Form Methods Syst Des
DOI 10.1007/s10703-011-0114-4

Runtime enforcement monitors: composition, synthesis, and enforcement abilities

Yliès Falcone · Laurent Mounier · Jean-Claude Fernandez · Jean-Luc Richier

Abstract Runtime enforcement is a powerful technique to ensure that a program will respect a given set of properties. We extend previous work on this topic in several directions. Firstly, we propose a generic notion of enforcement monitors based on a memory device and finite sets of control states and enforcement operations. Moreover, we specify their enforcement abilities w.r.t. the general Safety-Progress classification of properties. Furthermore, we propose a *systematic* technique to produce a monitor from the automaton recognizing a given safety, guarantee, obligation or response property. Finally, we show that this notion of enforcement monitors is more amenable to implementation and encompasses previous runtime enforcement mechanisms.

Keywords Runtime enforcement · Monitor · Safety-progress classification · Monitor synthesis · Composition

1 Introduction

The growing complexity of nowadays programs and systems induces a rise of needs in validation. With the enhancement of engineering methods, software components tend to be more and more reusable. When retrieving an external component, the question of how this code meets a set of proper requirements arises. Using formal methods appears as a solution

Y. Falcone (✉) · L. Mounier · J.-C. Fernandez
Grenoble INP, CNRS VERIMAG, UJF-Grenoble 1, Grenoble 38041, France
e-mail: Ylies.Falcone@imag.fr

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

J.-C. Fernandez
e-mail: Jean-Claude.Fernandez@imag.fr

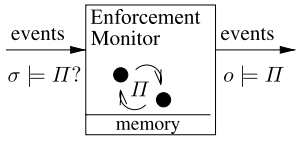
J.-L. Richier
Grenoble INP, CNRS LIG, UJF-Grenoble 1, Grenoble 38041, France
e-mail: Jean-Luc.Richier@imag.fr

Published online: 11 March 2011

to provide techniques to regain the needed confidence. However, these techniques should remain practical enough to be adopted by software engineers.

Runtime monitoring (see [1, 2] for brief overviews) falls in this category. It consists in supervising at runtime the execution of an underlying program against a set of expected properties: using a dedicated monitor allows to detect occurrences of specific property violations. Such a detection might provide a sufficient assurance. However, for some kind of systems a misbehavior might be not acceptable. To prevent this, a possible solution is to *enforce* the desired property: the monitor not only observes the current program execution, but it also controls it in order to ensure that the expected property is fulfilled.

Runtime enforcement monitoring was initiated by the work of Schneider [3] on *security automata*. In this work the monitors watch the current execution sequence and halt the underlying program whenever it deviates from the desired property. Such security automata are able to enforce the class of safety properties [4], stating that *nothing bad happens during program execution*. Later, Viswanathan [5] noticed that the class of enforceable properties is impacted by the computational power of the enforcement monitor. As enforcement mechanisms can implement no more than computable functions, the enforceable properties are included in the decidable ones. More recently, Ligatti et al. [6, 7] 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*, able to express some kinds of *obligations* used in security policies. To better cope with practical resource constraints, Fong [8] studied the effect of memory limitations on enforcement mechanisms (called *shallow-automata*). The various mechanisms and operated controls usually remain *transparent*, meaning that they always output the *longest correct prefix* of the original execution sequences. Therefore the initial sequence is minimally altered.



In this paper, we introduce a generic formalism for runtime enforcement under the transparency constraint. The proposed mechanism is schematically represented, in its most general form, by the figure on the left. This representation encompasses several real software implementations that can be assimilated to enforcement monitors, e.g., an access control mechanism where the input sequence is produced by a user and the output sequence is sent to a secured server.

A runtime enforcement monitor is a decision procedure dedicated to a property Π. It reads a finite (and possibly infinite) sequence of events σ and produces in output a new finite (and possibly infinite) sequence o. The monitor is equipped with an internal memory and a set of operations on the input events (possibly using the memory). Some constraints (e.g., transparency) may exist between σ and o that influence the operations performed by the monitor while reading σ. For instance, let us consider a transactional property Π to be enforced, stating that a given operation should be logged whenever it occurs. The transparency constraint leads the monitor to store some events of σ (and thus not producing them in output) as long as the transaction is not properly completed (the operation occurred, but it has not been logged yet). On the other hand, whenever the property Π is satisfied, the monitor simply dumps immediately each input event (together with the events previously stored in its memory). In some particular cases, by examining Π, the monitor may also determine that, at some point, whatever are the possible upcoming events, the input sequence will *never* (resp. will *always*) satisfy the property in the future. In such a situation this input sequence can be definitely blocked (resp. the monitor can be switched off, since it is not required anymore).

Our contributions In this paper, we propose to extend previous work in runtime enforcement monitoring in several directions. Firstly, we study the problem of enforcement relatively to the so-called *Safety-Progress* hierarchy of regular properties [9, 10]. This classification differs from the more classical safety-liveness classification [11, 12] by offering a rather clear characterization of a number of interesting kinds of properties (e.g., obligation, accessibility, justice, etc.). Thus, it provides a finer-grain classification of enforceable properties. Moreover, in this Safety-Progress hierarchy, each class of regular properties can be characterized by a particular kind of finite-state automaton $\mathscr{A}_{\Pi}$. Secondly, we introduce a generic notion of enforcement monitor based on a *finite set of control states* and an *auxiliary memory*. This general notion of enforcing monitor encompasses the previous notions of security automata, edit-automata and "shallow history" automata. Thirdly, we show how to generate an enforcement monitor for Π in a *systematic way*, from a recognizing automaton $\mathscr{A}_{\Pi}$.

A preliminary version of this paper appeared in [13]. This paper brings the following additional contributions. It first contains a more comprehensive theoretical basis as we revisit and extend results about the Safety-Progress classification of properties. Moreover, this paper introduces the notion of *e*-properties which are more suitable to represent and delineate the space of enforceable properties. We added more details in each section, and complete proofs of all mentioned theorems. Furthermore, we present the notion of enforcement monitor composition. At last we supply a comparison with related work and explain in details the advantages of the model of enforcement monitors proposed in this paper.

Paper organization The remainder of this article is organized as follows. Section 2 introduces some preliminary notations for our work. In Sect. 3 we recall briefly the necessary elements from the Safety-Progress classification of properties. We also add additional results to this classification. Then, we present our notion of enforcement monitor and their properties in Sect. 4. We address the problem of enforcement monitor composition in Sect. 5. Section 6 first exposes enforcement monitor synthesis and the proof of its correctness, and then studies enforcement capability of monitors w.r.t. Safety-Progress classes. Section 7 compares these results and the enforcement monitors with previous work. Finally, Sect. 8 gives some concluding remarks and directions for future work.

2 Preliminaries and notations

This section introduces some preliminary notations about the notions of *program execution sequences* and *program properties*.

2.1 Sequences, and execution sequences

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 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, by E^+ the set of non-empty finite sequences over E, and by E^ω the set of infinite sequences over E. The set $E^\infty = E^* \cup E^\omega$ is the set of all sequences (finite or not) over E. The empty sequence is denoted ε. The length (number of elements) of a finite sequence σ is denoted $|\sigma|$ and the $(i+1)$-th element of σ is denoted by σ_i. For two sequences $\sigma \in E^*, \sigma' \in E^\infty$, we denote by $\sigma \cdot \sigma'$ the concatenation of σ and σ'. When $\sigma \in E^*, \sigma' \in E^\infty \setminus \{\varepsilon\}$, we denote by $\sigma \prec \sigma'$

the fact that σ is a strict prefix of σ', that is, $\sigma \neq \varepsilon \Rightarrow |\sigma| < |\sigma'| \wedge \forall i \in [0, |\sigma| - 1], \sigma_i = \sigma'_i$. When $\sigma' \in E^*$, we note $\sigma \preceq \sigma' \stackrel{\text{def}}{=} \sigma \prec \sigma' \vee \sigma = \sigma'$. For $\sigma \in E^\infty \setminus \{\varepsilon\}$, we will need to designate its sub-sequences. In particular, for $n \in \mathbb{N}$, $\sigma_{\dots n}$ is the sub-sequence containing the $n+1$ first elements of σ. Also, when $|\sigma| > n$, the sub-sequence $\sigma_{n\dots}$ is the sequence containing all elements of σ but the n first ones. For $i, j \in \mathbb{N}$ with $i \leq j$, we denote by $\sigma_{i\dots j}$ the sub-sequence of σ containing the $(i+1)$-th to the $(j+1)$-th (included) elements.

Execution sequences A program $\mathscr{P}$ is considered as a generator of execution sequences. We are interested in a restricted set of actions or events the program can perform. These actions influence the truth value of properties the program is supposed to fulfill. Such execution sequences can be 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. We abstract these operations by a finite set of *events/actions*, namely an alphabet Σ. We denote by $\mathscr{P}_\Sigma$ a program for which the alphabet is Σ. The set of execution sequences of $\mathscr{P}_\Sigma$ is denoted $Exec(\mathscr{P}_\Sigma) \subseteq \Sigma^\infty$. This set is *prefix-closed*, that is $\forall \sigma \in Exec(\mathscr{P}_\Sigma), \forall \sigma' \in \Sigma^*, \sigma' \preceq \sigma \Rightarrow \sigma' \in Exec(\mathscr{P}_\Sigma)$.

2.2 Properties

Properties as sets of execution sequences In this paper we aim to enforce properties on programs. A property is generally defined as a set of execution sequences. More specifically a set $\phi \subseteq \Sigma^*$ of finite sequences of events (resp. $\varphi \subseteq \Sigma^\omega$ of infinite sequences of events) is called a *finitary property* (resp. an *infinitary property*). We denote by $\overline{\phi}$ (resp. $\overline{\varphi}$) the negation of ϕ (resp. φ), that is the complement of ϕ (resp. φ) in Σ^* (resp. Σ^ω), formally defined as $\Sigma^* \setminus \phi$ (resp. $\Sigma^\omega \setminus \varphi$). Considering a given finite (resp. infinite) execution sequence σ and a property ϕ (resp. φ), when $\sigma \in \phi$, denoted $\phi(\sigma)$ (resp. $\sigma \in \varphi$, denoted $\varphi(\sigma)$), we say that σ *satisfies* ϕ (resp. φ). 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). Moreover, for a finitary property ϕ and an execution sequence $\sigma \in \Sigma^\infty$, we denote by $\text{Pref}_\prec(\phi, \sigma)$ the set of all (strict) prefixes of σ satisfying ϕ, i.e., $\text{Pref}_\prec(\phi, \sigma) = \{\sigma' \in \phi \mid \sigma' \prec \sigma\}$. This set is a chain (i.e., a totally ordered set) regarding the order relation $\prec$. The (unique) maximal element of the set $\text{Pref}_\prec(\phi, \sigma)$, namely the longest prefix of σ satisfying ϕ (noted $\text{Max}(\text{Pref}_\prec(\phi, \sigma))$) is the maximal element regarding $\prec$ if $\text{Pref}_\prec(\phi, \sigma) \neq \emptyset$. Given a property $\phi \subseteq \Sigma^*$ and an execution sequence $\sigma \in \Sigma^*$, a straightforward property of the set $Pref_\prec(\phi, \sigma)$ is that $\forall a \in \Sigma, \neg\phi(\sigma) \Rightarrow Max(Pref_\prec(\phi, \sigma \cdot a)) = Max(Pref_\prec(\phi, \sigma))$.

Enforcement properties In this paper we are interested in enforceable properties. As stated in the introduction, enforcement monitors should output the longest "correct" prefix of an execution sequence which does not satisfy the expected property. To do so, an enforcement monitor decides property satisfaction using always a finite observation. Furthermore, as we consider finite and infinite execution sequences (that a program may produce), enforceable properties should characterize satisfaction for both kinds of sequences in a uniform way. We advocate that the separation of finitary and the infinitary parts of a property clarifies the understanding of monitoring. An enforcement monitor (or a monitor) can be seen as a

[1] This is the distinction, made by Schneider [3], 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).

decision procedure reading a finite prefix and examining the satisfaction of this prefix w.r.t. a given correctness property.

Therefore, we introduce *e*-properties (enforcement properties) as follows. An *e*-property is defined[2] as a pair $(\phi, \varphi) \subseteq \Sigma^* \times \Sigma^\omega$. Intuitively, the finitary property ϕ represents the desirable property that finite execution sequences should fulfill, whereas the infinitary property φ is the expected property for infinite execution sequences. The definition of negation of an *e*-property follows from definition of negation for finitary and infinitary properties. For an *e*-property (ϕ, φ), we define $\overline{(\phi, \varphi)}$ as $(\overline{\phi}, \overline{\varphi})$. Boolean combinations of *e*-properties are defined in a natural way. For $* \in \{\cup, \cap\}$, $(\phi_1, \varphi_1) * (\phi_2, \varphi_2) = (\phi_1 * \phi_2, \varphi_1 * \varphi_2)$. Considering an execution sequence $\sigma \in Exec(\mathscr{P}_\Sigma)$, we say that σ satisfies (ϕ, φ) when $\sigma \in \Sigma^* \wedge \phi(\sigma) \vee \sigma \in \Sigma^\omega \wedge \varphi(\sigma)$. For an *e*-property $\Pi = (\phi, \varphi)$, we note $\Pi(\sigma)$ when σ satisfies (ϕ, φ).

3 A Safety-Progress classification of *e*-properties

This section recalls and extends some results about the Safety-Progress [9, 10] classification of properties. In the original papers this classification introduced a hierarchy between *regular* properties[3] defined as sets of *infinite* execution sequences. We extend the classification to deal with finite-length execution sequences. As so we revisit this classification for regular *e*-properties.

3.1 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.
- *Persistence* properties are the properties for which whenever a sequence satisfies a property, *all but finitely many* of its prefixes satisfy this 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 combination of safety and guarantee properties.
- The *reactivity class* can be defined as the class obtained by Boolean combination of response and persistence properties. This is the most general class containing all linear temporal properties [9]. In this paper, we will focus on sub-classes of reactivity to characterize the set of enforceable properties.

[2] We advocate that a pair of sets makes the distinction between the finitary and the infinitary part of the property more explicit. Though other notations could be considered as well.

[3] In the rest of the paper, the term property will stand for regular property.

The requirements provided in the following example introduces the aforementioned classes of properties. In Example 2, we formalize those requirements as *e*-properties.

Example 1 (Informal requirements) Let us consider an operating system with a secured operation op_s (needing an authentication) and an unsecured operation op_u. The system is endowed with three primitives related to authentication: *r_auth* (requesting authentication) emitted by users, and *g_auth* (granting authentication), *d_auth* (denying authentication) emitted by an internal authentication mechanism. Then,

- the requirement Π_1 stating that "If op_s ever occurs, then it should be immediately preceded by a granted authentication *g_auth*." can be formalized as a *safety e*-property;
- the requirement Π_2 stating that "Each work session of a user should contain a complete authentication step terminated either by a grant (*g_auth*) or a deny (*d_auth*) operation. In case of a successful authentication, the work session may contain secured and unsecured operations. Otherwise, it should contain only unsecured operation and it should be terminated by a user disconnection (*disco*)." can be formalized as a *guarantee e*-property;
- the requirement Π_3 stating that "The system should run forever; or, if a *d_auth* is issued, then the user should be disconnected and then the system should terminate (*end*)." can be formalized as an *obligation e*-property;
- the requirement Π_4 stating that "Each occurrence of *r_ auth* should be first written in a log file and then answered either with a *g_auth* or a *d_auth* without any occurrence of op_s or *r_auth* in the meantime." can be formalized as a *response e*-property;
- the property Π_5 stating that "After a *d_auth*, a (forbidden) use of operation op_s should imply that at some point any future call to *r_auth* will always result in a *d_ auth* answer." can be formalized as a *persistence e*-property.

The Safety-Progress classification is an alternative to the classical Safety-Liveness [11, 12] dichotomy. Unlike this one, 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* [9]: 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 view links the classes to their expression in temporal logic. At last, the automata-view gives syntactic characterization on the automata recognizing the properties of a given class. In this paper, we consider only the automata view dedicated to *e*-properties.

3.2 The automata view of *e*-properties

For the automata view of the Safety-Progress classification, we follow [9, 14] and define *e*-properties using Streett automata. For each class of the Safety-Progress classification it is possible to syntactically characterize a recognizing finite-state automaton. We define[4] a variant of deterministic and complete Streett automata (introduced in [15] and used in [14]) for property recognition. 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 *e*-properties.

[4]There exist several equivalent definitions of Streett automata dedicated to infinite sequences recognition. We choose here to follow the definition used in [9] and also only consider finite-state automata in the remainder.

Definition 1 (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 $\longrightarrow : Q \times \Sigma \to Q$ 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. In the following $\mathscr{A} = (Q^{\mathscr{A}}, q_{\text{init}}^{\mathscr{A}}, \Sigma, \longrightarrow_{\mathscr{A}}, \{(R_1, P_1), \ldots, (R_m, P_m)\})$ designates a Streett m-automaton.

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

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

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

Definition 2 (Acceptance condition (infinite sequences)) For $\sigma \in \Sigma^{\omega}$, we say that $\mathscr{A}$ accepts σ if $\forall i \in [1, m], vinf(\sigma, \mathscr{A}) \cap R_i \neq \emptyset \vee vinf(\sigma, \mathscr{A}) \subseteq P_i$.

To deal with *e*-properties we need to define also an acceptance criterion for *finite* sequences.

Definition 3 (Acceptance condition (finite sequences)) For a finite-length execution sequence $\sigma \in \Sigma^*$ such that $|\sigma| = n$, we say that the m-automaton $\mathscr{A}$ accepts σ if $(\exists q_0, \ldots, q_n \in Q^{\mathscr{A}}, run(\sigma, \mathscr{A}) = q_0 \cdots q_n \wedge q_0 = q_{\text{init}}^{\mathscr{A}} \text{ and } \forall i \in [1, m], q_n \in P_i \cup R_i)$.

The hierarchy of automata The Safety-Progress hierarchy as defined in [14] can be seen in the automata view by setting syntactic restrictions on a Streett automaton.

