcal{A}_\Pi$, we show how it is possible to obtain a verification monitor for the r-property Π.

Definition 24 (*Streett2VM transformation*) Given a Streett m-automaton $\mathcal{A}_\Pi = (Q^{\mathcal{A}_\Pi}, q_{\text{init}}^{\mathcal{A}_\Pi}, \Sigma, \longrightarrow_{\mathcal{A}_\Pi}, \{(R_1, P_1), \ldots, (R_m, P_m)\})$ recognizing $\Pi \in MP^*(\mathbb{B}_4)$, we define the transformation Streett2VM$(\mathcal{A}_\Pi) = (Q^{\mathcal{A}_\Pi}, q_{\text{init}}^{\mathcal{A}_\Pi}, \rightarrow_{\mathcal{A}_\Pi}, \mathbb{B}_4, \Gamma)$ s.t. $\Gamma : Q^{\mathcal{A}_?} \rightarrow \mathbb{B}_4$ produces truth-values from states depending on the set $\mathbb{P}^{\mathcal{A}_\Pi}$: $\forall q \in Q^{\mathcal{A}_\Pi}$,

$$q \in Good^{\mathcal{A}_\Pi} \Rightarrow \Gamma(q) = \top, \quad q \in Good_c^{\mathcal{A}_\Pi} \Rightarrow \Gamma(q) = \top_c,$$
$$q \in Bad_c^{\mathcal{A}_\Pi} \Rightarrow \Gamma(q) = \bot_c, \quad q \in Bad^{\mathcal{A}_\Pi} \Rightarrow \Gamma(q) = \bot.$$

An r-property Π is verifiable on $\mathcal{P}_\Sigma$ by a VM $\mathcal{A}_{?\Pi}$ obtained by the application of Streett2VM on the automaton recognizing Π.

[12] The synthesis of verification monitors is presented for r-properties in $MP^*(\mathbb{B}_4)$ and can be adapted in a straightforward manner for other sets of monitorable properties using the results in Sect. 5.

Example 10 (*Verification monitors*) In Fig. 14 are represented VMs for the properties introduced in Example 1, specified by Streett automata in Fig. 3 and synthesized with the Streett2VM transformation.

Theorem 9 (Correctness of Streett2VM) *Given* $\mathcal{A}_\Pi$ *recognizing* Π*, we have:*

$$\Pi \in MP^*(\mathbb{B}_4) \wedge \mathcal{A}_{?\Pi} = \text{Streett2VM}(\mathcal{A}_\Pi) \Rightarrow Ver(\mathcal{A}_{?\Pi}, \Pi, \mathcal{P}_\Sigma).$$

Proof The proof of this theorem relies on the correctness of the computation performed while obtaining $\mathbb{P}^{\mathcal{A}_\Pi}$ for $\mathcal{A}_\Pi$ (Property 3). □

7.3 Synthesizing monitors for runtime enforcement

In the remainder, we consider enforceable r-properties (ϕ, φ) and $\Pi \in EP$, and a Streett m-automaton $\mathcal{A}_\Pi = (Q^{\mathcal{A}_\Pi}, q_{\text{init}}^{\mathcal{A}_\Pi}, \Sigma, \longrightarrow_{\mathcal{A}_\Pi}, \{(R_1, P_1), \ldots, (R_m, P_m)\})$ recognizing Π. An EM is producing enforcement operations depending on its current state.

Definition 25 (*Enforcement monitor*) An EM $\mathcal{A}_!$ is a 5-tuple $(Q^{\mathcal{A}_!}, q_{\text{init}}^{\mathcal{A}_!}, \longrightarrow_{\mathcal{A}_!}, Ops, \Gamma)$. Enforcement operations of Ops performed by the EM are aimed to operate a modification of the internal memory and potentially produce an output, i.e., each enforcement operation is a function: $\Sigma \times \Sigma^* \to \Sigma^* \times \Sigma^*$. Then $\Gamma : Q^{\mathcal{A}_!} \to Ops$ is the output function, producing enforcement operations from states.

The considered enforcement operations allow enforcement monitors either to halt the target program (when the current input sequence irreparably violates the property), or to store the current event in a *memory device* (when a decision has to be postponed), or to dump the content of the memory device (when the target program went back to a correct behavior), or to switch off the monitor when all possible continuations of the current input sequence are correct w.r.t. the property under scrutiny.[13]

Definition 26 (*Enforcement operations*) We define a set of enforcement operations $Ops = \{halt, store, dump, off\}$ as follows: $\forall e \in \Sigma \cup \{\epsilon_\Sigma\}, \forall m \in \Sigma^*$,

$$halt(e, m) = (\epsilon_\Sigma, m), \quad store(e, m) = (\epsilon_\Sigma, m \cdot e),$$
$$dump(e, m) = (m \cdot e, \epsilon_\Sigma), \quad off(e, m) = (m \cdot e, \epsilon_\Sigma).$$

where e designates the input event of the monitor and m the memory device content.

[13] Although *dump* and *off* have the same definition, distinguishing them is useful in practice. Indeed, the *off* operation is intended to be produced when all continuations of the current execution sequence are correct w.r.t. the property. It allows to determine when the EM is not needed anymore. Consequently, it helps reducing the performance impact on the underlying program.

Note that the *off* and *dump* operations have the same definitions. From a theoretical perspective, the *off* is indeed not necessary. However, it has a practical interest. In order to limit the monitor's impact on the original program (performance wise), it is of interest to know when the monitor is not needed anymore.

We define the transformation performed by an EM $\mathcal{A}_!$ while reading an input sequence $\sigma \in \Sigma^*$ and producing an output sequence $o \in \Sigma^*$.

Definition 27 (*Sequence transformation*) The sequence transformation function $[\![\mathcal{A}_!]\!](\cdot) : \Sigma^* \to \Sigma^*$ relies on the function $[\![\mathcal{A}_!]\!](\cdot, \cdot, \cdot) : \Sigma^* \times Q^{\mathcal{A}_!} \times \Sigma^* \to \Sigma^*$ defining the transformation performed on the current state and the internal memory content: $[\![\mathcal{A}_!]\!](\sigma, q, m)$ is the output sequence produced while reading σ from state q and (initial) memory content m.

$$\forall q \in Q^{\mathcal{A}_!}, \forall m \in \Sigma^*, [\![\mathcal{A}_!]\!](\epsilon_\Sigma, q, m) = \epsilon_\Sigma \tag{7}$$
$$[\![\mathcal{A}_!]\!](e \cdot \sigma, q, m) = o \cdot [\![\mathcal{A}_!]\!](\sigma, q', m') \tag{8}$$

with $q \xrightarrow{e}_{\mathcal{A}_!} q' \wedge \Gamma(q') = \alpha \in Ops \wedge \alpha(e, m) = (o, m')$.

The empty sequence ϵ_Σ is transformed into itself by $\mathcal{A}_!$, this is the case when the underlying program does not produce any event (7). An execution sequence $e \cdot \sigma$ is (incrementally) transformed according to the transition fired by the input e: the current memory content and the input e are applied to the enforcement operation of the arriving-state transition; it induces a new memory content and an output o (8).

We define now the notion of property-enforcement by an EM. The notion of enforcement relates the input sequence produced by the program and given to the EM and the output sequence allowed by the EM (correct w.r.t. the property under consideration).[14]

Definition 28 (*Property-enforcement*) For $\Pi = (\phi, \varphi) \in EP$, we say that $\mathcal{A}_!$ enforces Π on $\mathcal{P}_\Sigma$, noted $Enf(\mathcal{A}_!, \Pi, \mathcal{P}_\Sigma)$, iff for any $\sigma \in Exec(\mathcal{P}_\Sigma) \cap \Sigma^*$, there exists $o \in \Sigma^\infty$, s.t. the following constraints hold:

$$[\![\mathcal{A}_!]\!](\sigma, q_{\text{init}}^{\mathcal{A}_!}, \epsilon) = o \tag{9}$$
$$\Pi(\sigma) \Rightarrow \sigma = o \tag{10}$$
$$\neg\Pi(\sigma) \wedge Pref_\prec(\phi, \sigma) = \emptyset \Rightarrow o = \epsilon \tag{11}$$
$$\neg\Pi(\sigma) \wedge Pref_\prec(\phi, \sigma) \neq \emptyset \Rightarrow o = Max(Pref_\prec(\phi, \sigma)) \tag{12}$$

(9)–(12) ensure soundness and transparency of $\mathcal{A}_!$: (9) stipulates that the sequence σ is transformed by $\mathcal{A}_!$ into a sequence o; (10) ensures that if σ already satisfy the property then it

[14] In the general case, the comparison between input and output sequences is performed up to some equivalence relation $\approx \subseteq \Sigma^\infty \times \Sigma^\infty$. Note that the considered equivalence relation should preserve the r-property under consideration.

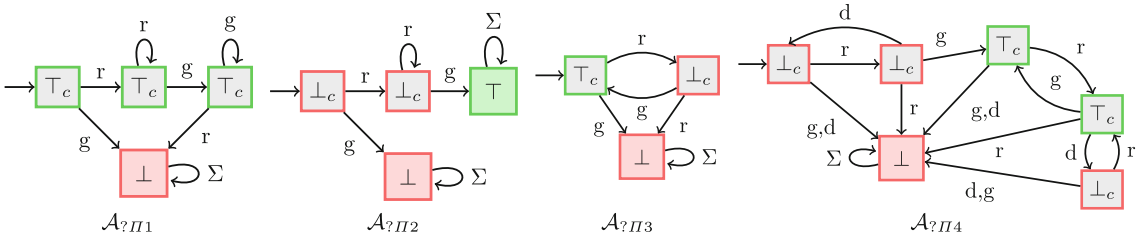

Fig. 14 Examples of verification monitors

is not transformed. (11) ensures that, when no prefix of σ satisfies the property, the EM outputs nothing (the empty sequence ϵ_Σ). (12) ensures that, if some prefix of σ satisfies the property, o is the longest prefix of σ satisfying the property.

Soundness comes from the fact that the produced sequence o, when different from ϵ_Σ, always satisfies the property ϕ. Transparency is ensured by the fact that correct execution sequences are not changed, and incorrect ones are restricted to their longest correct prefix.