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

Figure 1 schematizes each basic class. The sets of persistent and recurrent states are represented by squares. Allowed transitions between the different kinds of states are represented by arrows.

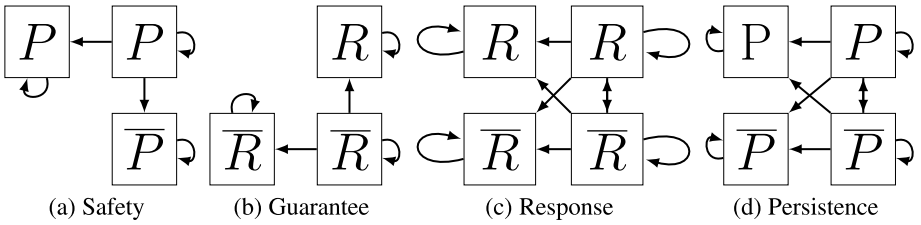
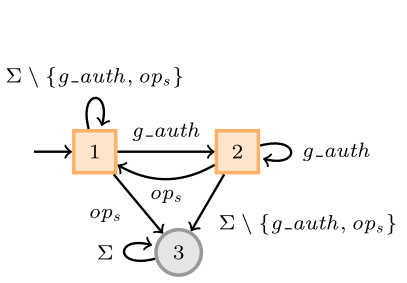
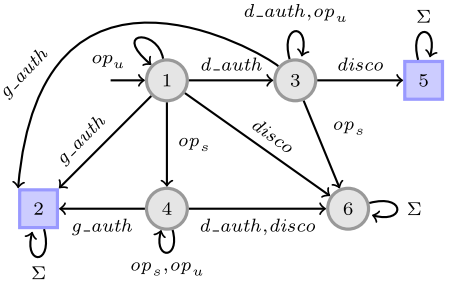


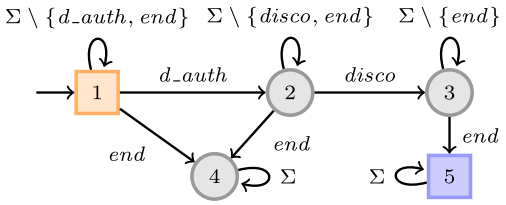
Fig. 1 Shapes of Streett automata for the basic classes



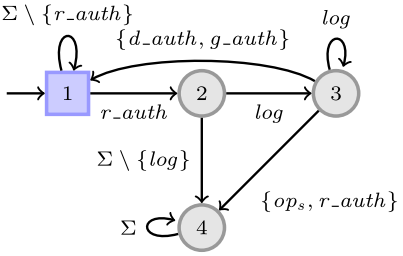
(a) Streett safety automaton for Π_1



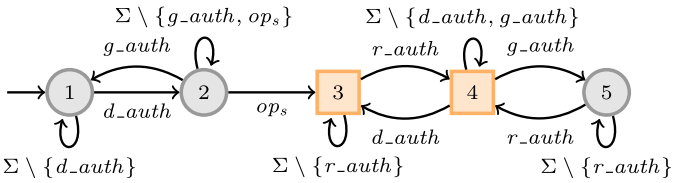
(b) Streett guarantee automaton for Π_2



(c) Streett obligation automaton for Π_3



(d) Streett response automaton for Π_4



(e) Streett persistence for Π_5

Fig. 2 Streett Automata for the *e*-properties formalizing the requirements of Example 1

Automata and e-properties We say that a Streett automaton $\mathscr{A}_\Pi$ *defines* an *e*-property $(\phi, \varphi) \in \Sigma^* \times \Sigma^\omega$ if the set of finite (resp. infinite) execution sequences accepted by $\mathscr{A}_\Pi$ is equal to ϕ (resp. φ). Conversely, an *e*-property $(\phi, \varphi) \in \Sigma^* \times \Sigma^\omega$ is said to be *specified* by an automaton $\mathscr{A}_\Pi$ if the set of finite (resp. infinite) execution sequences accepted by the automaton $\mathscr{A}_\Pi$ is ϕ (resp. φ).

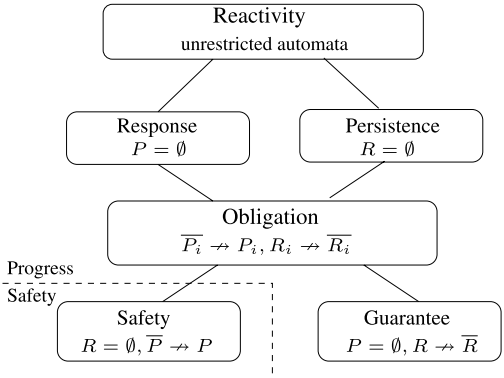


Fig. 3 The Safety-Progress classification of e-properties

Example 2 (Specifying e-properties by Streett automata) The requirements introduced in Example 1 can be formalized as e-properties specified by the Streett automata represented in Fig. 2. The requirement R_i is formalized by the e-property Π_i specified by the automaton $\mathscr{A}_{\Pi_i}$, $i \in \{1, 2, 3, 4, 5\}$, with initial state 1.

- For $\mathscr{A}_{\Pi_1}$ (Fig. 2a), the set of states is $\{1, 2, 3\}$, $R = \emptyset$, and $P = \{1, 2\}$.
- For $\mathscr{A}_{\Pi_2}$ (Fig. 2b), the set of states is $\{1, 2, 3, 4, 5, 6\}$, $P = \emptyset$, and $R = \{2, 5\}$.
- For $\mathscr{A}_{\Pi_3}$ (Fig. 2c), the set of states is $\{1, 2, 3, 4, 5\}$, $P = \{1\}$, and $R = \{5\}$.
- For $\mathscr{A}_{\Pi_4}$ (Fig. 2d), the set of states is $\{1, 2, 3, 4\}$, $P = \emptyset$, and $R = \{1\}$.
- For $\mathscr{A}_{\Pi_5}$ (Fig. 2e), the set of states is $\{1, 2, 3, 4, 5\}$, $P = \{3, 4\}$, and $R = \emptyset$.

It is possible to relate the syntactic characterization on the automata to the semantic characterization of the properties they specify. This is stated by the following definition (transposed from the initial definition in [14]).

Definition 4 (e-properties classes) An e-property (ϕ, φ) is a regular κ-e-property if it is specifiable by a finite state κ-automaton, where $\kappa \in \{$safety, guarantee, obligation, response, persistence, reactivity$\}$. Moreover, when an obligation e-property is specified by an m-obligation automaton, this e-property is said to be an m-obligation e-property.

Given an alphabet Σ, we note Safety(Σ) (resp. Guarantee(Σ), Obligation(Σ), Response(Σ), Persistence(Σ)) the set of safety (resp. guarantee, obligation, response, persistence) e-properties defined over Σ. Following [14], it can be shown that the Safety-Progress classification of e-properties is a hierarchy, presented in Fig. 3.

3.3 Some useful facts about e-properties

We present some straightforward consequences of the definitions of safety and guarantee e-properties.

Property 1 (Closure of e-properties) Considering an e-property Π specified by a Streett automaton $\mathscr{A}_\Pi$ defined over an alphabet Σ, the following facts hold:

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

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

Proof The proof is given in Appendix A.1. It uses the acceptance conditions and syntactic restrictions of Streett automata for safety and guarantee e-properties. □

Properties of automata Given a Streett m-obligation automaton (with m accepting pairs), it is possible first to express it as a conjunction of 1-obligation properties and second to characterize the languages accepted by "forgetting" some accepting pairs of the initial automaton. This is formalized as follows.

Lemma 1 (About obligation e-properties) *Given an m-automaton $\mathscr{A}_\Pi = (Q, q_{\text{init}}, \Sigma, \longrightarrow, \{(R_1, P_1), \ldots, (R_m, P_m)\})$ recognizing an m-obligation e-property Π. Π can be expressed as $\bigcap_{i=1}^{m} \Pi_i$, where Π_i is a 1-obligation e-property of the form $\Pi_i = \text{Safety}_i \cup \text{Guarantee}_i$ where Safety_i and Guarantee_i are respectively safety and guarantee e-properties. Moreover, given a subset $X \subseteq [1, m]$, the automaton $\mathscr{A}_{\Pi/X} = (Q, q_{\text{init}}, \Sigma, \longrightarrow, \{(R_i, P_i) \mid i \in X\})$ recognizes the property $\bigcap_{i \in X} \Pi_i$.*

Proof For infinite execution sequences, this proof has been done in [14]. For finite execution sequences, the proof is a straightforward adaptation. □

4 Property enforcement via enforcement monitors

Considering a program $\mathscr{P}_\Sigma$, we aim at constructing an enforcement monitor for an e-property (ϕ, φ) over Σ.

4.1 Enforcement monitors

We now define the central notion of enforcement monitor. Such a runtime device monitors a target program by watching its relevant events. It is an automaton-based mechanism endowed with an internal memory. On each input event its state evolves and an enforcement operation is performed. Enforcement operations operate a modification of the internal memory of the enforcement monitor and potentially produce an output. Enforcement monitors are parameterized by a set of enforcement operations *Ops*.

Definition 5 (*Ops*—Enforcement operations) Enforcement operations take as inputs an event and a memory content (i.e., a sequence of events) to produce an output sequence and a new memory content: $Ops \subseteq 2^{(\Sigma \times \Sigma^*) \to (\Sigma^* \times \Sigma^*)}$.

Definition 6 (Generic enforcement monitor (EM(Ops))) An *enforcement monitor* $\mathscr{A}_\downarrow$ is a 4-tuple $(Q^{\mathscr{A}_\downarrow}, q_{\text{init}}^{\mathscr{A}_\downarrow}, \longrightarrow_{\mathscr{A}_\downarrow}, Ops)$ defined relatively to a set of events Σ and parameterized by a set of enforcement operations *Ops*. The finite set $Q^{\mathscr{A}_\downarrow}$ denotes the control states, $q_{\text{init}}^{\mathscr{A}_\downarrow} \in Q^{\mathscr{A}_\downarrow}$ is the initial state. The complete function $\longrightarrow_{\mathscr{A}_\downarrow}: Q^{\mathscr{A}_\downarrow} \times \Sigma \to Q^{\mathscr{A}_\downarrow} \times Ops$ is the transition function. In the following we abbreviate $\longrightarrow_{\mathscr{A}_\downarrow} (q, a) = (q', \alpha)$ by $q \xrightarrow{a/\alpha}_{\mathscr{A}_\downarrow} q'$.

In the remainder of this section, $\sigma \in \Sigma^\infty$ designates an execution sequence, and $\mathscr{A}_\downarrow = (Q^{\mathscr{A}_\downarrow}, q_{\text{init}}^{\mathscr{A}_\downarrow}, \longrightarrow_{\mathscr{A}_\downarrow}, Ops)$ designates an EM(Ops).

Definition 7 (Run and trace) The *run* of σ on $\mathscr{A}_{\downarrow}$ is the sequence of states involved by the execution of $\mathscr{A}_{\downarrow}$ when σ is input. It is formally defined as $run(\sigma, \mathscr{A}_{\downarrow}) = q_0 \cdot q_1 \cdots$ where $q_0 = q_{\text{init}}^{\mathscr{A}_{\downarrow}} \wedge \forall i \ (q_i \in Q^{\mathscr{A}_{\downarrow}} \wedge q_i \xrightarrow{\sigma_i/\alpha_i}_{\mathscr{A}_{\downarrow}} q_{i+1})$. The *trace* resulting in the execution of σ on $\mathscr{A}_{\downarrow}$ is the sequence (finite or not) of tuples $(q_0, \sigma_0/\alpha_0, q_1) \cdot (q_1, \sigma_1/\alpha_1, q_2) \cdots (q_i, \sigma_i/\alpha_i, q_{i+1}) \cdots$ where $run(\sigma, \mathscr{A}_{\downarrow}) = q_0 \cdot q_1 \cdots$ and $\forall i, \alpha_i \in Ops$.

We formalize the way an EM(Ops) reacts to an input sequence provided by a target program through the standard notions of *configuration* and *derivation*.

Definition 8 (Configurations and derivations of an EM(Ops)) A *configuration* is a triplet $(q, \sigma, m) \in Q^{\mathscr{A}_{\downarrow}} \times \Sigma^* \times \Sigma^*$ where q denotes the current control state, σ the current input sequence, and m the current memory content.

We say that a configuration (q', σ', m') is *derivable in one step* from the configuration (q, σ, m) and *produces the output* $o \in \Sigma^*$, and we note $(q, \sigma, m) \xhookrightarrow{o} (q', \sigma', m')$ if and only if $\sigma = a \cdot \sigma' \wedge q \xrightarrow{a/\alpha}_{\mathscr{A}_{\downarrow}} q' \wedge \alpha(a, m) = (o, m')$.

We say that a configuration C' is *derivable in several steps* from a configuration C and *produces the output* $o \in \Sigma^*$, and we note $C \overset{o}{\Longrightarrow}_{\mathscr{A}_{\downarrow}} C'$, if and only if there exists $k \geq 0$ and configurations $C_0, C_1, \ldots, C_k$ such that $C = C_0$, $C' = C_k$, $C_i \xhookrightarrow{o_i} C_{i+1}$ for all $0 \leq i < k$, and $o = o_0 \cdot o_1 \cdots o_{k-1}$.

The notion of enforcement is based on how a monitor transforms a given input sequence in an output sequence. For the upcoming definitions we will distinguish between finite and infinite sequences.

Definition 9 (Sequence transformation) We define the transformation performed by an EM(Ops) while reading an input sequence $\sigma \in \Sigma^\infty$ (produced by a program $\mathscr{P}_\Sigma$) and producing an output sequence $o \in \Sigma^\infty$. The total function $\Downarrow_{\mathscr{A}_{\downarrow}} \subseteq \Sigma^\infty \times \Sigma^\infty$ is defined as follows:

- The empty sequence ε is transformed into itself by $\mathscr{A}_{\downarrow}$, i.e., $\varepsilon \Downarrow_{\mathscr{A}_{\downarrow}} \varepsilon$. This is the case when the underlying program does not produce any event.
- The sequence $\sigma \in \Sigma^+$ is transformed by $\mathscr{A}_{\downarrow}$ into the sequence $o \in \Sigma^*$, which is noted $\sigma \Downarrow_{\mathscr{A}_{\downarrow}} o$, if $\exists q' \in Q^{\mathscr{A}_{\downarrow}}, \exists m \in \Sigma^*, (q_{\text{init}}^{\mathscr{A}_{\downarrow}}, \sigma, \varepsilon) \overset{o}{\Longrightarrow}_{\mathscr{A}_{\downarrow}} (q', \varepsilon, m)$. That is, if there exists a derivation starting from the initial state and producing o.
- The sequence $\sigma \in \Sigma^\omega$ is transformed by $\mathscr{A}_{\downarrow}$ into the sequence $o \in \Sigma^*$, which is noted $\sigma \Downarrow_{\mathscr{A}_{\downarrow}} o$, if $\exists \sigma' \prec \sigma, \sigma' \Downarrow_{\mathscr{A}_{\downarrow}} o \wedge \forall \sigma'' \in \Sigma^*, \sigma' \prec \sigma'' \Rightarrow \sigma'' \Downarrow_{\mathscr{A}_{\downarrow}} o$. That is, the finite sequence o is produced if there exists a prefix of σ which produces o, and each continuation of this prefix produces o as well.
- The sequence $\sigma \in \Sigma^\omega$ is transformed by $\mathscr{A}_{\downarrow}$ into the sequence $o \in \Sigma^\omega$, which is noted $\sigma \Downarrow_{\mathscr{A}_{\downarrow}} o$, if

$$\forall o' \in \Sigma^*, o' \prec o \Rightarrow \exists \sigma'', o'' \in \Sigma^*, \sigma'' \prec \sigma \wedge o' \prec o'' \wedge \sigma'' \Downarrow_{\mathscr{A}_{\downarrow}} o''$$
$$\wedge \quad \forall \sigma', o' \in \Sigma^*, \sigma' \prec \sigma \wedge \sigma' \Downarrow_{\mathscr{A}_{\downarrow}} o' \Rightarrow o' \prec o.$$

That is, each prefix of o can be produced from a prefix of σ.

4.2 Enforcing a property

Roughly speaking, the purpose of an EM(Ops) is to read some unsafe input sequence produced by a program and to transform it into an output sequence that satisfies a given e-property Π. Before defining this notion more formally, we first explain what we mean exactly by *property enforcement*, and what are the consequences of this definition on the set of e-properties we shall consider.

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

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

A consequence of this definition of transparency is that an e-property (ϕ, φ) will be considered as *enforceable* only if *each incorrect sequence has a* longest *correct prefix*, or, equivalently, if *any infinite incorrect sequence has only a* finite number *of correct prefixes*. We use this criterion as a definition for *enforceable properties*. More formally:

Definition 10 (Enforceable e-property) An e-property (ϕ, φ) is *enforceable* iff:

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

The set of enforceable e-properties is denoted EP. Note that an EM(Ops) will output the empty sequence ε in two distinct cases: either when ε is the longest correct prefix of the input sequence, or when this input sequence has no correct prefix at all.[5]

Finally, since we have to deal with potentially infinite input sequences, the output sequence should be produced in an incremental way:[6] for each current prefix σ of the input sequence read by the EM(Ops), the *current* produced output o should be sound and transparent w.r.t. Π and σ. Furthermore, deciding whether a finite sequence σ satisfies Π or not should be computable in a finite amount of time (and by reading only a finite continuation of σ). It is indeed the case in our framework since we are dealing with regular properties.

This condition rules out particular properties saying for instance that "sequences containing an event e are accepted only if they are finite".

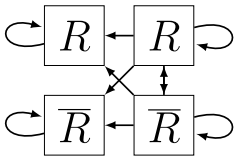
Enforceable properties w.r.t. the Safety-Progress classification In [16], we have given a characterization of the set of enforceable properties:

Theorem 1 (Enforceable properties [16]) *The set of response e-properties is the set of enforceable properties w.r.t. the Safety-Progress classification.*

Proof The formal proof can be found in [16]; we give here a sketch of this proof for the sake of completeness.