One may remark that it would have been possible to set $Max(Pref_\prec(\phi, \sigma))$ to ϵ_Σ when $Pref_\prec(\phi, \sigma) = \emptyset$ and merge the two last constraints. However, we choose to distinguish explicitly the case in which $Pref_\prec(\phi, \sigma) = \emptyset$ as it highlights some differences when an EM produces ϵ_Σ. Sometimes it corresponds to the only correct prefix of the property. But it can also be an incorrect sequence w.r.t. the property. In practice, when implementing an EM for a system, this sequence can be "tagged" as incorrect.[15]

Finally, since we have to deal with potentially infinite input sequences, the output sequence should be produced in an incremental way[16]: for each current prefix σ of the input sequence read by the EM, the *current* output o produced should be sound and transparent with respect to Π and σ. This means that deciding whether a finite sequence σ verifies Π or not should be computable in a finite amount of time (and reading only a finite continuation of σ).

We synthesize EMs from Streett automata in the framework of r-properties. This transformation was previously introduced in the form of several transformations [15,18]. Here, we generalize those transformations into a unique one.

Definition 29 (*Streett2EM transformation*) The transformation $\text{Streett2EM}(\mathcal{A}_\Pi) = (Q^{\mathcal{A}_\Pi}, q_{\text{init}}^{\mathcal{A}_\Pi}, \longrightarrow_{\mathcal{A}_\Pi}, Ops, \Gamma)$ is defined s.t. Γ: $Q^{\mathcal{A}_{!\Pi}} \rightarrow Ops$ produces enforcement operations: $\forall q \in \mathcal{A}_\Pi$,

$q \in Good^{\mathcal{A}_\Pi} \Rightarrow \Gamma(q) = off$; $q \in Good_c^{\mathcal{A}_\Pi} \Rightarrow \Gamma(q) = dump$;
$q \in Bad_c^{\mathcal{A}_\Pi} \Rightarrow \Gamma(q) = store$; $q \in Bad^{\mathcal{A}_\Pi} \Rightarrow \Gamma(q) = halt$.

Example 11 (*Enforcement monitors*) Figure 15 represents EMs for the properties introduced in Example 1, specified by Streett automata in Fig. 3, and synthesized with the Streett2EM transformation.

An r-property $\Pi \in EP$ is enforceable on $\mathcal{P}_\Sigma$ by an EM obtained by the application of Streett2EM on the automaton recognizing Π.

Theorem 10 (Correctness of Streett2EM) *Given $\mathcal{A}_\Pi$ recognizing $\Pi \in EP$, we have:*

$\mathcal{A}_{!\Pi} = \text{Streett2EM}(\mathcal{A}_\Pi) \Rightarrow Enf(\mathcal{A}_{!\Pi}, \Pi, \mathcal{P}_\Sigma)$.

Proof Correctness of the general transformation relies on the correctness (proved in [15]) of the transformations specific to each class of properties. Indeed, this general transformation reduces to the specific transformation when applied to a specific class of properties. □

Such a unified transformation is useful in practice (from an implementation point of view) as it can be applied to any Streett automaton (regardless of the class of the recognized property).

7.4 Discussion

One of the important challenges in runtime verification is the practical interest of the specification formalism. An ideal specification language should be easy to use by end-users. Moreover, a desirable feature, advocated by this paper, is the need for addressing both infinite and finite execution sequences.

One could reproach the two following facts to the proposed synthesis approach:

- First, several mechanisms are needed: DFAs, Streett automata, and finally verification and enforcement monitors.
- Second, one may question the usefulness of the DFA to Streett transformations. Arguably, it would be possible to generate monitors directly for properties specified by Streett automata (written by hand or generated from temporal logic formula).

[15] This latter case is avoided in [28] by assuming that properties under consideration always contain ϵ_Σ.

[16] From a more general perspective, we can see this limitation from a runtime verification point of view. Verifying infinitary properties at runtime on a produced execution sequence, in essence, should be done by checking finite prefixes of the current sequence.

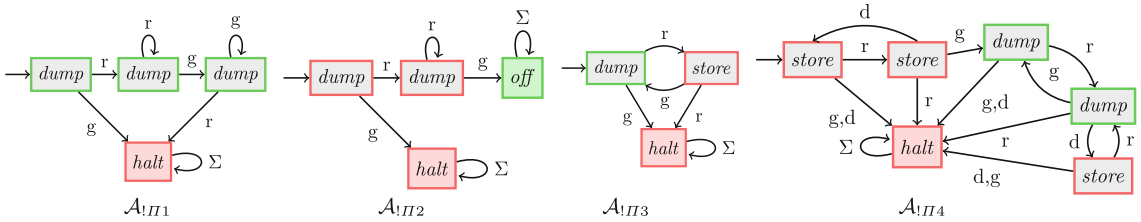

Fig. 15 Examples of enforcement monitors

In our point of view, using the DFA2Streett transformations in the synthesis process has the three following advantages:

- The DFA2Streett transformations complete the picture of the Safety-Progress classification, by providing constructive tools in the automata view that are the counterparts of the language-theoretic operators (A, E, R, P, A_f, E_f, R_f, P_f) in the language view.
- The direct translation from a supposed ideal specification formalism to a Streett automaton would be difficult. This formalism would have to address both finite and infinite behaviors, requiring a design expertise. It is likely that such a translation would be error-prone or lead to ambiguous specifications.
- One of the underlying arguments for using our approach is that the end-user is only required to specify a finite behavior and indicate the wished pattern (used with the finitary property). Thus, the infinite behavior is comprehended by the user by only seeing patterns like "always", "at least once", "regularly", or "persistently". The user is thus kept from specifying the infinite behavior with Streett automata.

A complex translation is avoided, replaced by two simpler transformations, and the required work from the user seems to be simpler.

8 Conclusion and future work

Conclusion We have extended the Safety-Progress classification of properties in a runtime validation context. This hierarchical organization of properties turned out to be a convenient framework for specifying properties purposed to be used at runtime. We addressed in a unified way the problem of monitorability and enforceability of properties at runtime using this general framework. We provided the first exact characterization of monitorable properties according to the classical definition of monitorability proposed by Pnueli and Zaks. We generalized the classical definition of monitorability by parameterizing it by a truth-domain. We introduced a new definition of monitorability based on distinguishability of good and bad execution sequences. This definition is based on positive and negative determinacy as well. However, we believe that it better corresponds to practical needs and tool implementations and fits better in the hierarchy of properties. Moreover, this alternative definition is able to better distinguish equivocal situations that a monitor would have to face off without a finite sequence interpretation. Furthermore, we have delineated the set of enforceable properties w.r.t. the SP classification. This set of properties was characterized independently from any enforcement mechanism. It is thus an upper-bound for the set of properties that could be addressed by any enforcement mechanism. Furthermore, we have given general synthesis procedures to generate runtime and enforcement monitors in this framework.

Future work 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, as it was initiated in [14]. Note that the definitions of positive and negative determinacy is rather appropriate in this context. Indeed, in a test campaign, one is concerned with the set of all execution sequences that can be produced by the underlying program. Notions of positive and negative determinacy seem to be a first approximation of the set of possible future execution sequences of a program.

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 set of properties for which other runtime-verification derived techniques can be applied (e.g., runtime reflection [26]).

Another research perspective is to relax the soundness and transparency constraints or consider different definitions of runtime enforcement. Specific enforcement mechanisms shall be designed. For instance, augmented enforcers may enjoy more handling abilities on the sequences submitted in input. It seems interesting to see the impact on the set of enforceable properties.

Furthermore, an interesting question would be to investigate how the automata-view of the SP classification transposes to other sets of properties such as context-free languages. The classification used in this paper focuses on regular properties. In the quest of expressiveness for specification languages, relying on a classification appears as a good way to delineate monitorable and enforceable properties.

Finally, the question of expressiveness is somehow also related to *parametric properties*, i.e., the properties expressed using events with formal parameters whose concrete values are obtained at runtime. Lots of runtime verification frameworks now handle parametric properties. Among them, two kinds of approaches can be distinguished in state-of-the-art frameworks:

- RuleR [2] is able to handle expressive parametric specifications thanks to a general approach based on rule rewriting,
- MOP [8] has been recently leveraged with the so-called slicing approach to monitor some form of parametric specification in a more efficient way.

It is rather clear that being able to express parametric properties is an asset in runtime verification and is surely desirable from a practical point of view. Now, as runtime verification is always concerned with efficiency, one question is to find the right balance between the gained expressiveness and the induced overhead, i.e., a runtime verification framework should be chosen so as to meet the just needed level of expressiveness to specify the requirements. Moreover, another related question is to investigate whether it is more efficient to enhance expressiveness by considering parametric properties or using a more expressive specification formalism.

Acknowledgments The authors of the paper would like to gratefully thank Howard Barringer, Klaus Havelund, Thierry Jéron, Hervé Marchand, and the anonymous reviewers for their helpful remarks.

Appendix A: Proofs

A.1 Proofs for Sect. 4

A.1.1 Proof of Property 2: Closure of r-properties

We prove the two facts in order:

1. We have either $\phi(\sigma)$ or $\varphi(\sigma)$, i.e., all prefixes σ' of σ belong to ψ. Necessarily, all prefixes σ'' of σ' also belong to ψ, i.e., $\psi(\sigma'')$. By definition, that means $\sigma' \in A_f(\psi)$, i.e., $\phi(\sigma')$ and $\Pi(\sigma')$.
2. $\Pi(\sigma)$ implies that σ has at least one prefix $\sigma_0 \preceq \sigma$ belonging to ψ: $\sigma \in E_f(\psi)$. Then any continuation of σ built using any finite or infinite sequence σ' has at least the same prefix belonging to ψ. If $\sigma' \in \Sigma^*$, we have $\sigma_0 \preceq \sigma \preceq \sigma \cdot \sigma'$ and $\sigma \cdot \sigma' \in E_f(\psi)$. If $\sigma' \in \Sigma^\omega$, we have $\sigma_0 \preceq \sigma \prec \sigma \cdot \sigma'$ and $\sigma \cdot \sigma' \in E(\psi)$. □

A.1.2 Proof of Theorem 1: Soundness of the transformations of DFAs to Streett automata

Considering a DFA $\mathcal{A}_\psi = (Q^{\mathcal{A}_\psi}, q_{\text{init}}^{\mathcal{A}_\psi}, \rightarrow_{\mathcal{A}_\psi}, F)$ (we omit the superscript of F for the sake of clarity), s.t. $\mathcal{L}(\mathcal{A}_\psi) = \psi$, we have to prove that:

$$\mathcal{A}_\Pi = \mathit{DFA2S\_Saf}(\mathcal{A}_\psi) \Rightarrow \mathcal{L}(\mathcal{A}_\Pi) = (A_f(\psi), A(\psi))$$
$$\mathcal{A}_\Pi = \mathit{DFA2S\_Guar}(\mathcal{A}_\psi) \Rightarrow \mathcal{L}(\mathcal{A}_\Pi) = (E_f(\psi), E(\psi))$$
$$\mathcal{A}_\Pi = \mathit{DFA2S\_Resp}(\mathcal{A}_\psi) \Rightarrow \mathcal{L}(\mathcal{A}_\Pi) = (R_f(\psi), R(\psi))$$
$$\mathcal{A}_\Pi = \mathit{DFA2S\_Per}(\mathcal{A}_\psi) \Rightarrow \mathcal{L}(\mathcal{A}_\Pi) = (P_f(\psi), P(\psi))$$

In the following proofs, for a finite sequence σ of length n, we may use the notion of run of σ on $\mathcal{A}_\psi$ or on a Streett automaton $\mathcal{A}_\Pi$ obtained by the transformations. We note:

- $\mathit{run}(\sigma, \mathcal{A}_\psi) = q_0 \cdots q_n$,
- $\mathit{run}(\sigma, \mathcal{A}_\Pi) = q'_0 \cdots q'_n$.