[5]This latter case is avoided in [6] by assuming that properties under consideration always contain ε.

[6]This limitation can be seen from a runtime verification point of view: verifying infinitary properties at runtime, on an execution sequence produced on-the-fly, should be done by checking finite prefixes of the current execution sequence.



First, we show that response e-properties are enforceable. Consider a response e-property (ϕ, φ) recognized by a response automaton, with the shape depicted on the left. Consider an infinite execution sequence $\sigma \in \Sigma^{\omega}$, and suppose that $\neg\varphi(\sigma)$. This means, according to the acceptance criterion for infinite sequences (Definition 2), that the R-states are not visited infinitely often. In other words, σ has finitely many prefixes for which the run ends in a R-state. According to the acceptance criterion for finite sequences (Definition 3), finitely many prefixes of σ belong to ϕ. Second, in order to explain that response properties are *exactly* the set of enforceable properties, [16] proceeds as follows. In the Safety-Progress *hierarchy*, it shows that the subset of enforceable persistence e-properties is actually included in the set of response e-properties. Indeed, it is possible to show that automata specifying enforceable properties can be encoded as response automata. The reader is referred to the examples in Sect. 6.2.2 presenting (non enforceable) persistence e-properties. □

As a straightforward consequence, safety, guarantee, and obligation e-properties are enforceable. While Theorem 1 provides a useful characterization of enforceable properties, there remain some fundamental questions: "How enforcement monitors should effectively enforce properties?" and "How is it possible to obtain such enforcement mechanisms from the definition of properties?". These questions are respectively addressed in the remainder of this section and in Sect. 6.

Property-enforcement We define the notion of property-enforcement by an EM(Ops). This notion of enforcement relates the input sequence produced by the program and fed to the EM(Ops) and the output sequence allowed by the EM(Ops) (correct w.r.t. the property under consideration). In practice, it might be difficult for an EM(Ops) to produce the same sequence since an EM(Ops) has to perform some additional statements to enforce the property or some non-observable actions or events may occur.

As a consequence, in the general case, the comparison between input and output sequences is performed up to some equivalence relation $\approx \subseteq \Sigma^{\infty} \times \Sigma^{\infty}$ (for which some events may be not considered). Note that the considered equivalence relation should preserve the e-property under consideration.

Definition 11 (Property-Enforcement$_{\approx}$) Let us consider an enforceable e-property $\Pi = (\phi, \varphi) \in EP$, we say that $\mathscr{A}_{\downarrow}$ enforces the property (ϕ, φ), relatively to an equivalence relation $\approx$, on a program $\mathscr{P}_{\Sigma}$ (noted $Enf_{\approx}(\mathscr{A}_{\downarrow}, (\phi, \varphi), \mathscr{P}_{\Sigma})$) iff for all $\sigma \in Exec(\mathscr{P}_{\Sigma})$, there exists $o \in \Sigma^{\infty}$, such that the following constraints hold:

$$\sigma \Downarrow_{\mathscr{A}_{\downarrow}} o \tag{1}$$

$$\Pi(\sigma) \Rightarrow \sigma \approx o \tag{2}$$

$$\neg\Pi(\sigma) \wedge Pref_{\prec}(\phi, \sigma) = \emptyset \Rightarrow o \approx \varepsilon \tag{3}$$

$$\neg\Pi(\sigma) \wedge Pref_{\prec}(\phi, \sigma) \neq \emptyset \Rightarrow o \approx Max(Pref_{\prec}(\phi, \sigma)) \tag{4}$$

(1), (2), (3), and (4) ensure soundness and transparency of $\mathscr{A}_{\downarrow}$: (1) stipulates that the sequence σ is transformed by $\mathscr{A}_{\downarrow}$ into a sequence o; (2) ensures that if σ already satisfied the property then it is not transformed. When there is no correct prefix of σ satisfying the property, (3) ensures that the EM(Ops) outputs nothing (the empty sequence ε). If there exists a prefix of σ satisfying the property (4) ensures that o is the longest prefix of σ satisfying the property.

Soundness is due to the fact that the produced sequence o, when different from ε, always satisfies the property Π. Transparency is ensured by the fact that, up to the equivalence relation $\approx$, correct execution sequences are not changed, and incorrect ones are restricted to their longest correct prefix.

One may remark that we could have set $Max(Pref_{\prec}(\phi, \sigma))$ to ε 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(Ops) produces ε. 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(Ops) for a system, this sequence can be "tagged" as incorrect.

4.3 Instantiating generic enforcement monitors

In the remainder of this article we will focus our study on some particular, but expressive enough (regarding enforcement), enforcement monitors. This kind of monitor will comply with the transparency constraint stated in Definition 10.

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),[7] or
- to *dump* the content of the memory device (when the input program comes back to a correct behavior), or
- to switch *off* permanently the monitor (when the property is satisfied for ever).

We give a more precise definition of such enforcement operations.

Definition 12 (Enforcement operations $\{halt, store, dump, off\}$) In the following we consider a set $Ops \stackrel{\text{def}}{=} \{halt, store, dump, off\}$, where the enforcement operations are defined as follows: $\forall a \in \Sigma \cup \{\varepsilon\}, \forall m \in \Sigma^*$,

$$\begin{array}{rr} halt(a, m) = (\varepsilon, m) & store(a, m) = (\varepsilon, m.a) \\ dump(a, m) = (m.a, \varepsilon) & off(a, m) = (m.a, \varepsilon) \end{array}$$

(a designates the input event of the monitor and m the memory device: its content).

Note that the *off* and *dump* operations have the same definitions. From a theoretical perspective, the *off* operation 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 useful to know when the monitor is not needed anymore.

We also distinguish two subsets of the set of states of an enforcement monitor instantiated with the set of enforcement operations $\{halt, store, dump, off\}$: the states in *Halt* (resp. *Off*) are used to represent the states in which the program (resp. monitor) should be stopped. Intuitively, states in *Halt* (resp. *Off*) are those entered by a transition labeled by a *halt* (resp. *off*) operation. Furthermore, we assume that, after performing a *halt* (resp. *off* operation), an EM cannot perform another operation than *halt* (resp. *off*).

[7] Note that postponing an event can be done only when there is no causal dependency with subsequent events in the system.

Definition 13 (Instantiated enforcement monitor) An EM is an instantiated EM(Ops) $(Q^{\mathscr{A}_\downarrow}, q_{\text{init}}^{\mathscr{A}_\downarrow}, \longrightarrow_{\mathscr{A}_\downarrow}, Ops)$ where $Ops \stackrel{\text{def}}{=} \{halt, store, dump, off\}$ and such that:

- $Halt^{\mathscr{A}_\downarrow} \stackrel{\text{def}}{=} \{q' \in Q^{\mathscr{A}_\downarrow} \mid \exists a \in \Sigma, \exists q \in Q^{\mathscr{A}_\downarrow}, q \xrightarrow{a/halt}_{\mathscr{A}_\downarrow} q'\}$,

$$\text{and} \quad \forall q \in Halt^{\mathscr{A}_\downarrow}, \forall a \in \Sigma, \forall \alpha \in Ops, \forall q' \in Q^{\mathscr{A}_\downarrow}, q \xrightarrow{a/\alpha}_{\mathscr{A}_\downarrow} q' \Rightarrow \alpha = halt$$

- $Off^{\mathscr{A}_\downarrow} \stackrel{\text{def}}{=} \{q' \in Q^{\mathscr{A}_\downarrow} \mid \exists a \in \Sigma, \exists q \in Q^{\mathscr{A}_\downarrow}, q \xrightarrow{a/off}_{\mathscr{A}_\downarrow} q'\}$,

$$\text{and} \quad \forall q \in Off^{\mathscr{A}_\downarrow}, \forall a \in \Sigma, \forall \alpha \in Ops, \forall q' \in Q^{\mathscr{A}_\downarrow}, q \xrightarrow{a/\alpha}_{\mathscr{A}_\downarrow} q' \Rightarrow \alpha = off$$

In the remainder of this article we consider only EMs.

Example 3 (Enforcement monitor) We illustrate the enforcement of some of the *e*-properties introduced in Example 2 with EMs.

- Figure 7b (p. 21) shows an EM $\mathscr{A}_{\downarrow\Pi_1}$ for the safety *e*-property Π_1. $\mathscr{A}_{\downarrow\Pi_1}$ has one halting state, $Halt^{\mathscr{A}_{\downarrow\Pi_1}} = \{3\}$, and its initial state is 1. From this initial state $\mathscr{A}_{\downarrow\Pi_1}$ simply *dumps* a first occurrence of *g_auth* and moves to state 2, where the op_s operation is allowed (i.e., *dumped*) and goes back to state 1. Otherwise, if the event op_s occurs while not being preceded by a *g_auth*, $\mathscr{A}_{\downarrow\Pi_1}$ moves to state 3 and halts the underlying program forever.
- Figure 6b (p. 20) shows an EM $\mathscr{A}_{\downarrow\Pi_2}$ for the guarantee *e*-property Π_2. The initial state of $\mathscr{A}_{\downarrow\Pi_2}$ is state 1, $Halt^{\mathscr{A}_{\downarrow\Pi_2}} = \{6\}$, and $Off^{\mathscr{A}_{\downarrow\Pi_2}} = \{2, 5\}$. Its behavior is the following. Occurrences of secured and unsecured operations are *stored* in memory until the answer of an authentication happens. If the authentication is granted, $\mathscr{A}_{\downarrow\Pi_2}$ dumps the whole memory content and switches off. Otherwise (denied authentication), according to whether the user tried to perform a secured operation or not, $\mathscr{A}_{\downarrow\Pi_2}$ either waits for the disconnection (forbidding any operation) and switches off, or halts immediately the underlying system.

4.4 Properties of enforcement monitors

We now study the properties of enforcement monitors with set of enforcement operations $\{halt, store, dump, off\}$.

Property 2 (About sequence transformation) For an execution sequence $\sigma \in Exec(\mathscr{P}_\Sigma) \cap \Sigma^\omega$ and an EM $\mathscr{A}_\downarrow$, s.t. the run of σ on $\mathscr{A}_\downarrow$ is expressed by

$$(q_0, \sigma_0/\alpha_0, q_1) \cdot (q_1, \sigma_1/\alpha_1, q_2) \cdots (q_i, \sigma_i/\alpha_i, q_{i+1}) \cdots,$$

the following properties hold:

- $\sigma \Downarrow_{\mathscr{A}_\downarrow} \sigma \Rightarrow \forall i \in \mathbb{N}, \exists j \in \mathbb{N}, i \leq j, \sigma_{\cdots j} \Downarrow_{\mathscr{A}_\downarrow} \sigma_{\cdots j}, \alpha_j \in \{dump, off\}$
- $\forall i \in \mathbb{N}, \exists j \in \mathbb{N}, i \leq j, \alpha_j \in \{dump, off\} \Rightarrow \sigma \Downarrow_{\mathscr{A}_\downarrow} \sigma$.

That is, for an EM, producing as output the same input sequence is equivalent to performing regularly a *dump* or a *off* operation.

Property 3 (Relation between input, memory, and output) Input execution sequence, memory content, and produced output are related by the following property: $\forall \sigma \in \Sigma^+$, $\forall \sigma' \in \Sigma^*$,

$$\exists q \in Q^{\mathscr{A}_\downarrow}, (q_{\text{init}}^{\mathscr{A}_\downarrow}, \sigma \cdot \sigma', \varepsilon) \stackrel{o}{\Rightarrow}_{\mathscr{A}_\downarrow} (q, \sigma', m)$$
$$\Rightarrow (\sigma = o \cdot m \wedge q \in Q^{\mathscr{A}_\downarrow} \setminus Halt^{\mathscr{A}_\downarrow}) \vee (o \prec \sigma \wedge q \in Halt^{\mathscr{A}_\downarrow})$$

Proof The proof can be found in Appendix A.2. It is done by induction on the length of the input sequence, according to the last enforcement operation performed. □

It follows that the equivalence relation considered for enforcement becomes the equality relation. This is due to the semantics of the enforcement operations we considered. Thus the enforcement predicate $Enf_{\approx}(\mathscr{A}_\downarrow, (\phi, \varphi), \mathscr{P}_\Sigma)$ becomes $Enf_{=}(\mathscr{A}_\downarrow, (\phi, \varphi), \mathscr{P}_\Sigma)$ (abbreviated $Enf(\mathscr{A}_\downarrow, (\phi, \varphi), \mathscr{P}_\Sigma)$ in the remainder of this article) when the e-property is enforced by $\mathscr{A}_\downarrow$ on $\mathscr{P}_\Sigma$. The following property is a straightforward consequence of Property 3 and the definition of enforcement operations.

Property 4 (Last enforcement operation and property satisfaction) Given an EM $\mathscr{A}_\downarrow$, an e-property Π s.t. $Enf(\mathscr{A}_\downarrow, \Pi, \mathscr{P}_\Sigma)$ and a finite execution sequence $\sigma \in Exec(\mathscr{P}_\Sigma) \cap \Sigma^+$ ($|\sigma| = n + 1$) which run on $\mathscr{A}_\downarrow$ is expressed $(q_0, \sigma_0/\alpha_0, q_1) \cdots (q_n, \sigma_n/\alpha_n, q_{n+1})$, we have:

- $\Pi(\sigma) \Rightarrow \alpha_n \in \{dump, off\}$
- $\neg\Pi(\sigma) \Rightarrow \alpha_n \in \{store, halt\}$

Meaning that, considering an EM which enforces an e-property, the last enforcement operation performed while reading an input sequence is *dump* or *off* (resp. *halt* or *store*) when the given sequence satisfies (resp. does not satisfy) the e-property.

Another consequence of these properties is that the produced output are always prefixes of the input execution sequence, that is: $\forall \sigma, o \in \Sigma^\infty, \sigma \Downarrow_{\mathscr{A}_\downarrow} o \Rightarrow o \preceq \sigma$.

5 Operations on enforcement monitors

Current development of information systems makes specifications going more and more complex. For assessing the value of EMs as a potential security mechanisms, it seems desirable to offer techniques to compose them so as to cope with their related specifications. In this section we describe and address the problem of EM composition. We give the formal definition of monitor composition w.r.t. Boolean combinations: union, intersection and negation, and prove their correctness.

5.1 Preliminary notations

We define the complete lattice $(Ops, \sqsubseteq)$ over enforcement operations, where $halt \sqsubseteq store \sqsubseteq dump \sqsubseteq off$ ($\sqsubseteq$ is a total order). Moreover, we define a negation operation on enforcement actions: for $\alpha \in Ops$, $\overline{\alpha}$ is the negation of α. We define $\overline{dump}$ as *store*, $\overline{off}$ as *halt*, and $\overline{\overline{\alpha}}$ as α.

5.2 Union and intersection

We show how disjunction (resp. conjunction) of basic (enforceable) properties can be enforced by constructing the union (resp. intersection) of their associated enforcement monitors. These operations between EMs are based on product constructions performed by combining enforcement operations w.r.t. the complete lattice $(Ops, \sqsubseteq)$.

Definition 14 (Union of EMs) Given two EMs $\mathscr{A}_{\downarrow 1} = (Q^{\mathscr{A}_{\downarrow 1}}, q_{\text{init}}^{\mathscr{A}_{\downarrow 1}}, \longrightarrow_{\mathscr{A}_{\downarrow 1}}, Ops)$, $\mathscr{A}_{\downarrow 2} = (Q^{\mathscr{A}_{\downarrow 2}}, q_{\text{init}}^{\mathscr{A}_{\downarrow 2}}, \longrightarrow_{\mathscr{A}_{\downarrow 2}}, Ops)$ defined relatively to a same input alphabet Σ, we define $\mathscr{A}_{\downarrow \sqcup} = \text{Union}(\mathscr{A}_{\downarrow 1}, \mathscr{A}_{\downarrow 2})$ with $Q^{\mathscr{A}_{\downarrow \sqcup}} = (Q^{\mathscr{A}_{\downarrow 1}} \times Q^{\mathscr{A}_{\downarrow 2}})$, $q_{\text{init}}^{\mathscr{A}_{\downarrow \sqcup}} = (q_{\text{init}}^{\mathscr{A}_{\downarrow 1}}, q_{\text{init}}^{\mathscr{A}_{\downarrow 2}})$. The transition relation of this enforcement monitor is defined by getting the supremum ($\sqcup$) of enforcement operations. More formally $\rightarrow_{\mathscr{A}_{\downarrow \sqcup}}: Q^{\mathscr{A}_{\downarrow \sqcup}} \times \Sigma \times Ops \rightarrow Q^{\mathscr{A}_{\downarrow \sqcup}}$ is defined as $\forall a \in \Sigma, \forall q = (q_1, q_2) \in Q^{\mathscr{A}_{\downarrow \sqcup}}$,

$$\frac{q_1 \xrightarrow{a/\alpha_1}_{\mathscr{A}_{\downarrow 1}} q_1' \qquad q_2 \xrightarrow{a/\alpha_2}_{\mathscr{A}_{\downarrow 2}} q_2'}{(q_1, q_2) \xrightarrow[\mathscr{A}_{\downarrow \sqcup}]{a/\bigsqcup(\{\alpha_1, \alpha_2\})} (q_1', q_2')} \mathscr{A}_{\downarrow \sqcup}$$

Note that $Halt^{\mathscr{A}_{\downarrow \sqcup}} = Halt^{\mathscr{A}_{\downarrow 1}} \times Halt^{\mathscr{A}_{\downarrow 2}}$ and $Off^{\mathscr{A}_{\downarrow \sqcup}} = Off^{\mathscr{A}_{\downarrow 1}} \times Q^{\mathscr{A}_{\downarrow 2}} \cup Q^{\mathscr{A}_{\downarrow 1}} \times Off^{\mathscr{A}_{\downarrow 2}}$. Notice also that this construction does not introduce non-determinism. Indeed, since the two initial EMs are deterministic, there is always one and only one transition with a given element of Σ in the resulting automaton. However, one can notice that it may be not minimal (as in Example 4).