For Safety properties. We show that the r-property accepted by $\mathcal{A}_\Pi$ (obtained using *DFA2S_Saf*) is exactly $(A_f(\psi), A(\psi))$.

Let $\sigma \in \Sigma^\infty$ s.t. $(A_f(\psi), A(\psi))(\sigma)$, let us prove that the sequence σ is accepted by $\mathcal{A}_\Pi$. We have two cases: σ is a finite sequence or not.

- Let us consider $\sigma \in \Sigma^*$ s.t. $|\sigma| = n$, then by definition of r-properties: $\sigma \in A_f(\psi)$, i.e., every prefix of σ belongs to ψ. Let us examine $\mathit{run}(\sigma, \mathcal{A}_\psi) = q_0 \cdots q_n$. As $\mathcal{L}(\mathcal{A}_\psi) = \psi$, we have $\forall i \in [0, n], q_i \in F$. By definition of the transformation DFA2S_Saf, we have $\forall i \in [0, n], q_i \in P$. According to (TSAFE1), we have $\mathit{run}(\sigma, \mathcal{A}_\Pi) = q_0 \cdots q_n$. Using the acceptance criterion for finite sequences, σ is accepted by $\mathcal{A}_\Pi$.
- Let $\sigma \in \Sigma^\omega$, then by definition of r-properties: $\sigma \in A(\psi)$, i.e., every finite prefix of σ belongs to ψ. Let us suppose that σ is not accepted by $\mathcal{A}_\Pi$. According to the acceptance criterion for infinite sequences (Definition 6), we would have $\mathit{vinf}(\sigma, \mathcal{A}_\Pi) \not\subseteq P$ (as $\mathcal{A}_\Pi$ is a safety automaton, $R = \emptyset$). By definition of the transformation *DFA2S_Saf* and the shape of the obtained automaton $\mathcal{A}_\Pi$, we have $\mathit{vinf}(\sigma, \mathcal{A}_\Pi) = \{\mathit{sink}\}$. Using (TSAFE2), we know that there exists a smallest prefix σ' of σ, s.t. the run of σ' on $\mathcal{A}_\Pi$ reaches the state *sink*. By the definition of *DFA2S_Saf*, we can deduce that the run of σ' on $\mathcal{A}_\psi$ ends in a state in $\overline{F}$. As $\mathcal{L}(\mathcal{A}_\psi) = \psi, \sigma' \notin \psi$. We obtain a contradiction with $\sigma \in A(\psi)$, and σ is actually accepted by $\mathcal{A}_\Pi$.

Let σ be a sequence accepted by $\mathcal{A}_\Pi$, let us prove that $\sigma \in (A_f(\psi), A(\psi))$. We distinguish again two cases: σ is a finite sequence or not.

- Let $\sigma \in \Sigma^*$ s.t. $|\sigma| = n$, then by definition of the acceptance criterion for finite sequences of Streett automata (Definition 7), we have $q'_n \in P$. As $\mathcal{A}_\Pi$ is a safety automaton, we can deduce that $\forall i \in [0, n], q'_i \in P$. Following the definition of *DFA2S_Saf*, we find that all the states visited during the run of σ on $\mathcal{A}_\psi$ are in F: $\forall i \in [0, n], q_i \in F$ (and $q_i = q'_i$). By definition of the acceptance criterion for DFAs, we can deduce that every prefix of σ is accepted by $\mathcal{A}_\psi$. As $\mathcal{L}(\mathcal{A}_\psi) = \psi$, we can deduce that all prefixes of σ belong to ψ, i.e., $\sigma \in A_f(\psi)$.
- If $\sigma \in \Sigma^\omega$, then by definition of the acceptance criterion for infinite sequences (Definition 6), we know that $\mathit{vinf}(\sigma, \mathcal{A}_\Pi) \subseteq P$. Let us suppose that $\sigma \notin A(\psi)$, by definition of the operator A (see Definition 2), there exists a strict prefix σ' of σ not belonging to ψ. Let $n' = |\sigma'|$. As $\mathcal{L}(\mathcal{A}_\psi) = \psi$, the run of σ' on $\mathcal{A}_\psi$, $\mathit{run}(\sigma', \mathcal{A}_\psi) = q_0 \cdots q_{n'}$, satisfies $q_0 = q_{\text{init}}^{\mathcal{A}_\psi} \wedge q_{n'} \notin F$. According to the definition of the transformation *DFA2S_Saf* and the rule (TSafe2), we have $q'_{n'} = \mathit{sink} \notin P$. Furthermore, using (TSafe3), every continuation of σ' has its run ending in *sink*. We deduce that $\mathit{vinf}(\sigma, \mathcal{A}_\Pi) = \{\mathit{sink}\} \not\subseteq P$. Which is a contradiction with the initial hypothesis, and gives us $\sigma \in A(\psi)$. □

For Guarantee properties We show that the sets of sequences accepted by $\mathcal{A}_\Pi$ obtained by *DFA2S_Guar* are exactly $(E_f(\psi), E(\psi))$.

Let $\sigma \in \Sigma^\infty$ s.t. $(E_f(\psi), E(\psi))(\sigma)$, let us prove that the sequence σ is accepted by $\mathcal{A}_\Pi$. We have two subcases: σ is a finite sequence or not.

- Let us consider $\sigma \in \Sigma^*$ s.t. $|\sigma| = n$, then by definition of r-properties: $\sigma \in E_f(\psi)$, i.e., σ has at least one prefix which belongs to ψ. Let us consider $S_{sat} = \{\sigma' \in \Sigma^* \mid \sigma' \preceq \sigma \wedge \sigma' \in \psi\}$, the set of prefixes of σ which belong to ψ. As $\sigma \in E_f(\psi)$, we can deduce that $S_{sat} \neq \emptyset$, S_{sat} has thus a smallest element σ_{min}. Let $n' = |\sigma_{min}|$. We have, by definition of σ_{min}, $\forall \sigma' \in \Sigma^*, \sigma' \prec \sigma_{min} \Rightarrow \sigma' \notin \psi$. Let us examine $\mathit{run}(\sigma_{min}, \mathcal{A}_\psi) = q_0 \cdots q_{n'}$. As $\mathcal{L}(\mathcal{A}_\psi) = \psi$, we have $\forall i \in [0, n' - 1], q_i \notin F \wedge q_{n'} \in F$. According to (TGuar2), we have $\mathit{run}(\sigma_{min}, \mathcal{A}_\Pi) = q_0 \cdots q_{n'}$ with $\forall i \in [0, n'-1], q_i \notin R \wedge q_{n'} \in R$. Following (TGuar1), we have $\forall i \in [n', n], q_i \in R$. According to the acceptance criterion for finite sequences, σ is accepted by $\mathcal{A}_\Pi$.
- Let $\sigma \in \Sigma^\omega$, then by definition of r-properties: $\sigma \in E(\psi)$, i.e., (at least) one finite prefix of σ belongs to ψ. Let us suppose that σ is not accepted by $\mathcal{A}_\Pi$. According to the acceptance criterion for infinite sequences (Definition 6), we have $\mathit{vinf}(\sigma, \mathcal{A}_\Pi) \cap R = \emptyset$ (as $\mathcal{A}_\Pi$ is a guarantee automaton, $P = \emptyset$). In other words, we have $\mathit{vinf}(\sigma, \mathcal{A}_\Pi) \subseteq \overline{R}$. As $\mathcal{A}_\Pi$ is a guarantee automaton, every state visited by the run of σ on $\mathcal{A}_\Pi$ is in $\overline{R}$. Indeed, according to the shape of the transition function of guarantee automata, if a state of R is visited, we have $\mathit{vinf}(\sigma, \mathcal{A}_\Pi) \cap R \neq \emptyset$. Let us consider now the prefixes of σ. During the run of these prefixes on $\mathcal{A}_\Pi$, none of them visits an R-state. It follows that, according to (TGuar2), none of the runs on $\mathcal{A}_\Pi$ of these prefixes visits a state in F. As $\mathcal{L}(\mathcal{A}_\psi) = \psi$, we deduce that none of the prefixes of σ belongs to ψ. We obtain a contradiction with $\sigma \in E(\psi)$, and consequently σ is actually accepted by $\mathcal{A}_\Pi$.

Let σ be a sequence accepted by $\mathcal{A}_\Pi$, let us prove that $\sigma \in (E_f(\psi), E(\psi))$. We distinguish two cases: σ is a finite sequence or not.

- Let $\sigma \in \Sigma^*$ s.t. $|\sigma| = n$, then by definition of the acceptance criterion for finite sequences (Definition 7), we have $q_n \in R$. Let us suppose that $\sigma \notin E_f(\psi)$, i.e., none of the prefixes of σ belongs to ψ. As $\mathcal{L}(\mathcal{A}_\psi) = \psi$, the run of σ on $\mathcal{A}_\psi$ satisfies: $\forall i \in [0, n], q_i \notin F$. Starting from $q_{\text{init}}^{\mathcal{A}_\psi} = q_{\text{init}}^{\mathcal{A}_\Pi} \notin R$, and using (TGuar2), we find that $\mathit{run}(\sigma, \mathcal{A}_\Pi) = q'_0 \cdots q'_n$ with $\forall i \in [0, n], q'_i \notin R$. This is a contradiction with $q'_n \in R$, and thus $\sigma \in E_f(\psi)$.
- Let $\sigma \in \Sigma^\omega$, then by definition of the acceptance criterion for infinite sequences (Definition 6), we have $\mathit{vinf}(\sigma, \mathcal{A}_\Pi) \cap R \neq \emptyset$. As $\mathcal{A}_\Pi$ is a guarantee automaton, it means that $\mathit{vinf}(\sigma, \mathcal{A}_\Pi) \subseteq R$. According to the shape of the transition function for guarantee automata, it means that there is a prefix σ' of σ on $\mathcal{A}_\Pi$ for which the run switches from states in $\overline{R}$ to states in R. More formally, $\exists \sigma' \in \Sigma^*, \sigma' \prec \sigma \wedge |\sigma'| = n' \wedge \forall i \in [0, n' - 1], q'_i \in \overline{R} \wedge \forall i \geq n', q'_i \in R$. Let us suppose that $\sigma \notin E(\psi)$, i.e., σ has no prefix belonging to ψ. As $\mathcal{L}(\mathcal{A}_\psi) = \psi$, the run of σ on $\mathcal{A}_\psi$ satisfies: $\forall i \in \mathbb{N}, q_i \notin F$. Similarly to the finitary case, and according to the transformation DFA2S_Guar (TGuar2), it would question the existence of σ'. We deduce that $\sigma \in E(\psi)$. □

For Response properties We show that the r-property accepted by $\mathcal{A}_\Pi$, obtained with *DFA2S_Resp* is exactly $(R_f(\psi), R(\psi))$.

Let $\sigma \in \Sigma^\infty$ s.t. $(R_f(\psi), R(\psi))(\sigma)$, let us prove that the sequence σ is accepted by $\mathcal{A}_\Pi$. We have two subcases: σ is a finite sequence or not.

- Let $\sigma \in \Sigma^*$, thus $\sigma \in R_f(\psi)$. Proving that σ is accepted by $\mathcal{A}_\Pi$ amounts to show that the run of σ on $\mathcal{A}_\Pi$, i.e., $\mathit{run}(\sigma, \mathcal{A}_\Pi)$, ends in a R-state (i.e., $q'_n \in R$). First of all, let

us remark that $\sigma \in R_f(\psi)$ gives us $\psi(\sigma)$. Furthermore, as $\mathcal{L}(\mathcal{A}_\psi) = \psi$, we can deduce that $q_n \in F$. As $\sigma \in R_f(\psi), \forall n \in \mathbb{N}, \exists \sigma' \in \Sigma^*, \sigma \prec \sigma' \wedge |\sigma'| \geq n \wedge \psi(\sigma')$ (cf. Definition 3).
Let $n' = |F|$ be the number of accepting states of $\mathcal{A}_\psi$. Now let us consider the set $S = \{\sigma'' \in \Sigma^* \mid \sigma \prec \sigma'' \wedge |\{\sigma' \in \Sigma^* \mid \sigma \prec \sigma' \prec \sigma'' \wedge \sigma' \in \psi\}| > n'\}$. This set contains the sequences which are continuations of σ and have at least n' prefixes longer than σ and belonging to ψ. As $\sigma \in R_f(\psi)$, we know that $S \neq \emptyset$, thus S has a smallest element σ_{min}. Let us examine the run of σ_{min} on $\mathcal{A}_\psi$: $run(\sigma, \mathcal{A}_\psi) = run(\sigma, \mathcal{A}_\Pi) = q_0 \cdots q_{|\sigma_{min}|} = q_0 \cdots q_n \cdots q_{|\sigma_{min}|}$. Between q_n and $q_{|\sigma_{min}|}$, there are at least $n' + 1$ accepting states. As $|F| = n'$, two states between q_n and $q_{|\sigma_{min}|}$ are identical. Moreover, we have $\forall i \in [n, |\sigma_{min}| - 1], q_i \rightarrow_{\mathcal{A}_\psi} q_{i+1}$. Which allows us to deduce, using the definition of *DFA2S_Resp* that $q_n = q'_n$ is tagged as a R-state. According to the acceptance criterion for finite sequences, σ is accepted by $\mathcal{A}_\Pi$.

– Let $\sigma \in \Sigma^\omega$, thus $\sigma \in R(\psi)$, i.e., $\forall \sigma' \in \Sigma^*, \exists \sigma'' \in \Sigma^*, \sigma' \prec \sigma'' \prec \sigma \wedge \psi(\sigma'')$ holds for σ. Let us examine the run of σ on $\mathcal{A}_\Pi$, we will show that this run visits at least one R-state infinitely often. Indeed, let us consider a prefix σ' of σ, we can find an unbounded number of σ'-continuations σ'', s.t. $\psi(\sigma'')$. Furthermore, for each of these continuations, it is possible to find an unbounded number of continuations σ''' s.t. $\psi(\sigma''')$. Using $\mathcal{L}(\mathcal{A}_\psi) = \mathcal{A}_\psi$, the runs of the sequences σ'' and the sequences σ''' on the automaton $\mathcal{A}_\psi$ end on a F-state. Using the same reasoning as the one used for finite sequences, the state on which the run of σ'' on $\mathcal{A}_\Pi$ is a R-state. Thus, we can build a series $(\sigma_i)_{i \in \mathbb{N}}$ of σ-prefixes (of strictly growing length) s.t. the run of each σ_i ends in a R-state. Thus, an infinite number of prefixes of σ go through a R-state. As $|R| \in \mathbb{N}$, there exists a state in R visited infinitely often during the run of σ on $\mathcal{A}_\Pi$. According to the acceptance criterion for infinite sequences, σ is accepted by $\mathcal{A}_\Pi$.

Let σ be a sequence accepted by $\mathcal{A}_\Pi$, let us prove that $\sigma \in (R_f(\psi), R(\psi))$. We distinguish again two cases: σ is a finite sequence or not.

– Let $\sigma \in \Sigma^*$ s.t. $|\sigma| = n$, then by definition of the acceptance criterion for finite sequences (Definition 7), we have $q'_n \in R$. According to the definition of the transformation *DFA2S_Resp*, we deduce that $q_n \in F$ and $\exists q_0, \ldots, q_l \in Q^{\mathcal{A}_\psi}, \exists e_0, \ldots, e_l \in \Sigma$,

$$\forall j \in [0, l-1], \quad q_j \xrightarrow{e_i}_{\mathcal{A}_\psi} q_{j+1} \tag{13}$$

$$\exists i \in [0, l], \quad \exists j \in [i, l-1], \quad q_j \in F \wedge q_i = q_l \wedge q_0 = q \tag{14}$$

Thus we can build a series $(\sigma^j)_{j \in \mathbb{N}}$ of σ-continuations s.t. $\forall j \in \mathbb{N}, \psi(\sigma^j)$ and σ^j is defined as $\sigma^j = \sigma \cdot e_0 \cdots e_i \cdot (e_{i+1} \cdots e_{l-1} \cdot e_0 \cdots e_i)^j$. This series exhibits strictly growing continuations of σ belonging to ψ. According to the definition of the operator R_f, we can deduce that $\sigma \in R_f(\psi)$.

– Let $\sigma \in \Sigma^\omega$, then by definition of the acceptance criterion for infinite sequences (Definition 6), we have $vinf(\sigma, \mathcal{A}_\Pi) \cap R \neq \emptyset$. Thus, σ has an infinite number of prefixes for which the run ends in a R-state. Using the definition of *DFA2S_Resp*, we know that all these prefixes are accepted by $\mathcal{A}_\psi$ (as by definition the ending state of their run is a R-state). Using $\mathcal{L}(\mathcal{A}_\psi) = \psi$, we know that all these prefixes belong to ψ and have an unrestricted number of continuations belonging to ψ. We can deduce that $\sigma \in R(\psi)$. □

For Persistence properties. We show that the set of sequences accepted by $\mathcal{A}_\Pi$, obtained with *DFA2S_Per* is exactly $(P_f(\psi), P(\psi))$. Let us remark that, according to the definition of the transformation (the transition function is not changed), we have $\forall j \in [n-1, n' + n'' - 1], q'_j \longrightarrow_{\mathcal{A}_\Pi} q'_{j+1} \wedge q_j \longrightarrow_{\mathcal{A}_\psi} q_{j+1}$. Moreover, as $Q^{\mathcal{A}_\Pi} = Q^{\mathcal{A}_\psi}$, we can merge the states q_j and q'_j visited by the runs of σ on $\mathcal{A}_\psi$ and $\mathcal{A}_\Pi$.

Let $\sigma \in \Sigma^\infty$ s.t. $(P_f(\psi), P(\psi))(\sigma)$, let us prove that the sequence σ is accepted by $\mathcal{A}_\Pi$. We have two subcases: σ is a finite sequence or not.

– Proving that σ is accepted by $\mathcal{A}_\Pi$ amounts to show that the run of σ on $\mathcal{A}_\Pi$ ends in a P-state ($q_n \in P$). First of all, let us remark that $\sigma \in P_f(\psi)$ gives us $\psi(\sigma)$. Furthermore, as $\mathcal{L}(\mathcal{A}_\psi) = \psi$, we can deduce that $q_{n-1} \in F$. As $\sigma \in P_f(\psi)$, there exist $\sigma', \mu \in \Sigma^*$ s.t. (cf. Definition 3):

$$\sigma \preceq \sigma' \wedge (\sigma' \cdot \mu^* \cdot pref(\mu)) \subseteq \psi \tag{15}$$

Let $n' = |\sigma'|$, and $n'' = |\mu|$. Then the runs of σ' and $\sigma' \cdot \mu$ on $\mathcal{A}_\Pi$ can be expressed:

$$run(\sigma', \mathcal{A}_\Pi) = q_0 \cdots q_n \cdots q_{n'}$$
$$run(\sigma' \cdot \mu, \mathcal{A}_\Pi) = q_0 \cdots q_n \cdots q_{n'} \cdot q_{n'+1} \cdots q_{n'+n''}$$

According to (15), we have $q_{n'} \in F$. We can show by induction that

$$run(\sigma \cdot \mu^*, \mathcal{A}_\Pi) = q_0 \cdots q_{n'} \cdot (q_{n'+1} \cdots q_{n'+n''})^*.$$

Moreover, we have $\forall j \in [n'+1, n'+n''], q'_j \in F$ and $q_{n'+n''} \longrightarrow_{\mathcal{A}_\Pi} q_{n'+1}$. We can deduce, following the definition of *DFA2S_Per*, that $q_n \in P$. Indeed, it is sufficient to take $l = n' + n'' - n$ and $i = n' + 1 - n$.