The intersection operation between enforcement monitors is defined in a similar way by using the infimum operator $\sqcap$ between enforcement operations:

Definition 15 (Intersection of EMs) Given two EMs $\mathscr{A}_{\downarrow 1} = (Q^{\mathscr{A}_{\downarrow 1}}, q_{\text{init}}^{\mathscr{A}_{\downarrow 1}}, \longrightarrow_{\mathscr{A}_{\downarrow 1}}, Ops)$ and $\mathscr{A}_{\downarrow 2} = (Q^{\mathscr{A}_{\downarrow 2}}, q_{\text{init}}^{\mathscr{A}_{\downarrow 2}}, \longrightarrow_{\mathscr{A}_{\downarrow 2}}, Ops)$ defined relatively to a same input alphabet Σ and enforcement operations Ops, we define $\text{Intersection}(\mathscr{A}_{\downarrow 1}, \mathscr{A}_{\downarrow 2}) = \mathscr{A}_{\downarrow \sqcap}$ with $Q^{\mathscr{A}_{\downarrow \sqcap}} = (Q^{\mathscr{A}_{\downarrow 1}} \times Q^{\mathscr{A}_{\downarrow 2}})$, $q_{\text{init}}^{\mathscr{A}_{\downarrow \sqcap}} = (q_{\text{init}}^{\mathscr{A}_{\downarrow 1}}, q_{\text{init}}^{\mathscr{A}_{\downarrow 2}})$. The transition relation is defined by getting the infimum ($\sqcap$) of enforcement operations. More formally $\rightarrow_{\mathscr{A}_{\downarrow \sqcap}}: Q^{\mathscr{A}_{\downarrow \sqcap}} \times \Sigma \times Ops \rightarrow Q^{\mathscr{A}_{\downarrow \sqcap}}$ is defined as $\forall a \in \Sigma, \forall q = (q_1, q_2) \in Q^{\mathscr{A}_{\downarrow \sqcap}}$,

$$\frac{q_1 \xrightarrow{a/\alpha_1}_{\mathscr{A}_{\downarrow 1}} q_1' \qquad q_2 \xrightarrow{a/\alpha_2}_{\mathscr{A}_{\downarrow 2}} q_2'}{(q_1, q_2) \xrightarrow[\mathscr{A}_{\downarrow \sqcap}]{a/\bigsqcap(\{\alpha_1, \alpha_2\})} (q_1', q_2')} \mathscr{A}_{\downarrow \sqcap}$$

Note that $Halt^{\mathscr{A}_{\downarrow \sqcap}} = Halt^{\mathscr{A}_{\downarrow 1}} \times Q^{\mathscr{A}_{\downarrow 2}} \cup Q^{\mathscr{A}_{\downarrow 1}} \times Halt^{\mathscr{A}_{\downarrow 2}}$ and $Off^{\mathscr{A}_{\downarrow \sqcap}} = Off^{\mathscr{A}_{\downarrow 1}} \times Off^{\mathscr{A}_{\downarrow 2}}$.

Example 4 (Union of EMs) Let us consider a system on which it is possible to evaluate two atomic propositions a and b. At system runtime, events are fed to a monitor. Those events contain the evaluations of a and b: either true or false.

Now let us consider the following requirement: "Either a is always true or b will be eventually true". Meaning that, for the observed sequence of events, a is evaluated to true in every event or that in one of the event b is evaluated to true.

In order to build an EM for this requirement, we use two EMs, one for the requirement "a is always true", and the second for the requirement "b will be eventually true". Next, we build the union of EMs to obtain an EM for the initial requirement. The alphabet of the EMs

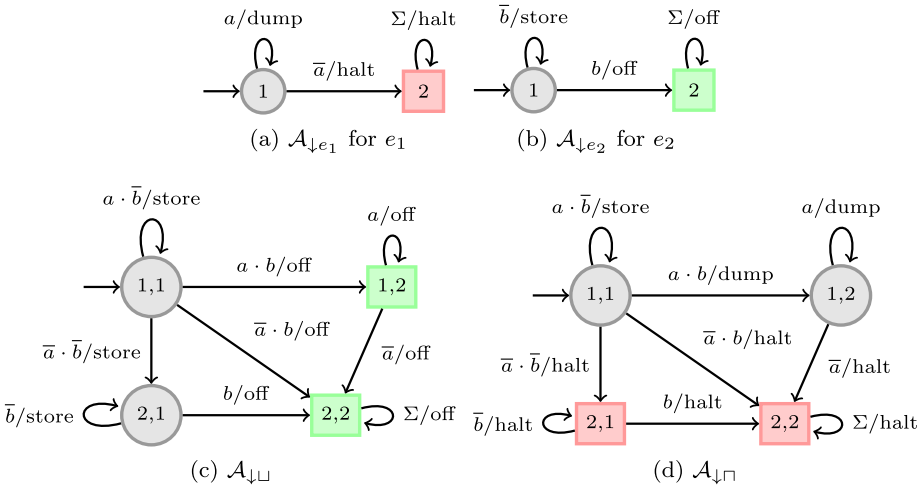


Fig. 4 Union and intersection of two enforcement monitors: $\mathscr{A}_{\downarrow e_1}$ and $\mathscr{A}_{\downarrow e_2}$

is made of all possible evaluations of the atomic propositions a and b, $\Sigma = \{\overline{a}\overline{b}, \overline{a}b, a\overline{b}, ab\}$. We use a Boolean notation, e.g., the event $a\overline{b}$ represents that a is evaluated to true and b to false, the event a means $ab \vee a\overline{b}$.

The EMs we consider are depicted in Fig. 4, states in *Halt* (resp. *Off*) are in red (resp. green).

- $\mathscr{A}_{\downarrow e_1}$ enforces the requirement "a is always true". $Halt^{\mathscr{A}_{\downarrow e_1}} = \{2\}$, $Off^{\mathscr{A}_{\downarrow e_1}} = \emptyset$.
- $\mathscr{A}_{\downarrow e_2}$ enforces the requirement "b is eventually true". $Halt^{\mathscr{A}_{\downarrow e_2}} = \emptyset$, $Off^{\mathscr{A}_{\downarrow e_2}} = \{2\}$.
- $\mathscr{A}_{\downarrow \sqcup}$ enforces the requirement "a is always true *or* b is eventually true". It is the EM union $\mathscr{A}_{\downarrow \sqcup}$ built from the EMs $\mathscr{A}_{\downarrow e_1}, \mathscr{A}_{\downarrow e_2}$. Following the definition of the construction, the set of states is the Cartesian product $Q^{\mathscr{A}_{\downarrow e_1}} \times Q^{\mathscr{A}_{\downarrow e_2}}$. The initial state is $(1,1)$. Note that there is no state in $Halt^{\mathscr{A}_{\downarrow \sqcup}}$ since $Halt^{\mathscr{A}_{\downarrow e_1}} \times Halt^{\mathscr{A}_{\downarrow e_2}} = \emptyset$. $\mathscr{A}_{\downarrow \sqcup}$ is not minimal and can be easily minimized by merging the states $(1,2)$ and $(2,2)$, which are states in $Off^{\mathscr{A}_{\downarrow \sqcup}}$. One can notice that $\mathscr{A}_{\downarrow \sqcup}$ complies to the constraints for states in $Halt^{\mathscr{A}_{\downarrow \sqcup}}$ and $Off^{\mathscr{A}_{\downarrow \sqcup}}$.

Example 5 (Intersection of EMs) Similarly to Example 4, we build an enforcement monitor for the requirement "a is always true *and* b is eventually true" by using the intersection construction. The resulting EM $\mathscr{A}_{\downarrow \sqcap}$ is shown in Fig. 4d. $Halt^{\mathscr{A}_{\downarrow \sqcap}} = \{(2,1),(2,2)\}$ and $Off^{\mathscr{A}_{\downarrow \sqcap}} = \emptyset$. This EM is not minimal and can be easily minimized by merging the states $(2,1)$ and $(2,2)$.

Theorem 2 (Union and Intersection of EMs) *Given two EMs* $\mathscr{A}_{\downarrow \Pi_1} = (Q^{\mathscr{A}_{\downarrow \Pi_1}}, q_{\text{init}}^{\mathscr{A}_{\downarrow \Pi_1}}, \longrightarrow_{\mathscr{A}_{\downarrow \Pi_1}}, Ops)$ *and* $\mathscr{A}_{\downarrow \Pi_2} = (Q^{\mathscr{A}_{\downarrow \Pi_2}}, q_{\text{init}}^{\mathscr{A}_{\downarrow \Pi_2}}, \longrightarrow_{\mathscr{A}_{\downarrow \Pi_2}}, Ops)$, *enforcing two enforceable properties* $\Pi_1, \Pi_2 \in EP$ *on a program* $\mathscr{P}_\Sigma$, *the property* $\Pi_1 \vee \Pi_2$ (*resp.* $\Pi_1 \wedge \Pi_2$) *is enforced by the union* (*resp. intersection*) *enforcement monitor. More formally*: $\forall \mathscr{A}_{\downarrow \Pi_1}, \mathscr{A}_{\downarrow \Pi_2}$,

- $\text{Union}(\mathscr{A}_{\downarrow \Pi_1}, \mathscr{A}_{\downarrow \Pi_2})$ *and* $\text{Intersection}(\mathscr{A}_{\downarrow \Pi_1}, \mathscr{A}_{\downarrow \Pi_2})$ *are EMs*
- $\begin{cases} Enf(\mathscr{A}_{\downarrow \Pi_1}, \Pi_1, \mathscr{P}_\Sigma) \\ Enf(\mathscr{A}_{\downarrow \Pi_2}, \Pi_2, \mathscr{P}_\Sigma) \end{cases} \Longrightarrow \begin{cases} Enf(\text{Union}(\mathscr{A}_{\downarrow \Pi_1}, \mathscr{A}_{\downarrow \Pi_2}), \Pi_1 \vee \Pi_2, \mathscr{P}_\Sigma) \\ Enf(\text{Intersection}(\mathscr{A}_{\downarrow \Pi_1}, \mathscr{A}_{\downarrow \Pi_2}), \Pi_1 \wedge \Pi_2, \mathscr{P}_\Sigma) \end{cases}$

The proof of this theorem can be found in Appendix A.3.

5.3 Negation

Considering a safety or guarantee (enforceable) e-property,[8] we show how to construct an EM enforcing the negation of the original e-property.

Definition 16 (Negation of an EM) Given an EM $\mathscr{A}_{\downarrow\Pi} = (Q^{\mathscr{A}_{\downarrow\Pi}}, q_{\text{init}}^{\mathscr{A}_{\downarrow\Pi}}, \longrightarrow_{\mathscr{A}_{\downarrow\Pi}}, Ops)$ defined relatively to an input alphabet Σ and enforcing Π, a safety or guarantee e-property, we define $Negation(\mathscr{A}_{\downarrow\Pi}) = \overline{\mathscr{A}_{\downarrow\Pi}} = (Q^{\overline{\mathscr{A}_{\downarrow\Pi}}}, q_{\text{init}}^{\overline{\mathscr{A}_{\downarrow\Pi}}}, \longrightarrow_{\overline{\mathscr{A}_{\downarrow\Pi}}}, Ops)$ as:

- $Q^{\overline{\mathscr{A}_{\downarrow\Pi}}} = Q^{\mathscr{A}_{\downarrow\Pi}}$, $q_{\text{init}}^{\overline{\mathscr{A}_{\downarrow\Pi}}} = q_{\text{init}}^{\mathscr{A}_{\downarrow\Pi}}$,
- $\rightarrow_{\overline{\mathscr{A}_{\downarrow\Pi}}}$ is the smallest relation verifying $q \xrightarrow{a/\overline{\alpha}}_{\overline{\mathscr{A}_{\downarrow\Pi}}} q'$ if $q \xrightarrow{a/\alpha}_{\mathscr{A}_{\downarrow\Pi}} q'$.

Note that $Halt^{\overline{\mathscr{A}_{\downarrow\Pi}}} = Off^{\mathscr{A}_{\downarrow\Pi}}$ and $Off^{\overline{\mathscr{A}_{\downarrow\Pi}}} = Halt^{\mathscr{A}_{\downarrow\Pi}}$.

Example 6 In Fig. 4, $\mathscr{A}_{\downarrow e_2}$ is the negation of $\mathscr{A}_{\downarrow e_1}$ if we replace $\overline{b}$ with a and b with $\overline{a}$.

Theorem 3 (Negation of an EM) *Given an EM $\mathscr{A}_{\downarrow\Pi} = (Q^{\mathscr{A}_{\downarrow\Pi}}, q_{\text{init}}^{\mathscr{A}_{\downarrow\Pi}}, \longrightarrow_{\mathscr{A}_{\downarrow\Pi}}, Ops)$ defined relatively to an input alphabet Σ and enforcing Π, a safety or guarantee e-property, the EM* Negation$(\mathscr{A}_{\downarrow\Pi})$ *enforces $\overline{\Pi}$. More formally*: $\forall \mathscr{A}_{\downarrow\Pi}$

$$\Pi \in \text{Safety}(\Sigma) \cup \text{Guarantee}(\Sigma) \wedge Enf(\mathscr{A}_{\downarrow\Pi}, \Pi, \mathscr{P}_\Sigma)$$
$$\Rightarrow \text{Negation}(\mathscr{A}_{\downarrow\Pi}) \textit{ is an EM } \wedge Enf(\text{Negation}(\mathscr{A}_{\downarrow\Pi}), \overline{\Pi}, \mathscr{P}_\Sigma)$$

The proof can be found in Appendix A.4.

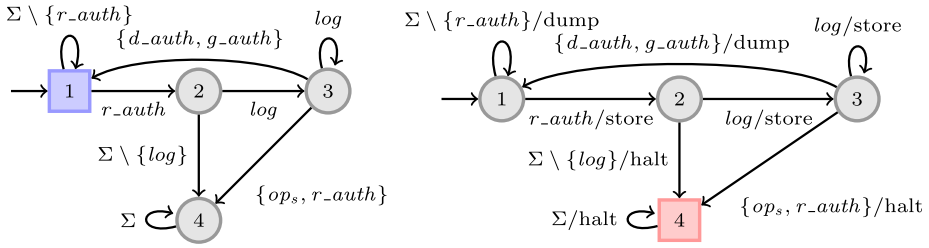
6 Enforcement w.r.t. the Safety-Progress classification

We now study how to practically enforce e-properties of the Safety-Progress hierarchy (Sect. 3). More precisely, we show which classes of properties can be effectively enforced by an EM, and more important, we provide a systematic construction of an EM for an e-property $\Pi \in EP$ from the Streett automaton defining this e-property.

6.1 From a recognizing automaton to an enforcement monitor

We define two general operations whose purpose is to transform a Streett automaton recognizing an enforceable e-property into an enforcement monitor enforcing the same e-property. The following operations use the set $Reach_{\mathscr{A}_\Pi}(q)$ of reachable states from a state q in $\mathscr{A}_\Pi$ (denoted $Reach(q)$ when clear from context). Given a Streett automaton $\mathscr{A}_\Pi$ with a set of states $Q^{\mathscr{A}_\Pi}$, we have $\forall q \in Q^{\mathscr{A}_\Pi}, Reach_{\mathscr{A}_\Pi}(q) = \{q' \in Q^{\mathscr{A}_\Pi} \mid \exists (q_i)_i, (a_i)_i, q \xrightarrow{a_0}_{\mathscr{A}_\Pi} q_0 \xrightarrow{a_1}_{\mathscr{A}_\Pi} q_1 \cdots q'\}$.

[8] It is only useful to deal with safety and guarantee e-properties: the negation of a response e-property is a persistence (thus not enforceable), and obligation e-properties can be always written under conjunctive normal form as a Boolean combination of safety and guarantee e-properties (Lemma 1).



(a) Response automaton for Π_4 (b) EM for Π_4

Fig. 5 A response-automaton and the corresponding EM for property Π_4

6.1.1 Response e-properties

Definition 17 (Transformation for response *e*-properties) Given a Streett response automaton $\mathscr{A}_\Pi = (Q^{\mathscr{A}_\Pi}, q_{\text{init}}^{\mathscr{A}_\Pi}, \Sigma, \longrightarrow_{\mathscr{A}_\Pi}, \{(R, \emptyset)\})$ recognizing a response (enforceable) *e*-property $\Pi \in \mathit{Response}(\Sigma)$, we define the transformation TransResponse$(\mathscr{A}_\Pi) = \mathscr{A}_{\downarrow\Pi} = (Q^{\mathscr{A}_{\downarrow\Pi}}, q_{\text{init}}^{\mathscr{A}_{\downarrow\Pi}}, \longrightarrow_{\mathscr{A}_{\downarrow\Pi}}, \mathit{Ops})$ using the following rules for $\rightarrow_{\mathscr{A}_{\downarrow\Pi}}$:

- $q \xrightarrow{a/\mathit{off}}_{\mathscr{A}_{\downarrow\Pi}} q'$ if $q' \in R \wedge q \xrightarrow{a}_{\mathscr{A}_\Pi} q' \wedge \mathit{Reach}_{\mathscr{A}_\Pi}(q') \subseteq R$ (TRESP1)
- $q \xrightarrow{a/\mathit{dump}}_{\mathscr{A}_{\downarrow\Pi}} q'$ if $q' \in R \wedge q \xrightarrow{a}_{\mathscr{A}_\Pi} q' \wedge \mathit{Reach}_{\mathscr{A}_\Pi}(q') \nsubseteq R$ (TRESP2)
- $q \xrightarrow{a/\mathit{store}}_{\mathscr{A}_{\downarrow\Pi}} q'$ if $q' \notin R \wedge q \xrightarrow{a}_{\mathscr{A}_\Pi} q' \wedge \mathit{Reach}_{\mathscr{A}_\Pi}(q') \nsubseteq \overline{R}$ (TRESP3)
- $q \xrightarrow{a/\mathit{halt}}_{\mathscr{A}_{\downarrow\Pi}} q'$ if $q' \notin R \wedge q \xrightarrow{a}_{\mathscr{A}_\Pi} q' \wedge \mathit{Reach}_{\mathscr{A}_\Pi}(q') \subseteq \overline{R}$ (TRESP4)

An EM $\mathscr{A}_{\downarrow\Pi}$ obtained via the TransResponse$(\mathscr{A}_\Pi)$ transformation, applied to an automaton $\mathscr{A}_\Pi$ recognizing a response *e*-property Π, processes the input execution sequence and enforces Π. While the current execution sequence does not satisfy Π (the current state is in $\overline{R}$), it stores each event of the input sequence (or halts the underlying program if Π can not be satisfied in the future). Once the execution sequence satisfies Π (the current state is in R), it dumps the content of the memory and the events stored so far (or switches off if Π is satisfied for ever).