– In order to prove $vinf(\sigma, \mathcal{A}_\Pi) \subseteq P$, it is sufficient to see that σ can be expressed $\sigma' \cdot \mu^\omega$. From this, every prefix of σ longer than σ' satisfies ψ and has its run which stops in a F-state on $\mathcal{A}_\psi$. Thus, we exhibit a strongly connected component of F-states which are tagged as P-states by DFA2S_Per. Thus, the states visited infinitely often during the run of σ on $\mathcal{A}_\Pi$ are the states of this strongly connected component, which gives us the expected result.

Let σ be a sequence accepted by $\mathcal{A}_\Pi$, let us prove that $\sigma \in (P_f(\psi), P(\psi))$. We distinguish two subcases: σ is a finite sequence or not.

– Let $\sigma \in \Sigma^*$ s.t. $|\sigma| = n$, then by definition of the acceptance criterion for finite sequences of Streett automata (Definition 7), we have $q_n \in P$. Then there exist two cases.
  - In the first one, we have on one hand $q_n \in F$, and on the other hand $\exists n \in \mathbb{N} \setminus \{0\}, \exists q_0, \ldots, q_n \in Q^{\mathcal{A}_\psi}$ s.t.:
    – $\forall j \in [0, n-1], q_j \longrightarrow_{\mathcal{A}_\psi} q_{j+1}$, and
    – $\exists i \in [0, n-1], \forall j \in [i, n], q_j \in F \wedge q_i = q_n \wedge q_0 = q_n$

    We have $\psi(\sigma)$ since $\mathcal{L}(\mathcal{A}_\psi) = \psi$. Moreover, there exist $e_0, \ldots, e_{n-1} \in \Sigma$ s.t. $\forall j \in [0, n-1], q_j \xrightarrow{e_j}_{\mathcal{A}_\psi} q_{j+1}$. We can deduce that $\psi(\sigma \cdot e_0 \cdots e_i), \psi(\sigma \cdot e_0 \cdots e_i \cdot e_{i+1})$, ..., $\psi(\sigma \cdot e_0 \cdots e_{n-1})$. Let us note $L_p = \sigma' \cdot ((e_0 \cdots e_n)^* \cdot e_0 + (e_0 \cdots e_n)^* \cdot e_0 \cdot e_1 + \cdots + (e_0 \cdots e_n)^* \cdot e_0 \cdots e_{n-1})$. As $q_i = q_n$ ($\{q_i, \ldots, q_n\}$ is a strongly connected component), we can prove by induction that $L_p \subseteq \psi$. Furthermore, $\forall \sigma' \in \Sigma^* \cap L_p, \sigma \cdot e_0 \cdots e_i \preceq \sigma' \Rightarrow \psi(\sigma')$, which proves that $\sigma \in P_f(\psi)$. Indeed, it is sufficient to take $\sigma' = \sigma \cdot e_0 \cdots e_i$, and $\mu = e_{i+1} \cdots e_{n-1}$.
  - In the second one, we have $q_n \in F$ and $q_n \longrightarrow_{\mathcal{A}_\psi} q_n$. Thus, $\exists e \in \Sigma, q_n \xrightarrow{e}_{\mathcal{A}_\psi} q_n$. We deduce that $\psi(\sigma)$ and $\sigma \cdot e^* \subseteq \psi$, as $\mathcal{L}(\mathcal{A}_\psi) = \psi$, which allows us to deduce easily that $\sigma \in P_f(\psi)$.

– Let $\sigma \in \Sigma^\omega$, then by definition of the acceptance criterion for infinite sequences of Streett automata (Definition 6), we have $vinf(\sigma, \mathcal{A}_\Pi) \subseteq P$. That is to say, all prefixes of σ from a certain point on have their run which ends in a P-state. As the automaton $\mathcal{A}_\Pi$ has a finite number of states, it means that there exists a strongly connected component C, s.t. the run of σ on $\mathcal{A}_\Pi$ "stays in". More formally, $\exists n, m \in \mathbb{N}, C = \{q'_0, \ldots, q'_n\} \subseteq Q^{\mathcal{A}_\Pi} \wedge run(\sigma, \mathcal{A}_\Pi) = q_0 \cdots q_m \cdots \wedge \forall i > m, q_i \in C$. Moreover, as $\{q'_0, \ldots q'_n\}$ is a SCC, from every state of C it is possible to reach any state of C. Let us suppose, without loss of generality, that $q'_0 \xrightarrow{e_0}_{\mathcal{A}_\Pi} q'_1 \xrightarrow{e_1}_{\mathcal{A}_\Pi} \cdots \xrightarrow{e_{n-1}}_{\mathcal{A}_\Pi} q'_n \xrightarrow{e_n}_{\mathcal{A}_\Pi} q'_0$, with $e_0, \ldots, e_n \in \Sigma$. According to the definition of *DFA2S_Per*, we have the same transitions on $\mathcal{A}_\psi$, i.e., $q'_0 \xrightarrow{e_0}_{\mathcal{A}_\psi} q'_1 \xrightarrow{e_1}_{\mathcal{A}_\psi} \cdots \xrightarrow{e_{n-1}}_{\mathcal{A}_\psi} q'_n \xrightarrow{e_n}_{\mathcal{A}_\psi} q'_0$.
  Let us note $L_p = \sigma' \cdot (e_0 \cdots e_n)^* \cdot (e_0 + e_0 \cdot e_1 + \cdots + e_0 \cdots e_{n-1}) = \sigma' \cdot (e_0 \cdots e_n) \cdot pref(e_0 \cdots e_{n-1})$. The sequence σ can be expressed $\sigma' \cdot (e_0 \cdots e_n)^\omega$ with the fact that for every sequence $\sigma'' \in L_p$ which is a continuation of σ', the run of σ'' ends in a P-state. This implies that the runs of these same sequences σ'' on $\mathcal{A}_\psi$ end in a F-state. As $\mathcal{L}(\mathcal{A}_\psi) = \psi$, we deduce that $\forall \sigma'' \in L_p, \sigma' \preceq \sigma'' \prec \sigma \Rightarrow \psi(\sigma'')$.
  This allows to deduce, using the definition of the operator P (see Sect. 4.2), that $\sigma \in P(\psi)$. □

A.2 Proofs for Sect. 5

A.2.1 Proof of Lemma 2: Closure of monitorable properties under boolean operations

Let us consider two r-properties $\Pi_1, \Pi_2 \in MP_c$.

– Proof of $\Pi_1 \wedge \Pi_2 \in MP_c$. It consists in showing that $\Pi_1 \wedge \Pi_2$ is σ-monitorable for any sequence $\sigma \in \Sigma^*$. Let $\sigma \in \Sigma^*$; let us exhibit an extension $\mu \in \Sigma^*$ s.t. $\Pi_1 \wedge \Pi_2$ is negatively or positively determined by $\sigma \cdot \mu$.
  As Π_1 is monitorable, there exists μ_1 s.t. Π_1 is negatively or positively determined by $\sigma \cdot \mu_1$, i.e., we have the two following subcases:
  - $\exists \mu_1 \in \Sigma^*, \forall \mu'_1 \in \Sigma^\infty, \neg\Pi_1(\sigma \cdot \mu_1 \cdot \mu'_1)$, Π_1 is negatively determined by $\sigma \cdot \mu_1$, or,
  - $\exists \mu_1 \in \Sigma^*, \forall \mu'_1 \in \Sigma^\infty, \Pi_1(\sigma \cdot \mu_1 \cdot \mu'_1)$, Π_1 is positively determined by $\sigma \cdot \mu_1$.

  As Π_2 is also monitorable, it is $\sigma \cdot \mu_1$-monitorable, there exists μ_2 s.t. Π_2 is negatively or positively determined by $\sigma \cdot \mu_1 \cdot \mu_2$, i.e., we have the two following subcases:
  - $\exists \mu_2 \in \Sigma^*, \forall \mu'_2 \in \Sigma^\infty, \neg\Pi_2(\sigma \cdot \mu_1 \cdot \mu_2 \cdot \mu'_2)$, Π_2 is negatively determined by $\sigma \cdot \mu_1 \cdot \mu_2$, or,
  - $\exists \mu_2 \in \Sigma^*, \forall \mu'_2 \in \Sigma^\infty, \Pi_2(\sigma \cdot \mu_1 \cdot \mu_2 \cdot \mu'_2)$, Π_2 is positively determined by $\sigma \cdot \mu_1 \cdot \mu_2$.

  By combination, there exist four cases depending on the facts that Π_1 is positively or negatively determined by $\sigma \cdot \mu_1$ and Π_2 is negatively or positively determined by $\sigma \cdot \mu_1 \cdot \mu_2$. We group them into two cases:
  - Let us distinguish the case where Π_1 is positively determined by $\sigma \cdot \mu_1$ and Π_2 is positively determined by $\sigma \cdot \mu_1 \cdot \mu_2$. Then, by taking $\mu = \sigma \cdot \mu_1 \cdot \mu_2$, we have that $\Pi_1 \wedge \Pi_2$ is positively determined by μ. This gives us the expected result.
  - In the other cases, it suffices to take $\mu = \sigma \cdot \mu_1 \cdot \mu_2$ to show that $\Pi_1 \wedge \Pi_2$ is negatively determined by μ.

– The proof of $\Pi_1 \vee \Pi_2 \in MP_c$ is similar.
– The proof of $\neg\Pi_1 \in MP_c$ is straightforward by noticing that for any sequence $\sigma \in \Sigma^*$, if Π_1 is positively (resp. negatively) determined by σ, then $\neg\Pi$ is negatively (resp. positively) determined by σ. □

A.2.2 Proof of Theorem 2: Obligation$(\Sigma) \subset MP_c$

The set of obligation r-properties is the set of all k-obligation r-properties for $k \in \mathbb{N}$, where a k-obligation is expressed as follows (Lemma 1):

$$\bigcap_{i=1}^{k} (\text{Safety}_i \cup \text{Guarantee}_i),$$

where Safety_i and Guarantee_i are safety and guarantee r-properties.

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

– Safety and guarantee properties are monitorable. Here is the proof[17]:
  – Let $\Pi = (A_f(\psi), A(\psi))$ be a safety r-property, and $\sigma \in \Sigma^*$. The proof is done by distinguishing two cases: either there exists a continuation $\sigma' \in \Sigma^*$ of σ s.t. $\neg\Pi(\sigma')$, or there does not exist. In the first case, we have $\neg A_f(\psi)(\sigma')$, i.e., σ' does not have all of its prefixes in ψ. Then the same holds for every continuation σ'' of σ': $\forall \sigma'' \in \Sigma^*, \sigma' \preceq \sigma'' \Rightarrow \neg A_f(\psi)(\sigma'')$. It follows that $\forall \sigma'' \in \Sigma^*, \sigma \preceq \sigma' \preceq \sigma'' \Rightarrow \neg\Pi(\sigma'')$. That is to say, Π is negatively determined by σ'. In the second case, every continuation of σ satisfies Π, i.e., Π is positively determined by $\sigma \cdot \epsilon$.
  – Let $\Pi = (E_f(\psi), E(\psi))$ be a guarantee r-property, let us prove that Π is monitorable. The proof can be similarly conducted. It suffices to consider $\sigma \in \Sigma^*$ and show that there exists a σ-continuation which makes that Π is negatively or positively determined by this continuation. Similarly, two cases can be distinguished whether there exists a σ-continuation which satisfies the property.
– Union and intersection of two monitorable properties are monitorable (Lemma 2).
– Example 6 shows that the inclusion is strict. □

[17] The proof can also be done by examining the syntactic restriction applying to an automaton recognizing a safety or a guarantee property: for all $\sigma \in \Sigma^*$, there exists a continuation μ s.t. this property is negatively or positively determined by $\sigma \cdot \mu$. For instance, in a safety automaton, for each state there exists a path which leads either to a terminal strongly connected component of states in which the property is satisfied or in a terminal strongly connected component in which the property is not satisfied.

A.2.3 Proof of Lemma 3: MP$^*(\mathbb{B}_3)$*, safety, and guarantee properties*

We prove this property by *reductio ad absurdum*. Let suppose the existence of a reactivity r-property $\Pi = (\phi, \varphi)$ defined on Σ which is neither a safety nor a guarantee: $\Pi \in \text{Reactivity}(\Sigma) \setminus (\text{Safety}(\Sigma) \cup \text{Guarantee}(\Sigma))$, and which is monitorable according to Definition 17 with $\mathbb{B}_3$.

As $\Pi \in MP^*(\mathbb{B}_3)$, by definition we have:

$$\forall \sigma_{good} \in \phi, \quad \forall \sigma_{bad} \in \overline{\phi}, \quad [\![\Pi]\!]_{\mathbb{B}_3}(\sigma_{good}) \neq [\![\Pi]\!]_{\mathbb{B}_3}(\sigma_{bad})$$

Let us remark that $\phi \neq \emptyset$ and $\overline{\phi} \neq \emptyset$ as Π is neither a safety nor a guarantee. Indeed, if $\phi = \emptyset$, then Π would be necessarily the r-property false, which is a safety. Likewise, if $\overline{\phi} = \emptyset$, i.e., $\phi = \Sigma^*$, Π would be the r-property true which is a safety as well.

Then we consider two sequences σ_{good} and σ_{bad} in Σ^∞:

– Let $\sigma_{good} \in \phi$ s.t. there exists $\sigma'_g \in \Sigma^\infty$ with $\neg\Pi(\sigma_{good} \cdot \sigma'_g)$. We know that such a sequence exists since $\Pi \notin \text{Guarantee}(\Sigma)$. This is a consequence of Property 2.
– Let $\sigma_{bad} \in \overline{\phi}$ s.t. there exists $\sigma'_b \in \Sigma^\infty$ with $\Pi(\sigma_{bad} \cdot \sigma'_b)$. We know that such a sequence exists since $\Pi \notin \text{Safety}(\Sigma)$. This is a consequence of Property 2.

According to the definition of the evaluation function for r-properties in a truth-domain (Definition 16), we have:

$$[\![\Pi]\!]_{\mathbb{B}_3}(\sigma_{good}) = [\![\Pi]\!]_{\mathbb{B}_3}(\sigma_{bad}) = ?$$

This is a contradiction with $\Pi \in MP^*(\mathbb{B}_3)$. □

A.2.4 Proof of Theorem 3: Multi-valued characterization of alternative monitorability

We prove each of these facts successively. Let $\Pi = (\phi, \varphi)$ be an r-property.

Proof of (i)

– Let $\Pi \in \text{Safety}(\Sigma)$, we show that $\Pi \in MP^*(\mathbb{B}_2^\perp)$. As $\Pi \in \text{Safety}(\Sigma)$, there exists a finitary property $\psi \subseteq \Sigma^*$, s.t. $\Pi = (A_f(\psi), A(\psi))$. Let us consider $\sigma_{good} \in \phi$ and $\sigma_{bad} \in \overline{\phi}$, we want to prove that the evaluations in $\mathbb{B}_2^\perp$ of these two sequences differ. On one hand, we have $\Pi(\sigma_{good})$ (since $\sigma_{good} \in \phi$) and thus $[\![\Pi]\!]_{\mathbb{B}_2^\perp}(\sigma_{good}) = ?$. On the other hand, we have $\neg\Pi(\sigma_{bad})$ and $\sigma_{bad} \notin A_f(\psi)$ (since $\sigma_{bad} \notin \phi$). Using Property 2, we have $\forall \mu \in \Sigma^\infty, \neg\Pi(\sigma_{bad} \cdot \mu)$, i.e., $[\![\Pi]\!]_{\mathbb{B}_2^\perp}(\sigma_{bad}) = \perp$.
– Let $\Pi \in MP^*(\mathbb{B}_2^\perp)$, we show that $\Pi \in \text{Safety}(\Sigma)$. According to the characterization of safety properties given in Property 1, showing that Π is a safety r-property amounts to show that it verifies $\Pi = (A_f(\mathit{Pref}(\phi)),$

$A(Pref(\varphi)))$. This is what we do by showing the inclusion in both ways.

- $\Pi \sqcap \Sigma^* \subseteq A_f(Pref(\phi))$ is immediate as for every sequence $\sigma \in \Pi \sqcap \Sigma^*$ (i.e., $\sigma \in \phi$), σ has all of its prefixes in $Pref(\phi)$. The same holds for $\Pi \sqcap \Sigma^\omega \subseteq A(Pref(\varphi))$.
- Let us show that $A_f(Pref(\phi)) \subseteq \Pi \sqcap \Sigma^*$. Let $\sigma \in A_f(Pref(\phi))$, we prove that $\sigma \in \Pi \sqcap \Sigma^*$. As $\sigma \in A_f(Pref(\phi))$, all prefixes of σ belong to $Pref(\phi)$, i.e., all prefixes of σ are the prefixes of a sequence in ϕ. Let σ_{min} be the smallest word in ϕ which is an continuation of σ. We distinguish two cases. If $\sigma_{min} = \sigma$, then $\sigma \in \Pi$. Else ($\sigma \prec \sigma_{min}$), as $\sigma_{min} \in \phi$, we have $[\![\Pi]\!]_{\mathbb{B}_2^\perp}(\sigma_{min}) =?$, and consequently, $[\![\Pi]\!]_{\mathbb{B}_2^\perp}(\sigma) =?$ (otherwise, we could not have $[\![\Pi]\!]_{\mathbb{B}_2^\perp}(\sigma_{min}) =?$). Using $\Pi \in MP^*(\mathbb{B}_2^\perp)$, we obtain $\sigma \in \phi$ and consequently $\sigma \in \Pi$.
  The same reasoning can be conducted to show that $A(Pref(\varphi))$
  $\subseteq \Pi \sqcap \Sigma^\omega$.

Finally, according to the definition of r-properties (Definition 1), we know that $\Pi = (\phi, \varphi)$ can be written $\Pi = (A_f(Pref(\phi)),$
$A(Pref(\varphi)))$, which gives us the expected result.

Proof of (ii)

- The reasoning used to prove that Guarantee$(\Sigma) \subseteq MP^*(\mathbb{B}_2^\top)$ is similar to the reasoning used to prove Safety$(\Sigma) \subseteq MP^*(\mathbb{B}_2^\perp)$. It suffices to show that all bad execution sequences are evaluated to "?". Furthermore, all good execution sequences are evaluated to $\top$. Indeed, once a sequence satisfies a guarantee r-property, all its continuations also satisfy it.
- Proving that $MP^*(\mathbb{B}_2^\top) \subseteq$ Guarantee(Σ) can be done, following the reasoning used to prove $MP^*(\mathbb{B}_2^\perp) \subseteq$ Safety(Σ), by showing that if $\Pi \in MP^*(\mathbb{B}_2^\top)$, then Π verifies $\Pi = \left(E_f(\overline{Pref(\overline{\phi})}), E(\overline{Pref(\overline{\varphi})})\right)$.

Proof of (iii)

- The proof of Safety$(\Sigma) \cup$ Guarantee$(\Sigma) \subseteq MP^*(\mathbb{B}_3)$ is straightforward by noticing that:
  - $MP^*(\mathbb{B}_2^\perp) \subset MP^*(\mathbb{B}_3)$,
  - and $MP^*(\mathbb{B}_2^\top) \subset MP^*(\mathbb{B}_3)$.
- The fact that $MP^*(\mathbb{B}_3) \subseteq$ Safety$(\Sigma) \cup$ Guarantee(Σ) is given by Lemma 3.

Proof of (iv) The proof is straightforward by noticing that every r-property can be evaluated by effectively distinguishing good and bad sequences. In other words, any reactivity r-property can be evaluated consistently with $\mathbb{B}_4$. Indeed, a good sequence σ_{good} is evaluated to $\top_c$ or $\top$ according to its continuations. A bad sequence σ_{bad} is evaluated to $\perp_c$ or $\perp$ according to its continuations. As we can see here, the truth values $\perp_c$ and $\top_c$ refine the verdict "?". □

A.2.5 Proof of Property 3: Correspondence between Streett automata states and $\mathbb{B}_4$

In this proof, $[\![\Pi]\!]$ stands for $[\![\Pi]\!]_{\mathbb{B}_4}$. Let us consider an execution sequence $\sigma \in \Sigma^*$ of length n.