Example 7 (Transformation for response *e*-properties) The right-hand side of Fig. 5 shows the EM $\mathscr{A}_{\downarrow\Pi_4}$ enforcing the response *e*-property Π_4, and obtained by TransResponse applied to $\mathscr{A}_{\Pi_4}$. We have $\mathit{Halt}^{\mathscr{A}_{\downarrow\Pi_4}} = \{4\}$ and $\mathit{Off}^{\mathscr{A}_{\downarrow\Pi_4}} = \emptyset$.

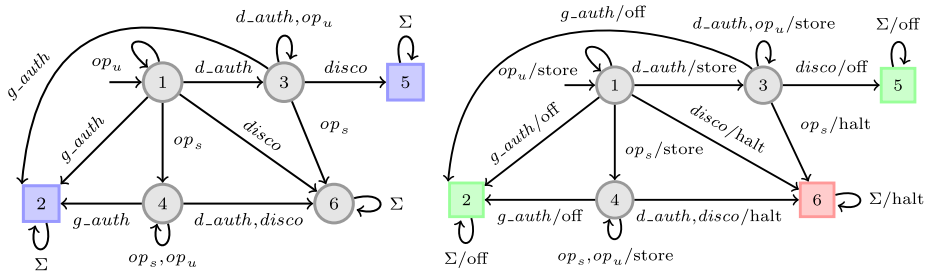
6.1.2 Guarantee e-properties

The TransResponse transformation can be directly applied to guarantee properties. Indeed, in guarantee automata, transitions leading from R-states to $\overline{R}$-states are absent. Thus the TransResponse transformation is applied for guarantee automata by ignoring (TRESP2).

Example 8 (Transformation for guarantee *e*-properties) Figure 6b shows the EM enforcing Π_2, obtained by TransResponse on $\mathscr{A}_{\Pi_2}$. $\mathit{Halt}^{\mathscr{A}_{\downarrow\Pi_2}} = \{6\}$ and $\mathit{Off}^{\mathscr{A}_{\downarrow\Pi_2}} = \{2, 5\}$.

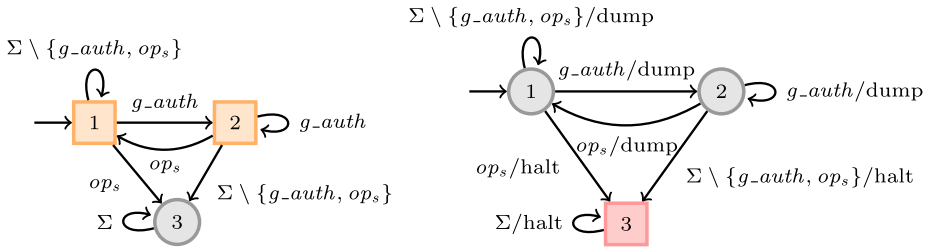
6.1.3 Safety e-properties

For safety *e*-properties, the TransResponse transformation can be also applied by "seeing" the underlying Streett safety automaton as a response automaton. We first notice that a safety



(a) Guarantee automaton for Π_2 (b) EM for Π_2

Fig. 6 A guarantee-automaton and the corresponding EM for property Π_2



(a) Safety automaton for Π_1 (b) EM for Π_1

Fig. 7 A safety-automaton and the corresponding EM for property Π_1

e-property with safety automaton $\mathscr{A}_\Pi = (Q^{\mathscr{A}_\Pi}, q_{\text{init}}^{\mathscr{A}_\Pi}, \Sigma, \longrightarrow_{\mathscr{A}_\Pi}, \{(\emptyset, P)\})$ can be recognized by the response automaton $\mathscr{A}'_\Pi = (Q^{\mathscr{A}_\Pi}, q_{\text{init}}^{\mathscr{A}_\Pi}, \Sigma, \longrightarrow_{\mathscr{A}_\Pi}, \{(P, \emptyset)\})$: same states and transitions, but different accepting conditions. P-states of $\mathscr{A}_\Pi$ become R-states of $\mathscr{A}'_\Pi$. These automata recognize the same sequences. Indeed, since there is no transition in $\mathscr{A}_\Pi$ from $\overline{P}$-states to P-states, there is no transition from $\overline{R}$-states to R-states in $\mathscr{A}'_\Pi$. According to the acceptance conditions (Definitions 2 and 3) and transition restrictions, for $\mathscr{A}_\Pi$ and $\mathscr{A}'_\Pi$ there is no difference between P-states and R-states regarding the role they play in the acceptance condition. Thus, using TransResponse on $\mathscr{A}'_\Pi$ gives an enforcement monitor for Π.

Example 9 (Transformation for safety e-properties) The right-hand side of Fig. 7 shows the EM $\mathscr{A}_{\downarrow\Pi_1}$ obtained by first converting $\mathscr{A}_{\Pi_1}$ into a response automaton and by then applying the transformation for response e-properties (i.e., TransResponse) to $\mathscr{A}_{\Pi_1}$.

6.1.4 Obligation e-properties

Since an obligation e-property can be written as intersection of union of safety and guarantee e-properties (Lemma 1), it is possible to obtain an EM for an *obligation* property by using the TransResponse transformation and the *Union* and *Intersection* operations. However, building such an EM requires first to express the obligation property in conjunctive normal form, and second the knowledge of the associated Streett safety and guarantee automata. Thus, we also define a direct transformation for obligation automata.

Definition 18 (Transformation for obligation e-properties) Given a Streett m-obligation automaton $\mathscr{A}_\Pi = (Q^{\mathscr{A}_\Pi}, q_{\text{init}}^{\mathscr{A}_\Pi}, \Sigma, \longrightarrow_{\mathscr{A}_\Pi}, \{(R_1, P_1), \ldots, (R_m, P_m)\})$ recognizing an m-obligation (enforceable) e-property $\Pi \in \mathit{Obligation}(\Sigma)$, we define the transformation $\text{TransObligation}(\mathscr{A}_\Pi) = \mathscr{A}_{\downarrow\Pi} = (Q^{\mathscr{A}_{\downarrow\Pi}}, q_{\text{init}}^{\mathscr{A}_{\downarrow\Pi}}, \longrightarrow_{\mathscr{A}_{\downarrow\Pi}}, \mathit{Ops})$ s.t.:

- $Q^{\mathscr{A}_{\downarrow\Pi}} = Q^{\mathscr{A}_\Pi}, q_{\text{init}}^{\mathscr{A}_{\downarrow\Pi}} = q_{\text{init}}^{\mathscr{A}_\Pi}$,
- $\rightarrow_{\mathscr{A}_{\downarrow\Pi}}$ is defined as the smallest relation verifying:
  $q \xrightarrow{a/\alpha}_{\mathscr{A}_{\downarrow\Pi}} q'$ if $q \xrightarrow{a}_{\mathscr{A}_\Pi} q'$ and $\alpha = \bigsqcap_{i=1}^{m}(\bigsqcup(\beta_i, \gamma_i))$ where the β_i and γ_i are obtained in the following way:
  - $\beta_i = \mathit{off}$ if $q' \in P_i \wedge \mathit{Reach}_{\mathscr{A}_\Pi}(q') \subseteq P_i$,
  - $\beta_i = \mathit{dump}$ if $q' \in P_i \wedge \mathit{Reach}_{\mathscr{A}_\Pi}(q') \not\subseteq P_i$,
  - $\beta_i = \mathit{halt}$ if $q' \notin P_i$,
  - $\gamma_i = \mathit{off}$ if $q' \in R_i$,
  - $\gamma_i = \mathit{store}$ if $q' \notin R_i \wedge \mathit{Reach}_{\mathscr{A}_\Pi}(q') \not\subseteq \overline{R_i}$,
  - $\gamma_i = \mathit{halt}$ if $q' \notin R_i \wedge \mathit{Reach}_{\mathscr{A}_\Pi}(q') \subseteq \overline{R_i}$.

Note that there is no transition from $q \in R_i$ to $q' \in \overline{R_i}$, and no transition from $q \in \overline{P_i}$ to $q' \in P_i$. One can notice, as a direct consequence of the definition of $\rightarrow_{\mathscr{A}_{\downarrow\Pi}}$, that:

- $\mathit{Halt}^{\mathscr{A}_{\downarrow\Pi}} = \{q \in \bigcup_{i=1}^{m}(\overline{P_i} \cap \overline{R_i}) \mid \mathit{Reach}_{\mathscr{A}_\Pi}(q) \subseteq \bigcup_{i=1}^{m}(\overline{P_i} \cap \overline{R_i})\}$, and
- $\mathit{Off}^{\mathscr{A}_{\downarrow\Pi}} = \{q \in \bigcap_{i=1}^{m}(P_i \cup R_i) \mid \mathit{Reach}_{\mathscr{A}_\Pi}(q) \subseteq \bigcap_{i=1}^{m}(P_i \cup R_i)\}$.

We note $\mathscr{A}_{\downarrow\Pi} = \text{TransObligation}(\mathscr{A}_\Pi)$.

Example 10 (Transformation for obligation e-properties) In Fig. 8b is depicted the EM enforcing the 1-obligation property Π_3 of Example 2, obtained by the TransObligation transformation. $\mathit{Halt}^{\mathscr{A}_{\downarrow\Pi_3}} = \{4\}$ and $\mathit{Off}^{\mathscr{A}_{\downarrow\Pi_3}} = \{5\}$.

6.2 Enforcement w.r.t. the Safety-Progress classification

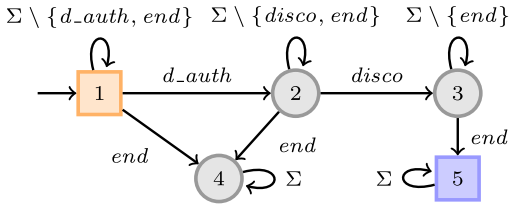
Using the aforementioned transformations it is possible to derive an EM for a given regular (enforceable) property from its recognizing finite-state automaton. In the following, we prove the correctness of the transformations. Furthermore, we discuss and justify the enforcement limitation for non-enforceable properties.

6.2.1 Enforceable properties

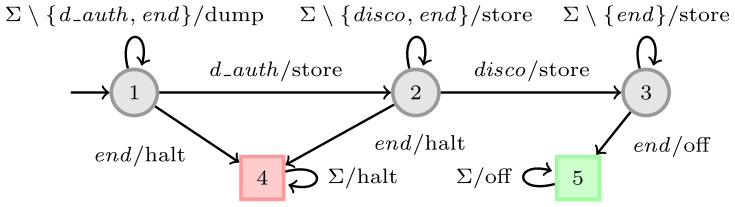
Given *any* safety (resp. guarantee, obligation, response) Streett automaton recognizing a property Π, one can *synthesize* an enforcing monitor for Π using the systematic transformations previously presented. The following theorem proves the correctness of these transformations. It also proves that safety, guarantee, obligation, and response properties are enforceable by EMs.

Theorem 4 (Correctness of the transformations) *Given a program $\mathscr{P}_\Sigma$, a* regular *safety* (*resp. guarantee*, *obligation*, *response*) *e-property Π is enforceable on $\mathscr{P}_\Sigma$ by an EM obtained by the application of the previous transformations on the automaton recognizing Π. More formally*:

$$(\Pi \in \text{Guarantee}(\Sigma) \wedge \mathscr{A}_{\downarrow\Pi} = \text{TransResponse}(\mathscr{A}_\Pi)) \Rightarrow \mathit{Enf}(\mathscr{A}_{\downarrow\Pi}, \Pi, \mathscr{P}_\Sigma),$$
$$(\Pi \in \text{Obligation}(\Sigma) \wedge \mathscr{A}_{\downarrow\Pi} = \text{TransObligation}(\mathscr{A}_\Pi)) \Rightarrow \mathit{Enf}(\mathscr{A}_{\downarrow\Pi}, \Pi, \mathscr{P}_\Sigma),$$
$$(\Pi \in \text{Response}(\Sigma) \wedge \mathscr{A}_{\downarrow\Pi} = \text{TransResponse}(\mathscr{A}_\Pi)) \Rightarrow \mathit{Enf}(\mathscr{A}_{\downarrow\Pi}, \Pi, \mathscr{P}_\Sigma),$$



(a) Obligation automaton for Π_3



(b) EM for Π_3

Fig. 8 A 1-obligation-automaton and the corresponding EM for property Π_3

$$
\begin{aligned}
&(\Pi \in \text{Safety}(\Sigma) \wedge \mathscr{A}_\Pi = (Q^{\mathscr{A}_\Pi}, q_{\text{init}}^{\mathscr{A}_\Pi}, \Sigma, \longrightarrow_{\mathscr{A}_\Pi}, (\emptyset, P))) \\
&\quad \Rightarrow \mathscr{A}_{\downarrow\Pi} = \text{TransResponse}((Q^{\mathscr{A}_\Pi}, q_{\text{init}}^{\mathscr{A}_\Pi}, \Sigma, \longrightarrow_{\mathscr{A}_\Pi}, (P, \emptyset))) \\
&\quad \Rightarrow \mathit{Enf}(\mathscr{A}_{\downarrow\Pi}, \Pi, \mathscr{P}_\Sigma).
\end{aligned}
$$

Proof We have to show that $\forall \sigma \in \mathit{Exec}(\mathscr{P}_\Sigma), \exists o \in \Sigma^*$,

$$\sigma \Downarrow_{\mathscr{A}_{\downarrow\Pi}} o \tag{5}$$

$$\Pi(\sigma) \Rightarrow \sigma = o \tag{6}$$

$$\neg\Pi(\sigma) \wedge \mathit{Pref}_\prec(\phi, \sigma) = \emptyset \Rightarrow o = \varepsilon \tag{7}$$

$$\neg\Pi(\sigma) \wedge \mathit{Pref}_\prec(\phi, \sigma) \neq \emptyset \Rightarrow o = \mathit{Max}(\mathit{Pref}_\prec(\phi, \sigma)) \tag{8}$$

Note first that we only need to prove the correctness of TransResponse and TransObligation. We note $\mathscr{A}_{\downarrow\Pi} = (Q^{\mathscr{A}_{\downarrow\Pi}}, q_{\text{init}}^{\mathscr{A}_{\downarrow\Pi}}, \longrightarrow_{\mathscr{A}_{\downarrow\Pi}}, \mathit{Ops})$ the EM obtained from the transformation. We only sketch the proofs, the full versions can be found in Appendixes A.5 and A.6.

For the TransResponse *transformation* we examine the run of an execution sequence $\sigma \in \mathit{Exec}(\mathscr{P}_\Sigma)$, and, using the definition of TransResponse, we deduce the shape of the sequence of enforcement operations performed by $\mathscr{A}_{\downarrow\Pi}$.

– The first case is $\Pi(\sigma)$. We distinguish whether σ is finite or not.
  - If σ is a finite sequence, it means that the run of σ on $\mathscr{A}_\Pi$ ends in a R-state. Hence, the last enforcement operation performed by $\mathscr{A}_{\downarrow\Pi}$ is either *dump* or *off*. The shape of the sequence of enforcement operations is $(\mathit{store} + \mathit{dump})^* \cdot (\mathit{dump} + \mathit{off}^*)$.
  - If σ is an infinite sequence, it means that an R-state is visited infinitely often. Hence, $\mathscr{A}_{\downarrow\Pi}$ performs regularly the *dump* operation or persistently the *off* operation. Then the shape of the sequence of enforcement operations is $(\mathit{store}^* \cdot \mathit{dump})^\omega + ((\mathit{store} + \mathit{dump})^* \cdot \mathit{off}^\omega)$.

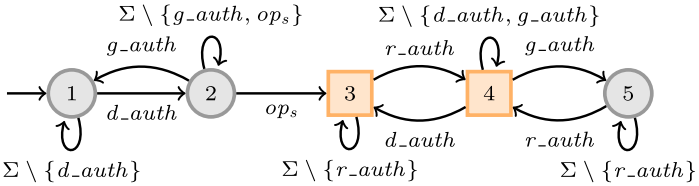


Fig. 9 Automaton recognizing the persistence *e*-property Π_5

– The second case is $\neg\Pi(\sigma)$. We distinguish whether σ is finite or not.
  - If σ is a finite sequence, it means that the run of σ on $\mathscr{A}_\Pi$ ends in a $\overline{R}$-state. Hence, the last enforcement operation performed by $\mathscr{A}_{\downarrow\Pi}$ is *store* or *halt*. The shape of the sequence of enforcement operations is $(store + dump)^* \cdot (store + halt^*)$.
  - If σ is an infinite sequence, it means that R-states are visited *finitely* often. Hence, $\mathscr{A}_{\downarrow\Pi}$ performs always the *halt* or the *store* operation from a certain prefix of σ. Then the shape of the sequence of enforcement operations is $(store + dump)^* \cdot (halt + store)^\omega$.

For the TransObligation *transformation* we perform an induction on k where Π is a k-obligation *e*-property.

– *Induction basis.* We have $k = 1$, Π is a simple obligation recognized by a 1-obligation automaton $\mathscr{A}_\Pi = (Q^{\mathscr{A}_\Pi}, q_{\text{init}}^{\mathscr{A}_\Pi}, \Sigma, \longrightarrow_{\mathscr{A}_\Pi}, \{(R, P)\})$. The proof is done by showing that the two following EMs are equal:
  - The first EM is obtained by decomposing Π into a conjunction of a safety and a guarantee *e*-property. Then, we apply the TransResponse transformation to obtain two EMs to which we apply the Intersection construction. The resulting EM is correct by construction.
  - The second EM is obtained by applying directly TransObligation to $\mathscr{A}_\Pi$.
– *Induction step.* The proof is done by showing that the two following EMs are equal:
  - The first EM is obtained by decomposing the $(k + 1)$-obligation automaton into the intersection of one simple obligation and one k-obligation automata, using Lemma 1. Then, we apply the TransObligation transformation to the two obligation automata and the Intersection operation. The resulting EM is correct by construction.
  - The second EM is obtained by the direct application of TransObligation on the automaton recognizing the $(k + 1)$-obligation property Π.