Proof of $q_n \in Good^{\mathcal{A}_\Pi} \Leftrightarrow [\![\Pi]\!](\sigma) = \top$

- Let us suppose that $q_n \in Good^{\mathcal{A}_\Pi}$. Using the acceptance criterion for finite sequences, we have that σ is accepted by $\mathcal{A}_\Pi$. Furthermore, as $\mathcal{A}_\Pi$ specifies Π, we have $\Pi(\sigma)$. Now, let us consider $\mu \in \Sigma^+$ s.t. $|\sigma| + |\mu| = n' > n$ and $run(\sigma \cdot \mu, \mathcal{A}_\Pi) = q_0 \cdots q_{n'-1}$. As $q_n \in Good^{\mathcal{A}_\Pi}$, we deduce $\forall k \in \mathbb{N}, n \leq k \leq n' - 1 \Rightarrow q_k \in \bigcap_{i=1}^m R_i \cup P_i$ and consequently $\Pi(\sigma \cdot \mu)$. Let us consider $\mu \in \Sigma^\omega$, one may remark that $\forall i \in [1, m]$, $inf(\sigma \cdot \mu, \mathcal{A}_\Pi) \cap R_i \neq \emptyset \vee inf(\sigma \cdot \mu, \mathcal{A}_\Pi) \subseteq P_i$, which implies $\Pi(\sigma \cdot \mu)$. We have $\Pi(\sigma) \wedge \forall \mu \in \Sigma^\infty, \Pi(\sigma \cdot \mu)$, i.e., $[\![\Pi]\!](\sigma) = \top$.
- Conversely, let us suppose that $[\![\Pi]\!](\sigma) = \top$. By definition, it means $\forall \mu \in \Sigma^\infty, \Pi(\sigma \cdot \mu)$. According to the acceptance criterion of Streett automata, we deduce $\forall k \geq n, \forall \mu \in \Sigma^*, run(\sigma \cdot \mu, \mathcal{A}_\Pi) = q_0 \cdots q_n \cdots q_k \Rightarrow q_k \in \bigcap_{i=0}^m R_i \cup P_i$. That is to say $Reach_{\mathcal{A}_\Pi}(q_n) \subseteq \bigcap_{i=1}^m (R_i \cup P_i)$, i.e., $q_n \in Good^{\mathcal{A}_\Pi}$.

Proof of $q_n \in Good_c^{\mathcal{A}_\Pi} \Leftrightarrow [\![\Pi]\!](\sigma) = \top_c$. The proof is straightforward by examining the acceptance criterion for finite sequences.

- Let us suppose that $q_n \in Good_c^{\mathcal{A}_\Pi}$. Using the acceptance criterion for finite sequences, we have that σ is accepted by $\mathcal{A}_\Pi$. Moreover, as $\mathcal{A}_\Pi$ specifies Π, we deduce $\Pi(\sigma)$. Now, as $Reach_{\mathcal{A}}(q) \not\subseteq \bigcap_{i=1}^m (R_i \cup P_i)$, there exists a state q' of $\mathcal{A}_\Pi$ reachable from q and belonging to $\bigcup_{i=1}^m (\overline{R_i} \cap \overline{P_i})$. Consequently, there exists $\mu \in \Sigma^*$ s.t. $run(\sigma \cdot \mu) = q_0 \cdots q_n \cdots q'$. Still following the acceptance criterion we deduce $\neg\Pi(\sigma \cdot \mu)$, i.e., $[\![\Pi]\!](\sigma) = \top_c$.
- Conversely, the same reason can be used to prove the seeked result.

Proof of $q_n \in Bad_c^{\mathcal{A}_\Pi} \Leftrightarrow [\![\Pi]\!](\sigma) = \perp_c$. Similarly, the proof is straightforward by examining the acceptance criterion for finite sequences of Streett automata.

- Let us suppose that $q_n \in Bad_c^{\mathcal{A}_\Pi}$. Using the acceptance criterion of finite sequences, we have that σ is not

accepted by $\mathcal{A}_\Pi$. Furthermore, as $\mathcal{A}_\Pi$ specifies Π, we deduce $\neg\Pi(\sigma)$. Now, as $Reach_{\mathcal{A}}(q) \not\subseteq \bigcup_{i=1}^{m}(\overline{R_i} \cup \overline{P_i})$, there exists a state q' of $\mathcal{A}_\Pi$ reachable from q and belonging to $\bigcap_{i=1}^{m}(R_i \cup P_i)$. Consequently, there exists $\mu \in \Sigma^*$ s.t. $run(\sigma \cdot \mu) = q_0 \cdots q_n \cdots q'$. Still following the acceptance criterion, we deduce $\Pi(\sigma \cdot \mu)$, i.e., $[\![\Pi]\!](\sigma) = \bot_c$.
- Conversely, the same reason can be conducted.

Proof of $q_n \in Bad^{\mathcal{A}_\Pi} \Leftrightarrow [\![\Pi]\!](\sigma) = \bot$. The proof can be done following the same proof principle that the one used to prove $q_n \in Good^{\mathcal{A}_\Pi} \Leftrightarrow [\![\Pi]\!](\sigma) = \top$.

- Let us suppose that $q_n \in Bad^{\mathcal{A}_\Pi}$. Using the acceptance criterion on finite sequences, we have that σ is not accepted by $\mathcal{A}_\Pi$. Furthermore, as $\mathcal{A}_\Pi$ specifies Π, we deduce $\neg\Pi(\sigma)$. Now, let us consider $\mu \in \Sigma^+$ s.t. $|\sigma| + |\mu| = n' > n$ and $run(\sigma \cdot \mu, \mathcal{A}_\Pi) = q_0 \cdots q_{n'-1}$. As $q_n \in Bad^{\mathcal{A}_\Pi}$, we have $\forall k \in \mathbb{N}, n \leq k \leq n' - 1 \Rightarrow q_k \in \bigcup_{i=1}^{m} \overline{R_i} \cap \overline{P_i}$ and consequently $\neg\Pi(\sigma \cdot \mu)$. Let us consider $\mu \in \Sigma^\omega$, one may remark that $\forall i \in [1, m]$, $vinf(\sigma \cdot \mu, \mathcal{A}_\Pi) \cap R_i = \emptyset \wedge vinf(\sigma \cdot \mu, \mathcal{A}_\Pi) \not\subseteq P_i$, which implies $\neg\Pi(\sigma \cdot \mu)$. We have $\neg\Pi(\sigma) \wedge \forall \mu \in \Sigma^\infty, \neg\Pi(\sigma \cdot \mu)$, i.e., $[\![\Pi]\!](\sigma) = \bot$.
- Conversely, let us suppose that $[\![\Pi]\!](\sigma) = \bot$. By definition, it means $\forall \mu \in \Sigma^\infty, \neg\Pi(\sigma \cdot \mu)$. According to the acceptance criterion of Streett automata, we deduce $\forall k \geq n, \forall \mu \in \Sigma^*, run(\sigma \cdot \mu, \mathcal{A}_\Pi) = q_0 \cdots q_n \cdots q_k \Rightarrow q_k \in \bigcup_{i=0}^{m} \overline{R_i} \cap \overline{P_i}$. That is to say, $Reach_{\mathcal{A}_\Pi}(q_n) \subseteq \bigcup_{i=1}^{m}(\overline{R_i} \cap \overline{P_i})$, i.e., $q_n \in Bad^{\mathcal{A}_\Pi}$. □

A.3 Proofs for Sect. 6

A.3.1 Proof of Property 4: Equivalence between enforcement criteria

Before proving the equivalence between enforcement criteria, we state and prove an intermediate lemma.

Lemma 1 *Considering an m-automaton $\mathcal{A}_\Pi$ recognizing an r-property $\Pi = (\phi, \varphi)$ and $s \in \mathcal{S}(\mathcal{A}_\Pi)$ a strongly connected component of $\mathcal{A}_\Pi$. We have:*

$$I_s \neq \emptyset \Leftrightarrow \forall \sigma \in \Sigma^\omega, \quad vinf(\sigma, \mathcal{A}_\Pi) = s \Rightarrow \neg\varphi(\sigma)$$

The proof is in two steps by proving implications in both ways.

- Suppose $I_s \neq \emptyset$ and let us consider $\sigma \in \Sigma^\omega$ s.t. $vinf(\sigma, \mathcal{A}_\Pi) = s$. As $I_s \neq \emptyset$, then $\exists i \in I_s (\subseteq [1, m])$, $vinf(\sigma, \mathcal{A}_\Pi) \subseteq \overline{R_i} \wedge vinf(\sigma, \mathcal{A}_\Pi) \cap \overline{P_i} \neq \emptyset$. Then, using the acceptance criterion for infinite sequences, one can deduce that σ is not accepted by $\mathcal{A}_\Pi$.
- The other direction is straightforward using the acceptance criterion for infinite sequences (Definition 6). □

Now, let us prove Property 4. This proof relies on the computation of strongly connected components of a Streett automaton (SCC), both maximal and non maximal ones. The proof is in two steps by proving implications in both ways.

- (4) ⇒ (5). Let s be a SCC of $\mathcal{A}_\Pi$ s.t. $I_s \neq \emptyset$. Then using the previous lemma, every sequence s.t. $vinf(\sigma, \mathcal{A}_\Pi) = s$ is rejected by $\mathcal{A}_\Pi$. As Π is enforceable and satisfies (4), necessarily all prefixes of σ terminating in a state of s are not accepted. Otherwise, there would exist an accepting state (w.r.t. the acceptance criterion of finite sequences) in s. It would then be possible to build $\sigma' \in \Sigma^\omega$ with an infinite number of accepting prefixes, i.e., s.t. (4) is not satisfied.
- (5) ⇒ (4). Let us consider $\sigma \in \Sigma^\omega$ s.t. $\neg\varphi(\sigma)$. During its run on $\mathcal{A}_\Pi$, σ visits an SCC s infinitely often. As $\neg\varphi(\sigma)$, using the previous lemma, we have $I_s \neq \emptyset$. Using (5), $\exists i \in I_s, vinf(\sigma, \mathcal{A}_\Pi) \subseteq \overline{P_i}$ (and we already know $vinf(\sigma, \mathcal{A}_\Pi) \subseteq \overline{R_i}$). Let us consider $\sigma' \in \Sigma^*$ s.t. $\sigma' \prec \sigma$, the run of σ' on $\mathcal{A}_\Pi$ is s.t. $\exists q_1, \ldots, q_n \in Q^{\mathcal{A}_\Pi}, run(\sigma', \mathcal{A}_\Pi) = q_{\text{init}}^{\mathcal{A}_\Pi} \cdot q_1 \cdots q_n \cdot \{q \in vinf(\sigma, \mathcal{A}_\Pi)\}^*$. Consequently, we have $\forall \sigma' \prec \sigma, |\sigma'| \geq n \Rightarrow run(\sigma, \mathcal{A}_\Pi) = q_{\text{init}}^{\mathcal{A}_\Pi} \cdots q$, with $q \in \overline{R_i} \wedge q \in \overline{P_i}$. Using the acceptance criterion for finite sequences, we have that σ' is not accepted by $\mathcal{A}_\Pi$. □

A.3.2 Proof of Theorem 8: Enforceable m-reactivity properties are response properties

We show that any enforceable m-reactivity property Π is indeed a response property. The proof is based on the automata view, showing that the (m-reactivity) automaton $\mathcal{A}_\Pi$ associated with Π can be transformed into a response automaton $\mathcal{A}_\Pi{}'$ recognizing the same language.