  The equality is shown by exhibiting a bijection between those EMs. □

6.2.2 *Non-enforceable properties*

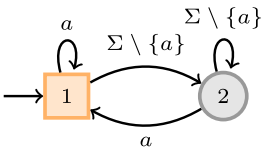
Pure persistence properties are not enforceable by our enforcement monitors and by any enforcement mechanism complying to the soundness and transparency constraints [16]. By discussing two examples of pure persistence properties, we explain with more details than in [16] the enforcement limitation (Example 11) and why it is not desirable to enforce pure persistence properties in practice (Example 12).

Example 11 (Non-enforceable pure persistence properties) Let us go back on the *e*-property Π_5 recognized by the Streett automaton in Fig. 9 (with acceptance criterion $vinf(\sigma, \mathscr{A}_{\Pi_5}) \subseteq P$ and $P = \{1, 3\}$). This property is not enforceable since it has incorrect infinite sequences with an infinite number of correct prefixes. Indeed consider $\sigma_{bad} = d\_auth \cdot op_s \cdot$

$(r\_auth \cdot d\_auth \cdot r\_auth \cdot g\_auth)^{\omega}$. Such an (infinite) execution sequence does not satisfy Π_5 since $vinf(\sigma_{bad}, \mathscr{A}_{\Pi_5}) = \{3, 4, 5\} \not\subseteq \{3, 4\}$. Moreover according to the acceptance criterion for finite sequences, each prefix σ'_{bad} of the form $d\_auth \cdot op_s \cdot (r\_auth \cdot d\_auth \cdot r\_auth \cdot g\_auth)^* \cdot r\_auth \cdot d\_auth$ satisfies the property Π_5. We have exhibited an infinite incorrect execution sequence with no longest correct prefix.

The following example permits to understand why it would be unrealistic and undesirable to enforce pure persistence properties.

Example 12 (Non-enforceable pure persistence properties) An example of (pure) persistence property, defined on $\Sigma \supset \{a\}$, is $\Sigma^* \cdot a^{\omega}$ stating that "it will be eventually true that the event a always occurs". This property can be formalized by the persistence automaton on the left with $P = \{1\}$. This property is neither a safety, nor a guarantee and nor an obligation property. As in the previous example, this property admits infinite incorrect sequences with an infinite number of correct prefixes.



One can explain the enforcement limitation intuitively with the following argument: if this property was enforceable it would imply that an enforcement monitor could 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. From a more formal perspective, the enforcement limitation can be understood as follows. As stated in Sect. 4.2, an e-property (ϕ, φ) is enforceable if for all infinite execution sequences of the program when $\neg\varphi(\sigma)$, the longest prefix of σ satisfying ϕ $(Max_{\prec}(Pref(\phi, \sigma))$ always exists; which is not the case for this property.

Suppose that we try to build a *sound and transparent* enforcement monitor for the property "it will be eventually true that a always occur". Now, suppose that $b \in \Sigma$ and the sequence $(a \cdot b)^{\omega}$ is submitted in input to such a monitor:

- When receiving a, the monitor has to output the sequence a. Indeed, a is correct w.r.t. the e-property and it is the longest correct prefix of the input sequence.
- When receiving $a \cdot b$, the monitor does not produce a new output (the output is still a). Indeed, $a \cdot b$ is incorrect w.r.t. the e-property.
- When receiving $a \cdot b \cdot a$, the monitor has to output the sequence $a \cdot b \cdot a$. Indeed, $a \cdot b \cdot a$ is correct w.r.t. the e-property and it is the longest correct prefix of the input sequence.

Thus the enforcement monitor would output the input sequence: $(a \cdot b)^{\omega}$; which is not correct w.r.t. the considered e-property.

Remark 1 As a consequence, properties of the reactivity class (containing the persistence class) are not enforceable by our enforcement monitors.

7 Related work and discussion

This section compares our results with related work in runtime enforcement monitoring. Moreover, we refer to the comparison of enforcement mechanisms provided in [4] as it sets up enforcement at runtime w.r.t. other enforcement mechanisms from a computational point of view.

7.1 Computability power of enforcement mechanisms

Hamlen, Morisett and Schneider proposed in [4, 17] a classification of enforceable properties considering a program as a Turing machine. Their purpose was to delineate the set of enforceable properties according to the mechanism used for the enforcement purpose. Properties are classified according to the modifications that the enforcement mechanism can perform on the underlying program. Notably each mechanism corresponds to a computability class of property:

- *Properties enforceable by static analysis of the underlying program.* These are decidable properties on the underlying program.
- *Properties enforceable by runtime execution monitor.* These are co-recursively enumerable properties.
- *Properties enforceable by program rewriting.* The set of enforceable properties depends on the equivalence relation used between programs.

By modifying the execution sequence, our enforcement monitor can be seen as a restricted form of program rewriting (also noticed in [4]). However we believe that the proposed mechanism can be affixed to a program using the constraints of a runtime mechanism. It seems to us a good trade-off between pure runtime monitoring and program rewriting, in the sense that we give more enforcement capability to our mechanism without any modification of the underlying program. The only control we need on the underlying program is being able to "encapsulate" events and delay them with minimal semantic impacts. There the EMs introduced in this paper can be framed in the runtime execution monitor category of enforcement mechanisms. In the following we focus on related works dedicated to this category.

7.2 Characterizing the set of enforceable properties in the Safety-Progress hierarchy independently from any enforcement mechanism [16]

In [16], we presented a unified view of runtime verification and enforcement of properties in the Safety-Progress classification. We characterized the set of properties which can be verified (monitorable properties) and enforced (enforceable properties) at runtime. In particular, we proposed an alternative definition of "property monitoring" to the one classically used in this context. This definition is parameterized by a truth domain of interest, and we showed that it better suits practical needs of runtime verification tools. However, these characterizations were independent of any specific runtime enforcement mechanism, and they did not tell how to build an enforcement monitor from a property.

7.3 Characterizing the set of enforceable properties with execution monitors

Security automata and decidable safety properties Schneider introduced *security automata* as the first runtime mechanism for enforcing properties. In [3], he defined a variant of Büchi automaton which runs in parallel with an underlying program. These automata are able to halt the program whenever the security automaton detects a violation of the property under scrutiny. Schneider announced in this paper that the set of enforceable properties with security automata is the set of safety properties. Then in [4] Schneider, Hamlen and Morisett refined the set of enforceable properties using such a mechanism. They showed that security automata are in fact restrained by some computational limits. Indeed, Viswanathan noticed in [5] that the class of enforceable properties is impacted by the computational power of the enforcement monitor. As the enforcement mechanism can implement no more than computable functions, the enforceable properties are included in the decidable ones. Hence, it

is shown in [4] that the set of safety properties is a strict superior limit to the power of (execution) enforcement monitors defined as security automata. Since in this article we are focusing on the regular fragment of safety properties, this fragment corresponds to the set of enforceable properties with security automata and the set of enforceable properties as defined in [4, 5].

Edit-automata and infinite renewal properties [6, 7, 18, 19] Ligatti et al. introduced *edit-automata* as runtime execution monitors. They noticed that, by only halting the program, the original security automata of Schneider were too restricted. 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 the current input (possibly memorized in the control state for later on). Enforcement with edit-automata was studied under the soundness and transparency constraints. Thus, the insertions of events were performed after suppressions in order to produce an output sequence which is always a prefix of the input sequence.

The properties enforced by edit-automata are called *infinite renewal* properties. They have been defined as the properties for which every infinite valid sequence has an infinite number of valid prefixes [6]. The set of renewal properties is a super-set of safety properties and contains some liveness properties (but not all). Formally, considering a common alphabet Σ, the space of properties considered in [6, 7, 18] is 2^{Σ^∞}. Then a property θ is said to be an infinite renewal properties iff $\forall \sigma \in Exec(\mathscr{P}_\Sigma) \cap \Sigma^\infty, \theta(\sigma) \Leftrightarrow \forall \sigma' \in \Sigma^*, \sigma' \prec \sigma \Rightarrow \exists \sigma'', \sigma' \preceq \sigma'' \prec \sigma \wedge \theta(\sigma'')$. The definition of renewal properties matches as expected our definition of enforceable properties (Definition 10). Hence, according to Theorem 1, in the Safety-Progress classification of *e*-properties, infinite renewal properties are response *e*-properties.

Shallow History Automata and an information-based lattice of enforceable policies [8] Fong studied some restricted forms of runtime execution monitors and their enforcement capabilities. Shallow History Automata (SHA) keep as history a *set* of events the underlying program performed, irrespectively to the order of their arrival. 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 do not distinguish between prefixes of execution sequences obtained from the same events).

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

7.4 Comparing EM with previous runtime enforcement mechanisms (Fig. 10)

It is rather clear that our EMs look like edit-automata. Their computations are produced from a set of operations performed on a memory device. In edit-automata the computation is realized using a set of control states. However the enforcement mechanisms we propose differ in several points. To the best of our knowledge these features are novel regarding enforcement monitoring.

First, let us highlight the genericity of the EM(Ops) introduced in this paper. Security automata of Schneider fall in the scope of our generic enforcement monitors. In fact one can

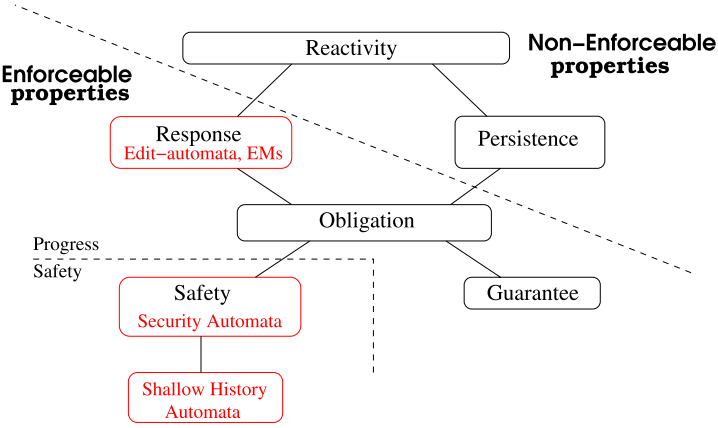


Fig. 10 Enforceable properties and enforcement mechanisms w.r.t. the Safety-Progress classification of properties

notice that by restraining the set *Ops* of enforcement operations to the set $\{halt, dump\}$, it is possible to find an equivalent enforcement monitor to any security automaton. As SHA are a restriction of Schneider's automata, they fall in the scope of our EMs. Edit-automata fall also in the scope of our general enforcement monitors. Indeed one can notice that the primitive sets of edit-automata and EMs are the same.

We propose a translation of a recognizing automaton into an enforcing one. This systematic transformation eases the definition of the enforcement mechanism. Finding and encoding an enforcement mechanism using edit-automata is not an intuitive task. In the case where the property enforced by a security or edit automaton is known, we can synthesize a more concise enforcement monitor in the number of states. Indeed, for a security automaton or an edit-automaton enforcing a property Π, we synthesize an EM using TransObligation or TransResponse applied on $\mathscr{A}_\Pi$ where $\mathscr{A}_\Pi$ is a recognizing automaton for Π.

Compared to edit-automata,[9] our EMs propose a clear distinction between control states (used for property recognition) and the sequence memorization (when the current execution deviates from the property) in the memory device for potential replay (if the execution sequence meets the property again). Hence such a mechanism is easier to implement, since it relies on a finite (and restricted) set of control states. Meanwhile, linking EMs to their implementation is more compatible with formal reasoning. This provides more confidence in the implementation of such mechanisms. Indeed, reducing the size of the trusted computing mechanisms is a persistent challenge in the security domain.

7.5 Synthesis of runtime enforcement mechanisms

There is relatively few research effort dedicated to the synthesis of runtime enforcement mechanisms.

[9]Edit-automata use a potentially infinite number of control states for property recognition and sequence memorization. Thus, even for a simple guarantee property e.g., "eventually b" an edit-automaton needs an infinite number of states to memorize the potential incorrect sequence of events belonging to $\Sigma \setminus \{b\}$. Furthermore, one can notice that the size of an edit-automaton is here almost independent of the alphabet Σ under consideration.

In [20] Martinelli and Matteucci tackle the synthesis of enforcement mechanism as defined by Ligatti. More generally the authors consider Schneider's security automata (truncation automata), insertion, suppression 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 [21] Matteucci extends the approach in the context of real-time systems.

7.6 Implementations

The runtime enforcement monitoring approach was implemented in numerous tools (see [22, 23] for instance). Most of them are based more or less on security automata, whereas Polymer [19] introduces a more expressive framework based on edit-automata. Polymer is a formal-semantics supported language and system which can be used to define, compose and enforce security policies.

7.7 Discussion

In previous work in enforcement monitoring, as in our work, the enforcement mechanisms are restrained to use a peculiar set of enforcement primitives. It may be interesting to notice that, when considering enforcement with the transparency constraint (as in Definition 10) the specialized enforcement mechanism we propose have enough enforcement abilities. Thus, considering more general forms of EMs with alternative set of enforcement operations would not add any further enforcement ability.

Moreover, the previous development, starting from Sect. 4.3, can be conducted considering a set of enforcement operations $\{store, dump\}$. The interest of the operations *halt* and *off* is only practical: the operation *halt* (resp. *off*) is used to bound the size of the memory when it is no longer necessary to memorize further events (resp. to suppress the monitor's performance overhead on the program execution when it is not worth monitoring the input sequence anymore).

8 Conclusion and perspectives

Conclusion In this paper our purpose was to extend previous work on property checking through runtime enforcement in several directions. Firstly, we proposed a generic notion of enforcement monitors based on a memory device, finite sets of control states and enforcement operations. This notion of EM encompasses previous similar ones: security-automata (and consequently shallow-history automata) and edit-automata in a rather obvious way. Moreover, we specified their enforcement abilities w.r.t. the general safety-progress classification of properties. It allowed a fine-grain characterization of the space of enforceable properties. Furthermore, we studied the question of EM composition w.r.t. Boolean operators. Also, we proposed a systematic technique to produce an enforcing monitor from the Streett automaton recognizing a given safety, guarantee, obligation or response property.

Perspectives An important working direction is now to make this runtime enforcement technique better able to cope with practical limitations in order to deal with larger examples. In particular it is likely that not all events produced by an underlying program can be freely

observed, suppressed, or inserted. This leads to well-known notions of *observable* and/or *controllable* events, that have to be taken into account by the enforcement mechanisms. Moreover, it could be also necessary to limit the resources consumed by the monitor by storing in memory only an *abstraction* of the sequence of events observed (i.e., using a *bag* instead of a FIFO queue). From a theoretical point of view, this means to define enforcement up to some *abstraction preserving trace equivalence relations*. We strongly believe that our notion of enforcement monitors (with a generic memory device) is a suitable framework to study and implement these features.

Similarly, it would be interesting to study the notion of enforcement when weakening the transparency constraint. In this case, the more general form of edit-automata and our generic EMs could be used. Their complete enforcement potentials remain to be studied. This perspective would involve defining other relations between the input and the output sequences; and thus defining other enforcement primitives to enforce properties in an automatic fashion. It seems to us that such alternative constraints should be motivated by practical needs.

Another working direction is a prototype tool, currently under development. To validate and extend the previously defined approach we are elaborating a framework implemented as a Java toolbox, using Aspect Oriented Programming [24] as an underlying technique. This framework has been sketched in [25]. Taking, as input, an *e*-property Π specified by a Streett automaton $\mathscr{A}_\Pi$, encoded in XML, it uses a first tool (consisting mainly in implementing the aforementioned transformations) to produce an EM for Π. Then a connected tool, using the generated EM, produces an AspectJ aspect to be weaved with a target Java program. The resulting program then meets property Π, in the sense that this property is actually enforced. We believe that this prototype framework will be a good platform to investigate the impact of the aforementioned practical constraints. Also, we are currently studying alternative rewriting techniques (not based on aspects) to replace the tool for monitor integration in the underlying program (such as BCEL [26] technology, or dynamic binary code insertion [27]). The benefits would be to perform runtime enforcement from binary versions of the target program.

Acknowledgements The authors would like to gratefully thank the anonymous referees for their helpful remarks.

Appendix: Proofs

A.1 Proof of Property 1 (p. 9)

We prove the two facts successively:

1. As Π is a safety *e*-property, then there exists a Streett safety-automaton $\mathscr{A}_\Pi = (Q^{\mathscr{A}_\Pi}, q_{\text{init}}^{\mathscr{A}_\Pi}, \Sigma, \longrightarrow_{\mathscr{A}_\Pi}, \{(\emptyset, P)\})$ specifying it. Let σ be a sequence belonging to Π, then σ is accepted by $\mathscr{A}_\Pi$. If σ is finite, it means that the last state visited during the run of σ on $\mathscr{A}_\Pi$ is in P (Definition 3). Thus, each prefix of σ has its run ending in P since there is no transition from $\overline{P}$ to P. According to the acceptance criterion of Streett automata for finite sequences, all prefixes are accepted by $\mathscr{A}_\Pi$ and thus belong to Π. If σ is an infinite sequence, it means that all states visited infinitely often during the run of σ on $\mathscr{A}_\Pi$ are in P ($\mathit{inf}(\sigma, \mathscr{A}_\Pi) \subseteq P$). Since there is no transition from $\overline{P}$ to P, no prefix of σ visits a state in $\overline{P}$; i.e., all prefixes of σ belong to Π.
2. Similarly, when considering a guarantee property and its specifying automaton, all accepted sequences (belonging to the property) have their run ending in a R-state. Finite

continuations of accepted sequences still have their run ending in a R-state since there is no transition from R to $\overline{R}$. Infinite continuations of an accepted sequence visit at least one state in R infinitely often: the R-state in which the run of the accepted sequence ends in.

A.2 Proof of Property 3 (p. 16)

This proof is done by induction on the length of the input sequence σ.

Induction basis $|\sigma| = 1$; we have $\sigma = a$ with $a \in \Sigma$. Using the definition of evolution of configurations (Definition 8), we have $\exists q \in Q^{\mathscr{A}_\downarrow}, (q_{\text{init}}^{\mathscr{A}_\downarrow}, \sigma \cdot \sigma', \varepsilon) \overset{o}{\Longrightarrow}_{\mathscr{A}_\downarrow} (q, \sigma', m)$ with $\alpha(\sigma, \varepsilon) = (o, m)$ and $q_{\text{init}}^{\mathscr{A}_\downarrow} \xrightarrow{\sigma/\alpha} q$.

- If $\alpha = halt$, then $o = \varepsilon, m = \varepsilon$ and $q \in Halt^{\mathscr{A}_\downarrow}$. We have $o \prec \sigma$.
- Else, if $\alpha = store$, then $o = \varepsilon, m = \sigma$. We have $\sigma = o \cdot m$.
- Else ($\alpha = dump$ or $\alpha = off$), $o = a, m = \varepsilon$ and $\sigma = o \cdot m$.

Induction step Suppose that the property is verified for every execution sequence of length n and consider an execution sequence $\sigma \cdot a$ of length $n + 1$, where $a \in \Sigma$. By reading σ, $\mathscr{A}_\downarrow$ enters a state $q \in Q^{\mathscr{A}_\downarrow}$, produces an output o, and has m in its memory: $\exists q \in Q^{\mathscr{A}_\downarrow}, (q_{\text{init}}^{\mathscr{A}_\downarrow}, \sigma \cdot a \cdot \sigma', \varepsilon) \overset{o}{\Longrightarrow}_{\mathscr{A}_\downarrow} (q, a \cdot \sigma', m)$. Moreover, the induction hypothesis gives: $(q \in Halt^{\mathscr{A}_\downarrow} \wedge o \preceq \sigma) \vee (q \notin Halt^{\mathscr{A}_\downarrow} \wedge \sigma = o \cdot m)$. As $\mathscr{A}_\downarrow$ is complete w.r.t. $Q^{\mathscr{A}_\downarrow} \times \Sigma$ (definition of EMs), $\exists \alpha \in Ops, \exists q' \in Q^{\mathscr{A}_\downarrow}, q \xrightarrow{a/\alpha}_{\mathscr{A}_\downarrow} q'$. So, $\exists o', m' \in \Sigma^*, (q, a \cdot \sigma', m) \overset{o'}{\hookrightarrow}_{\mathscr{A}_\downarrow} (q', \sigma', m')$ with $\alpha(a, m) = (o', m')$. Which results in $(q_{\text{init}}^{\mathscr{A}_\downarrow}, \sigma \cdot a \cdot \sigma', \varepsilon) \overset{o \cdot o'}{\Longrightarrow}_{\mathscr{A}_\downarrow} (q', \sigma', m')$ and $(q \in Halt^{\mathscr{A}_\downarrow} \wedge o \preceq \sigma) \vee (q \notin Halt^{\mathscr{A}_\downarrow} \wedge \sigma = o \cdot m)$. We want to show that $(q' \in Halt^{\mathscr{A}_\downarrow} \wedge o \cdot o' \preceq \sigma \cdot a) \vee (q' \notin Halt^{\mathscr{A}_\downarrow} \wedge \sigma \cdot a = o \cdot o' \cdot m')$. Let us treat three cases for the enforcement operation α.

- Case $\alpha = halt$. We have $\alpha(a, m) = (\varepsilon, m)$. So $o' = \varepsilon$ and $m = m'$. And we also have, according to the definition of EMs (Definition 6), $q' \in Halt^{\mathscr{A}_\downarrow}$. Then, we apply the induction hypothesis with σ, and depending on the membership of q in $Halt^{\mathscr{A}_\downarrow}$. If $q \in Halt^{\mathscr{A}_\downarrow}$, $o \preceq \sigma \Rightarrow o \cdot \varepsilon \preceq \sigma \cdot a$. Else ($q \notin Halt^{\mathscr{A}_\downarrow}$), we have $o \cdot \varepsilon \preceq \sigma \cdot a$.
- Case $\alpha = store$. We have $q \notin Halt^{\mathscr{A}_\downarrow}$, and $\alpha(a, m) = (\varepsilon, m \cdot a)$, so $o' = \varepsilon$ and $m' = m \cdot a$. By induction hypothesis, $q' \notin Halt^{\mathscr{A}_\downarrow}$ (Definition 6) and $\sigma = o \cdot m$. Hence, we have $\sigma \cdot a = o \cdot m \cdot a = o \cdot o' \cdot m'$.
- Case $\alpha \in \{dump, off\}$. We have $q \notin Halt^{\mathscr{A}_\downarrow}$, and $\alpha(a, m) = (m \cdot a, \varepsilon)$. Then $o' = m \cdot a$ and $m' = \varepsilon$. By induction hypothesis, we have necessarily $q' \notin Halt^{\mathscr{A}_\downarrow}$ (Definition 6), and $\sigma = o \cdot m$. Hence, we have $\sigma \cdot a = o \cdot m \cdot a = o \cdot o' \cdot m'$.

A.3 Correctness of the Union and Intersection operations (Theorem 2, p. 18)

Note first that these constructions effectively build EMs. We only prove the Union operator. For $i \in \{1, 2\}$, $Enf(\mathscr{A}_{\downarrow \Pi_i}, \Pi_i, \mathscr{P}_\Sigma)$, i.e., $\forall \sigma \in Exec(\mathscr{P}_\Sigma), \exists o_i \in \Sigma^*$:

$$\sigma \Downarrow_{\mathscr{A}_{\downarrow \Pi_i}} o_i \tag{9}$$

$$\Pi_i(\sigma) \Rightarrow \sigma = o_i \tag{10}$$

$$\neg \Pi_i(\sigma) \wedge Pref_\prec(\phi_i, \sigma) = \emptyset \Rightarrow o_i = \varepsilon \tag{11}$$

$$\neg \Pi_i(\sigma) \wedge Pref_\prec(\phi_i, \sigma) \neq \emptyset \Rightarrow o_i = Max(Pref_\prec(\phi_i, \sigma)) \tag{12}$$

Let us note $\mathscr{A}_{\sqcup} = \text{Union}(\mathscr{A}_{\downarrow\Pi_1}, \mathscr{A}_{\downarrow\Pi_2}) = (Q, q_{\text{init}}, \longrightarrow, Ops)$, $\Pi = \Pi_1 \vee \Pi_2$, and $\Rightarrow$ the multistep derivation relation defined over configurations of $\mathscr{A}_{\sqcup}$ and $\longrightarrow$. We have to show $Enf(\mathscr{A}_{\sqcup}, \Pi, \mathscr{P}_{\Sigma})$, that is, given $\sigma \in Exec(\mathscr{P}_{\Sigma})$, $\exists o \in \Sigma^*$ s.t.,

$$\sigma \Downarrow_{\mathscr{A}_{\sqcup}} o \tag{13}$$

$$\Pi(\sigma) \Rightarrow \sigma = o \tag{14}$$

$$\neg\Pi(\sigma) \wedge Pref_{\prec}(\phi, \sigma) = \emptyset \Rightarrow o = \varepsilon \tag{15}$$

$$\neg\Pi(\sigma) \wedge Pref_{\prec}(\phi, \sigma) \neq \emptyset \Rightarrow o = Max(Pref_{\prec}(\phi, \sigma)) \tag{16}$$

We first consider $\sigma \in \Sigma^*$, and use induction on $|\sigma|$.

Induction basis For the induction basis, we have $|\sigma| = 0$ and $\sigma = \varepsilon$. Then we have (13) and (14) as $\varepsilon \Downarrow_{\mathscr{A}_{\downarrow\Pi}} \varepsilon$. Moreover, $Pref_{\prec}(\phi, \varepsilon) = \emptyset$ gives us (15).

Inductive step Let $n \in \mathbb{N}$ and suppose that for all sequences σ s.t. $|\sigma| = n$, there exists an output o of $\mathscr{A}_{\sqcup}$ s.t. (13), (14), (15), and (16).

As $\sigma \Downarrow_{\mathscr{A}_{\sqcup}} o$ (induction hypothesis), there exists a configuration $(q, \varepsilon, m) \in Q \times \Sigma^* \times \Sigma^*$ s.t. $(q_{\text{init}}, \sigma, \varepsilon) \overset{o}{\Rightarrow} (q, \varepsilon, m)$, which implies that $(q_{\text{init}}, \sigma \cdot a, \varepsilon) \overset{o}{\Rightarrow} (q, a, m)$. That is, after reading σ, the EM $\mathscr{A}_{\sqcup}$ is in a state q with a as input, and m as memory content. Then from the configuration (q, a, m), it evolves towards a configuration (q', ε, m'), that is $(q, a, m) \overset{o'}{\hookrightarrow} (q', \varepsilon, m')$ with $\alpha(a, m) = (o', m')$, $\alpha \in Ops$. By reading $\sigma \cdot a$, $\mathscr{A}_{\sqcup}$ produces the output $o \cdot o'$. Also, reading of $\sigma \cdot a$ on $\mathscr{A}_{\downarrow\Pi_i}$, $i \in \{1, 2\}$, induces the following evolution of configurations:

$$(q_{\text{init}}, \sigma \cdot a, \varepsilon) \overset{o_i}{\Rightarrow} (q_i, a, m_i) \overset{o'_i}{\hookrightarrow} (q'_i, \varepsilon, m'_i),$$

with $\alpha_i(a, m_i) = (o'_i, m'_i)$; $q_i, q'_i \in Q^{\mathscr{A}_{\downarrow\Pi_i}}$; $m_i, m'_i, o_i, o'_i \in \Sigma^*$.
There are two cases depending on $\phi(\sigma \cdot a)$.

- $\phi(\sigma \cdot a)$. In this case, either $\phi_1(\sigma \cdot a)$ or $\phi_2(\sigma \cdot a)$. Let us consider $\phi_1(\sigma \cdot a)$, $\phi_2(\sigma \cdot a)$ is similar. As $Enf(\Pi_1, \mathscr{A}_{\downarrow\Pi_1}, \mathscr{P}_{\Sigma})$, $\exists o_1 \in \Sigma^*, \sigma \cdot a \Downarrow_{\mathscr{A}_{\downarrow\Pi_1}} o_1$. Moreover, $\phi_1(\sigma \cdot a)$ implies that $o_1 = \sigma \cdot a$. Inevitably the last enforcement operation α_1 of $\mathscr{A}_{\downarrow\Pi_1}$ is *dump* or *off* (Property 4). Then $\alpha = \bigsqcup(\{\alpha_1, \alpha_2\}) = dump \vee \alpha = off$. According to the definition of enforcement operation and Property 3, $\sigma \cdot a \Downarrow_{\mathscr{A}_{\sqcup}} \sigma \cdot a$, i.e., (13) and (14).
- $\neg\phi(\sigma \cdot a)$. Then $\neg\phi_1(\sigma \cdot a) \wedge \neg\phi_2(\sigma \cdot a)$. Using the definition of enforcement, we have four cases depending on whether $Pref_{\prec}(\phi_i, \sigma \cdot a) = \emptyset$ or not, $i \in \{1, 2\}$.
  - The first case is $Pref_{\prec}(\phi_i, \sigma \cdot a) \neq \emptyset, i \in \{1, 2\}$. For $i \in \{1, 2\}$, as $Enf(\Pi_i, \mathscr{A}_{\downarrow\Pi_i}, \mathscr{P}_{\Sigma})$, $\neg\phi_i(\sigma \cdot a)$ gives $\exists o_i \in \Sigma^*, o_i = Max(Pref_{\prec}(\phi_i, \sigma \cdot a))$. Now, we have either $o_1 \prec o_2$, $o_2 \prec o_1$ or $o_1 = o_2$.
    - $o_1 \prec o_2$ ($o_2 \prec o_1$ is symmetrical). We have $\forall o'_1 \in \Sigma^*, o_1 \prec o'_1 \preceq \sigma \cdot a \Rightarrow \neg\phi_1(o'_1)$, and $\forall o'_2 \in \Sigma^*, o_2 \prec o'_2 \preceq \sigma \cdot a \Rightarrow \neg\phi_2(o'_2)$. Then $o_1 \prec o_2$ implies that $o_2 = Max(Pref_{\prec}(\phi, \sigma \cdot a))$. We have to show that $\sigma \cdot a \Downarrow_{\mathscr{A}_{\sqcup}} o_2$. Let us examine the sequence of enforcement operations performed by $\mathscr{A}_{\sqcup}$. We have $o_2 \Downarrow_{\mathscr{A}_{\sqcup}} o_2$, as the last enforcement operation performed while reading $o_2 \preceq \sigma \cdot a$ is a *dump* ($\mathscr{A}_{\sqcup}$ is obtained by taking the upperbound of enforcement operations).
    - If $o_1 = o_2$, $o_1 = o_2 = Max(Pref_{\prec}(\phi, \sigma \cdot a))$. The previous reasoning holds.
  - The second case is $Pref_{\prec}(\phi_i, \sigma \cdot a) = \emptyset, i \in \{1, 2\}$. For $i \in \{1, 2\}$, as $Enf(\Pi_i, \mathscr{A}_{\downarrow\Pi_i}, \mathscr{P}_{\Sigma})$, $\neg\phi_i(\sigma \cdot a)$ gives $\sigma \cdot a \Downarrow_{\mathscr{A}_{\downarrow\Pi_i}} \varepsilon, i \in \{1, 2\}$.

- The third case is $Pref_{\prec}(\phi_1, \sigma \cdot a) = \emptyset \vee Pref_{\prec}(\phi_2, \sigma \cdot a) = \emptyset$. Since $Enf(\Pi_i, \mathscr{A}_{\downarrow \Pi_i}, \mathscr{P}_{\Sigma})$, $i \in \{1, 2\}$, it gives us two sequences $o_i \in \Sigma^*$, s.t. $o_i = Max(Pref_{\prec}(\phi_i, \sigma \cdot a))$, $i \in \{1, 2\}$ that can be compared similarly to the first case.

For infinite sequences, the reasoning is similar to the one for finite sequences. It is done on the shape of the sequence of enforcement operations and by distinguishing according to whether $\varphi(\sigma)$ or not. Indeed, depending on $\varphi(\sigma)$, and using the fact that $\mathscr{A}_{\downarrow \Pi_i}$ enforces $\mathscr{A}_{\downarrow \Pi_i}$, $i \in \{1, 2\}$, we associate the possible sequences of enforcement operations on $\mathscr{A}_{\downarrow \Pi_i}$ to the sequence of enforcement operations on $\mathscr{A}_{\sqcup}$.

Note that we have indeed $Halt^{\mathscr{A}_{\sqcup}} = Halt^{\mathscr{A}_{\downarrow \Pi_1}} \times Halt^{\mathscr{A}_{\downarrow \Pi_2}}$. Using the definition of $\mathscr{A}_{\downarrow \sqcup}$, we have:

$$
\begin{aligned}
Halt^{\mathscr{A}_{\sqcup}} &= \{q' \in Q^{\mathscr{A}_{\sqcup}} \mid \exists a \in \Sigma, \exists q \in Q^{\mathscr{A}_{\sqcup}}, q \xrightarrow{a/halt}_{\mathscr{A}_{\downarrow \sqcup}} q'\} \\
&= \{(q_1', q_2') \in Q^{\mathscr{A}_{\downarrow \Pi_1}} \times Q^{\mathscr{A}_{\downarrow \Pi_2}} \mid \\
&\quad \exists a \in \Sigma, \exists (q_1, q_2) \in Q^{\mathscr{A}_{\downarrow \Pi_1}} \times Q^{\mathscr{A}_{\downarrow \Pi_2}}, (q_1, q_2) \xrightarrow{a/halt}_{\mathscr{A}_{\sqcup}} (q_1', q_2')\} \\
&= \{(q_1', q_2') \in Q^{\mathscr{A}_{\downarrow \Pi_1}} \times Q^{\mathscr{A}_{\downarrow \Pi_2}} \mid \exists a \in \Sigma, \\
&\quad \exists q_1 \in Q^{\mathscr{A}_{\downarrow \Pi_1}}, \exists q_2 \in Q^{\mathscr{A}_{\downarrow \Pi_2}}, q_1 \xrightarrow{a/\alpha_1}_{\mathscr{A}_{\downarrow \Pi_1}} q_1' \wedge q_2 \xrightarrow{a/\alpha_2}_{\mathscr{A}_{\downarrow \Pi_2}} q_2' \wedge \alpha_1 \sqcup \alpha_2 = halt\} \\
&= \left\{(q_1', q_2') \in Q^{\mathscr{A}_{\downarrow \Pi_1}} \times Q^{\mathscr{A}_{\downarrow \Pi_2}} \mid \exists a \in \Sigma, \right. \\
&\quad \left. \left(\exists q_1 \in Q^{\mathscr{A}_{\downarrow \Pi_1}}, q_1 \xrightarrow{a/halt}_{\mathscr{A}_{\downarrow \Pi_1}} q_1'\right) \wedge \left(\exists q_2 \in Q^{\mathscr{A}_{\downarrow \Pi_2}}, q_2 \xrightarrow{a/halt}_{\mathscr{A}_{\downarrow \Pi_2}} q_2'\right)\right\} \\
&= \{q_1' \in Q^{\mathscr{A}_{\downarrow \Pi_1}} \mid \exists a \in \Sigma, \exists q_1 \in Q^{\mathscr{A}_{\downarrow \Pi_1}}, q_1 \xrightarrow{a/halt}_{\mathscr{A}_{\downarrow \Pi_1}} q_1'\} \\
&\quad \times \{q_2' \in Q^{\mathscr{A}_{\downarrow \Pi_2}} \mid \exists a \in \Sigma, \exists q_2 \in Q^{\mathscr{A}_{\downarrow \Pi_2}}, q_2 \xrightarrow{a/halt}_{\mathscr{A}_{\downarrow \Pi_2}} q_2'\} \\
&= Halt^{\mathscr{A}_{\downarrow \Pi_1}} \times Halt^{\mathscr{A}_{\downarrow \Pi_2}}
\end{aligned}
$$

Note also that the states in $Halt^{\mathscr{A}_{\sqcup}}$ verify the constraint expressed in Sect. 4.3. That is $\forall q \in Halt^{\mathscr{A}_{\sqcup}}, \forall a \in \Sigma, \forall \alpha \in Ops, \forall q' \in Q^{\mathscr{A}_{\sqcup}}, q \xrightarrow{a/\alpha}_{\mathscr{A}_{\sqcup}} q' \Rightarrow \alpha = halt$. It is a direct consequence of the fact that $Halt^{\mathscr{A}_{\sqcup}} = Halt^{\mathscr{A}_{\downarrow \Pi_1}} \times Halt^{\mathscr{A}_{\downarrow \Pi_2}}$ and the fact that a *halt* operation is performed on $\mathscr{A}_{\sqcup}$ iff the operation *halt* is performed on the two corresponding transitions in $\mathscr{A}_{\downarrow \Pi_1}$ and $\mathscr{A}_{\downarrow \Pi_2}$.