Let us consider $\mathcal{A}_\Pi = (Q^{\mathcal{A}_\Pi}, q_{\text{init}}^{\mathcal{A}_\Pi}, \Sigma, \longrightarrow_{\mathcal{A}_\Pi}, \{(R_1, P_1), \ldots, (R_m, P_m)\})$ the Streett automaton associated with an enforceable property Π. We now consider the response automaton $\mathcal{A}_\Pi{}' = (Q^{\mathcal{A}_\Pi}, q_{\text{init}}^{\mathcal{A}_\Pi}, \Sigma, \longrightarrow_{\mathcal{A}_\Pi}, \{(R'_1, \emptyset), \ldots, (R'_m, \emptyset)\})$ with $R'_i = R_i \cup P_i$ for i in $[1, m]$.

We now show that a sequence σ of Σ^∞ is accepted by $\mathcal{A}_\Pi$ if and only if it is accepted by $\mathcal{A}_\Pi{}'$. We distinguish two cases according to whether σ is a finite or an infinite sequence.

- For a finite sequence σ, it is an accepting sequence of $\mathcal{A}_\Pi$ if and only if its run terminates either in a R_i-state or in a P_i-state for i in $[1, m]$. Consequently, its run also terminates in a R'_i-state of $\mathcal{A}_\Pi{}'$ for i in $[1, m]$, and therefore, it is an accepting sequence of $\mathcal{A}_\Pi{}'$.
- For an infinite sequence σ, note that the sets $vinf(\sigma, \mathcal{A}_\Pi)$ and $vinf(\sigma, \mathcal{A}_\Pi{}')$ coincide.

Let us assume first that σ is an accepting sequence of $\mathcal{A}_\Pi$, namely

$$\bigwedge_{i=1}^{m} \mathit{vinf}(\sigma, \mathcal{A}_\Pi) \cap R_i \neq \emptyset \vee \mathit{vinf}(\sigma, \mathcal{A}_\Pi) \subseteq P_i.$$

Since $R'_i \subseteq R_i$ and $R'_i \subseteq P_i$ we have

$$\bigwedge_{i=1}^{m} \mathit{vinf}(\sigma, \mathcal{A}_\Pi) \cap R'_i \neq \emptyset.$$

Consequently, σ is an accepting sequence of $\mathcal{A}_\Pi{}'$.
Now, let us assume that σ is a non-accepting sequence of $\mathcal{A}_\Pi$. Since Π is enforceable, then, according to Property 4, there exists $i \in [1, m]$ such that $\mathit{vinf}(\sigma, \mathcal{A}_\Pi) \subseteq \overline{R_i}$ and $\mathit{vinf}(\sigma, \mathcal{A}_\Pi) \subseteq \overline{P_i}$. Thus, we can deduce $\mathit{vinf}(\sigma, \mathcal{A}_\Pi) \subseteq (\overline{R_i} \cap \overline{P_i})$ and consequently $\mathit{vinf}(\sigma, \mathcal{A}_\Pi{}') \subseteq \overline{R'_i}$. Therefore, σ is a non-accepting sequence of $\mathcal{A}_\Pi{}'$. □

References

1. Alpern, B., Schneider, F.B.: Defining liveness. Inf. Process. Lett. **21**(4), 181–185 (1985)
2. Barringer, H., Rydeheard, D.E., Havelund, K.: Rule systems for run-time monitoring: from eagle to ruler. J. Log. Comput. **20**(3), 675–706 (2010)
3. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. Technical Report TUM-I0724, Institut für Informatik, Technische Universität München, December 2007
4. Bauer, A., Leucker, M., Schallhart, C.: Comparing ltl semantics for runtime verification. J. Log. Comput. **20**(3), 651–674 (2010)
5. Chang, E., Manna, Z., Pnueli, A.: The Safety-Progress Classification. Technical report, Department of Computer Science, Stanford University (1992)
6. Chang, E.Y., Manna, Z., Pnueli, A.: Characterization of temporal property classes. In: Automata, Languages and Programming, pp. 474–486 (1992)
7. Chen, F., Roşu, G.: MOP: an efficient and generic runtime verification framework. In: Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'07), pp. 569–588. ACM press, New York (2007)
8. Chen, F., Roşu, G.: Parametric trace slicing and monitoring. In: Kowalewski, S., Philippou, A. (eds) TACAS, Lecture Notes in Computer Science, vol. 5505, pp. 246–261. Springer, Berlin (2009)
9. Clarke, E.M., Wing, J.M.: Formal methods: state of the art and future directions. ACM Comput. Surv. **28**, 626–643 (1996)
10. Cousot, P., Cousot, R.: Abstract interpretation and application to logic programs. J. Log. Program. **13**(2–3), 103–179 (1992)
11. d'Amorim, M., Roşu, G.: Efficient monitoring of ω-languages. In: Proceedings of 17th International Conference on Computer-aided Verification (CAV'05). Lecture Notes in Computer Science, vol. 3576, pp. 364–378. Springer, Berlin (2005)
12. Emerson, E.A., Clarke, E.M.: Characterizing correctness properties of parallel programs using fixpoints. In: Proceedings of the 7th Colloquium on Automata, Languages and Programming, pp. 169–181. Springer, Berlin (1980)
13. Falcone, Y.: You should better enforce than verify. In: Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G.J., Roşu, G., Sokolsky, O., Tillmann, N. (eds.) RV. Lecture Notes in Computer Science, vol. 6418, pp. 89–105. Springer, Berlin (2010)
14. Falcone, Y., Fernandez, J.-C., Jéron, T., Marchand, H., Mounier, L.: More testable properties. In: Petrenko, A., da Silva Simão, A., Maldonado, J.C. (eds.) ICTSS. Lecture Notes in Computer Science, vol. 6435, pp. 30–46. Springer, Berlin (2010)
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, pp. 41–55. Springer, Berlin (2008)
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, pp. 593–600. ACM Press, New York (2009)
17. Falcone, Y., Fernandez, J.-C., Mounier, L.: Runtime verification of safety-progress properties. In: Bensalem, S., Peled, D. (eds.) RV. Lecture Notes in Computer Science, vol. 5779, pp. 40–59. Springer, Belin (2009)
18. Falcone, Y., Mounier, L., Fernandez, J.-C., Richier, J.-L.: Runtime enforcement monitors: composition, synthesis, and enforcement abilities. Formal Methods Syst. Des. **38**(3) (2011)
19. Fong, P.W.L.: Access control by tracking shallow execution history. In: Proceedings of the 2004 IEEE Symposium on Security and Privacy, pp. 43–55. IEEE Computer Society Press, Los Alamitos (2004)
20. Hamlen, K.W., Morrisett, G., Schneider, F.B.: Computability classes for enforcement mechanisms. ACM Trans. Program. Lang. Syst. **28**(1), 175–205 (2006)
21. 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, 10–13 Oct 2005, Revised Selected Papers and Discussions, pp. 374–383. Springer, Berlin (2008)
22. Havelund, K., Roşu, G.: Efficient monitoring of safety properties. Softw. Tools Technol. Transf. **6**(2), 158–173 (2002)
23. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, MA (1979)
24. Kupferman, O., Vardi, M.Y.: Model checking of safety properties. Formal Methods Syst. Des. **19**(3), 291–314 (2001)
25. Lamport, L.: Proving the correctness of multiprocess programs. IEEE Trans. Softw. Eng. **3**(2), 125–143 (1977)
26. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Log. Algebr. Program. **78**(5), 293–303 (2008)
27. Ligatti, J., Bauer, L., Walker, D.: Enforcing non-safety security policies with program monitors. In: ESORICS, pp. 355–373 (2005)
28. Ligatti, J., Bauer, L., Walker, D.: Run-time enforcement of nonsafety policies. ACM Trans. Inf. Syst. Secur. **12**(3), 1–41 (2009)
29. Manna, Z., Pnueli, A.: A hierarchy of temporal properties (invited paper, 1989). In: PODC '90: Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing, pp. 377–410. ACM Press, New York (1990)
30. Martinelli, F., Matteucci, I.: Through modeling to synthesis of security automata. Electron. Notes Theor. Comput. Sci. **179**, 31–46 (2007)
31. Matteucci, I.: Automated synthesis of enforcing mechanisms for security properties in a timed setting. Electron. Notes Theor. Comput. Sci. **186**, 101–120 (2007)
32. Pnueli, A., Zaks, A.: PSL model checking and run-time verification via testers. In: Misra J., Nipkow, T., Sekerinski, E. (eds.) FM. Lecture Notes in Computer Science, vol. 4085, pp. 573–586. Springer, Berlin (2006)
33. Roşu, G., Chen, F., Ball, T.: Synthesizing monitors for safety properties—this time with calls and returns. In: Workshop on Runtime

Verification (RV'08). Lecture Notes in Computer Science, vol. 5289, pp. 51–68. Springer, Berlin (2008)
34. Runtime Verification, 2001–2010. http://www.runtime-verification.org
35. Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. **3**(1), 30–50 (2000)
36. Streett, R.S.: Propositional dynamic logic of looping and converse. In: STOC '81: Proceedings of the 13th Annual ACM Symposium on Theory of Computing, pp. 375–383. ACM Press, New York (1981)
37. Viswanathan, M., Kim, M.: Foundations for the run-time monitoring of reactive systems—fundamentals of the MaC language. In: Liu, Z., Araki, K. (eds.) ICTAC. Lecture Notes in Computer Science, vol. 3407, pp. 543–556. Springer, Berlin (2004)