Similarly, we can show that first $Off^{\mathscr{A}_{\sqcup}} = Off^{\mathscr{A}_{\downarrow \Pi_1}} \times Q^{\mathscr{A}_{\downarrow \Pi_2}} \cup Q^{\mathscr{A}_{\downarrow \Pi_1}} \times Off^{\mathscr{A}_{\downarrow \Pi_2}}$ and second that states in $Off^{\mathscr{A}_{\sqcup}}$ verify the constraint of Definition 13. Therefore $\mathscr{A}_{\sqcup}$ is indeed an EM.

The proof for the intersection operator is conducted similarly.

A.4 Correctness of the Negation operation (Theorem 3, p. 19)

Let the e-property Π be (ϕ, φ), with $\phi \subseteq \Sigma^*$ and $\varphi \subseteq \Sigma^\omega$. Let us note $\overline{\mathscr{A}_{\downarrow \Pi}} = \text{Negation}(\mathscr{A}_{\downarrow \Pi})$, and $\Rightarrow_{\overline{\mathscr{A}_{\downarrow \Pi}}}$ the multistep derivation relation defined over configurations of $\overline{\mathscr{A}_{\downarrow \Pi}}$ and $\longrightarrow_{\overline{\mathscr{A}_{\downarrow \Pi}}}$. Also, since $Q^{\mathscr{A}_{\downarrow \Pi}} = Q^{\overline{\mathscr{A}_{\downarrow \Pi}}}$, we will use Q to denote the set of states of both EMs. Similarly q_{init} denotes the starting states of both EMs. We have to show $Enf(\overline{\mathscr{A}_{\downarrow \Pi}}, \overline{\Pi}, \mathscr{P}_{\Sigma})$, that is, $\forall \sigma \in Exec(\mathscr{P}_{\Sigma}), \exists o \in \Sigma^\infty$ s.t.

$$\sigma \Downarrow_{\overline{\mathscr{A}_{\downarrow\Pi}}} o \tag{17}$$

$$\overline{\Pi}(\sigma) \Rightarrow \sigma = o \tag{18}$$

$$\neg\overline{\Pi}(\sigma) \wedge Pref_{\prec}(\overline{\phi}, \sigma) = \emptyset \Rightarrow o = \varepsilon \tag{19}$$

$$\neg\overline{\Pi}(\sigma) \wedge Pref_{\prec}(\overline{\phi}, \sigma) \neq \emptyset \Rightarrow o = Max(Pref_{\prec}(\overline{\phi}, \sigma)) \tag{20}$$

The proof is in two steps: the first one is for finite sequences, the second one for infinite sequences.

A.4.1 Finite sequences

The proof is done by induction on $|\sigma|$.

Induction basis $|\sigma| = 0$; $\sigma = \varepsilon$, so we have (17) and (18) as $\varepsilon \Downarrow_{\mathscr{A}_{\downarrow\Pi}} \varepsilon$. Moreover, $Pref_{\prec}(\phi, \varepsilon) = \emptyset$, which gives (19).

Inductive step Let $n \in \mathbb{N}$ and suppose that for all sequences σ s.t. $|\sigma| = n$, there exists an output $o \in \Sigma^*$ s.t. the constraints (17), (18), (19) and (20) hold. Considering $a \in \Sigma$ and a sequence $\sigma \cdot a$ s.t. $|\sigma \cdot a| = n + 1$, we study the effect of the submission in input of the last event a. We will prove that there exists a new output s.t. the same constraints hold.

As $\sigma \Downarrow_{\overline{\mathscr{A}_{\downarrow\Pi}}} o$ (induction hypothesis), there exists a configuration $(q, \varepsilon, m) \in Q \times \Sigma^* \times \Sigma^*$ such that $(q_{\text{init}}, \sigma, \varepsilon) \overset{o}{\Longrightarrow}_{\overline{\mathscr{A}_{\downarrow\Pi}}} (q, \varepsilon, m)$, which implies that $(q_{\text{init}}, \sigma \cdot a, \varepsilon) \overset{o}{\Longrightarrow}_{\overline{\mathscr{A}_{\downarrow\Pi}}} (q, a, m)$. That is, after reading σ, $\overline{\mathscr{A}_{\downarrow\Pi}}$ is in a state q with a in input, and m as memory content. Then from the configuration (q, a, m), it evolves towards a configuration (q', ε, m'), that is $(q, a, m) \overset{o'}{\hookrightarrow}_{\overline{\mathscr{A}_{\downarrow\Pi}}} (q', \varepsilon, m')$ with $\alpha(a, m) = (o', m')$, $\alpha \in Ops$. The reading of $\sigma \cdot a$ on $\mathscr{A}_{\downarrow\Pi}$ induces the evolution of configurations:

$$(q_{\text{init}}, \sigma \cdot a, \varepsilon) \overset{o}{\Longrightarrow}_{\overline{\mathscr{A}_{\downarrow\Pi}}} (q, a, m) \overset{o'}{\hookrightarrow}_{\overline{\mathscr{A}_{\downarrow\Pi}}} (q', \varepsilon, m')$$

$$(q_{\text{init}}, \sigma \cdot a, \varepsilon) \overset{p}{\Longrightarrow}_{\mathscr{A}_{\downarrow\Pi}} (q, a, n) \overset{p'}{\hookrightarrow}_{\mathscr{A}_{\downarrow\Pi}} (q', \varepsilon, n'),$$

with:

- $q \xrightarrow{a/\alpha}_{\overline{\mathscr{A}_{\downarrow\Pi}}} q', \alpha(a, m) = (o', m'), \alpha \in Ops; q, q' \in Q; m, m', o, o' \in \Sigma^*$;
- $q \xrightarrow{a/\alpha'}_{\mathscr{A}_{\downarrow\Pi}} q', \alpha'(a, n) = (p', n'), \alpha' \in Ops; q, q' \in Q; n, n', p, p' \in \Sigma^*$.

There are two cases depending on $\phi(\sigma \cdot a)$:

- $\phi(\sigma \cdot a)$. As $Enf(\Pi, \mathscr{A}_{\downarrow\Pi}, \mathscr{P}_\Sigma)$, $\mathscr{A}_{\downarrow\Pi}$ produces $\sigma \cdot a$, i.e., $\sigma \cdot a \Downarrow_{\mathscr{A}_{\downarrow\Pi}} \sigma \cdot a$. Necessarily, $\alpha' \in \{dump, off\}$. It corresponds to an operation $\alpha \in \{store, halt\}$ on $\overline{\mathscr{A}_{\downarrow\Pi}}$. Now we distinguish according to whether $\phi(\sigma)$ or not.
  - If $\phi(\sigma)$, using the induction hypothesis ($|\sigma| = n$), we have either $o = \varepsilon$ (when $Pref_{\prec}(\overline{\phi}, \sigma) = \emptyset$) or $o = Max(Pref_{\prec}(\overline{\phi}, \sigma))$ (when $Pref_{\prec}(\overline{\phi}, \sigma) \neq \emptyset$).
    - If $Pref_{\prec}(\overline{\phi}, \sigma) = \emptyset$, then we also have $Pref_{\prec}(\overline{\phi}, \sigma \cdot a) = \emptyset$. The output of $\overline{\mathscr{A}_{\downarrow\Pi}}$ is still ε, i.e., $o \cdot o' = \varepsilon$. We have (19).
    - If $Pref_{\prec}(\overline{\phi}, \sigma) \neq \emptyset$, using the induction hypothesis, $o = Max(Pref_{\prec}(\overline{\phi}, \sigma))$. Yet $\phi(\sigma \cdot a)$, it implies that $o = Max(Pref_{\prec}(\overline{\phi}, \sigma \cdot a))$. We have (20).

- If $\neg\phi(\sigma)$, i.e., $\overline{\phi}(\sigma)$, using the induction hypothesis, we have $\sigma \Downarrow_{\overline{\mathscr{A}_{\downarrow\Pi}}} o$ with $\sigma = o$. Then $\sigma = Max(Pref_{\prec}(\overline{\phi}, \sigma))$ since $\overline{\phi}(\sigma)$. We also obtain (20).

– $\overline{\phi}(\sigma \cdot a)$. Then, we have $Max(Pref_{\prec}(\phi, \sigma \cdot a)) \prec \sigma \cdot a$. It follows that $\alpha' \in \{store, halt\}$. As a consequence $\alpha \in \{dump, off\}$ and $\sigma \cdot a \Downarrow_{\overline{\mathscr{A}_{\downarrow\Pi}}} \sigma \cdot a$. We have (17) and (18).

A.4.2 Infinite sequences

We distinguish according to the class of Π. Let us consider $\sigma \in \Sigma^{\omega}$.

– *Π is a safety e-property.* We have two cases, depending on whether $\varphi(\sigma)$ or not.
- $\varphi(\sigma)$. As $Enf(\Pi, \mathscr{A}_{\downarrow\Pi}, \mathscr{P}_{\Sigma})$, we have $\sigma \Downarrow_{\mathscr{A}_{\downarrow\Pi}} \sigma$. Moreover as Π is a safety e-property, all prefixes of σ satisfy ϕ (Property 1), that is $\forall\sigma' \in \Sigma^*, \sigma' \prec \sigma \Rightarrow \phi(\sigma')$, and consequently $\sigma' \Downarrow_{\mathscr{A}_{\downarrow\Pi}} \sigma'$. It follows (Property 4) that the sequence of enforcement operations on $\mathscr{A}_{\downarrow\Pi}$ belongs to $(dump)^{\omega} + dump^* \cdot off^{\omega}$. Then using the definition of Negation, we find that the sequence of enforcement operations on $\overline{\mathscr{A}_{\downarrow\Pi}}$ belongs to $store^* \cdot halt^{\omega} + store^{\omega}$. It follows that $\sigma \Downarrow_{\overline{\mathscr{A}_{\downarrow\Pi}}} \varepsilon$, i.e., (17). As $Pref_{\prec}(\overline{\phi}, \sigma) = \emptyset$, we obtain (19).
- $\neg\varphi(\sigma)$. As $Enf(\Pi, \mathscr{A}_{\downarrow\Pi}, \mathscr{P}_{\Sigma})$, we have two cases: either $Pref_{\prec}(\phi, \sigma) = \emptyset \wedge o = \varepsilon$ or $\exists o \in \Sigma^*, o = Max(Pref_{\prec}(\phi, \sigma))$.
  - Let us deal first with the case $Pref_{\prec}(\phi, \sigma) = \emptyset$. We have $\forall\sigma' \in \Sigma^*, \sigma' \prec \sigma \Rightarrow \neg\phi(\sigma')$. It follows that the sequence of enforcement operations on $\mathscr{A}_{\downarrow\Pi}$ belongs to $store^* \cdot halt^{\omega} + store^{\omega}$. Using the definition of Negation, the sequence of enforcement operations of $\overline{\mathscr{A}_{\downarrow\Pi}}$ is off^{ω}. It follows that $\sigma \Downarrow_{\overline{\mathscr{A}_{\downarrow\Pi}}} \sigma$. We obtain (17).
  - Let us deal now with the case $Pref_{\prec}(\phi, \sigma) \neq \emptyset$. Let $n = |o|$. As Π is a safety e-property, we have $\forall 1 \leq i \leq n, \phi(\sigma_{\ldots i-1}) \wedge \forall i > n, \neg\phi(\sigma_{\ldots i})$. Then using Property 4, we can find the sequence of enforcement operations performed by $\mathscr{A}_{\downarrow\Pi}$: $(dump)^n \cdot halt^{\omega}$. On $\overline{\mathscr{A}_{\downarrow\Pi}}$, using the definition of the transformation Negation, the sequence of enforcement operations becomes $store^n \cdot off^{\omega}$. It follows that $\sigma \Downarrow_{\overline{\mathscr{A}_{\downarrow\Pi}}} \sigma$ (17). Then, $\overline{\varphi(\sigma)}$ and $\sigma = \sigma$ ensure (18).

– *Π is a guarantee e-property.* We have two cases, depending on whether $\varphi(\sigma)$ or not.
- $\varphi(\sigma)$. As $Enf(\Pi, \mathscr{A}_{\downarrow\Pi}, \mathscr{P}_{\Sigma})$, we have $\sigma \Downarrow_{\mathscr{A}_{\downarrow\Pi}} \sigma$. Moreover as Π is a guarantee e-property, there exists a prefix σ' of σ s.t. $\forall\sigma'' \in \Sigma^*, \sigma' \preceq \sigma'' \Rightarrow \phi(\sigma'') \wedge \forall\sigma'' \in \Sigma^*, \sigma'' \prec \sigma' \Rightarrow \neg\phi(\sigma'')$. Let us note $n = |\sigma'|$. Consequently, as Π is enforced by $\mathscr{A}_{\downarrow\Pi}$, we have $\forall\sigma'' \in \Sigma^*, \sigma' \preceq \sigma'' \Rightarrow \sigma'' \Downarrow_{\mathscr{A}_{\downarrow\Pi}} \sigma'' \wedge \forall\sigma'' \in \Sigma^*, \sigma'' \prec \sigma' \Rightarrow \sigma'' \Downarrow_{\mathscr{A}_{\downarrow\Pi}} \varepsilon$. It follows that the sequence of enforcement operations on $\mathscr{A}_{\downarrow\Pi}$ is $store^{n-1} \cdot off^{\omega}$. Note that for guarantee e-properties, the *dump* operation is never used: once a finite sequence satisfies a guarantee e-property, all its continuations also do. Then, using the definition of the transformation Negation, we find that the sequence of enforcement operations on $\overline{\mathscr{A}_{\downarrow\Pi}}$ is $dump^{n-1} \cdot halt^{\omega}$. It follows that $\sigma \Downarrow_{\overline{\mathscr{A}_{\downarrow\Pi}}} \sigma'$ (17). Moreover as we have seen that $\phi(\sigma')$, we have (20).
- $\neg\varphi(\sigma)$. Π is a guarantee e-property, $\neg\varphi(\sigma)$ implies that there is no prefix of σ satisfying ϕ. As $Enf(\Pi, \mathscr{A}_{\downarrow\Pi}, \mathscr{P}_{\Sigma})$, we have $\forall\sigma' \prec \sigma, \sigma \Downarrow_{\mathscr{A}_{\downarrow\Pi}} \varepsilon$. The sequence of enforcement operations performed by $\mathscr{A}_{\downarrow\Pi}$ belongs to $store^* \cdot halt^{\omega}$. Using the definition of the Negation transformation, the sequence of enforcement operations on $\overline{\mathscr{A}_{\downarrow\Pi}}$ belongs to $dump^* \cdot off^{\omega}$. It follows that $\sigma \Downarrow_{\overline{\mathscr{A}_{\downarrow\Pi}}} \sigma$. We have (17) and (18).

Finally, due to the definition of $\longrightarrow_{\overline{\mathscr{A}_{\downarrow\Pi}}}$ we have easily $Halt^{\overline{\mathscr{A}_{\downarrow\Pi}}} = Off^{\mathscr{A}_{\downarrow\Pi}}$ and $Off^{\overline{\mathscr{A}_{\downarrow\Pi}}} = Halt^{\mathscr{A}_{\downarrow\Pi}}$. Moreover, the constraints for $Halt^{\overline{\mathscr{A}_{\downarrow\Pi}}}$ and $Off^{\overline{\mathscr{A}_{\downarrow\Pi}}}$ states are respected since they are respected for states in $Off^{\mathscr{A}_{\downarrow\Pi}}$ and $Halt^{\mathscr{A}_{\downarrow\Pi}}$, and $\overline{\mathscr{A}_{\downarrow\Pi}}$is an EM.

A.5 Correctness of the TransResponse transformation (Theorem 4, p. 22)

Intuitively, the proof can be understood as follows. When a sequence satisfies a response property, there exists an alternation in the satisfaction of the prefixes of this sequence. When a sequence does not satisfy the property, there exists an index from which the run of the recognized sequence is composed of "bad states" forever.

We note $\mathscr{A}_{\Pi} = (Q^{\mathscr{A}_{\Pi}}, q_{\text{init}}^{\mathscr{A}_{\Pi}}, \Sigma, \longrightarrow_{\mathscr{A}_{\Pi}}, \{(R, \emptyset)\})$. Let us consider an execution sequence of the program $\sigma \in Exec(\mathscr{P}_{\Sigma})$. We study the effect of the submission of σ to $\mathscr{A}_{\downarrow\Pi}$. We will associate the execution of σ on $\mathscr{A}_{\Pi}$ to the execution of σ on $\mathscr{A}_{\downarrow\Pi}$. The execution of σ on $\mathscr{A}_{\Pi}$ produces a trace $(q_0, \sigma_0, q_1) \cdot (q_1, \sigma_1, q_2) \cdots (q_i, \sigma_i, q_{i+1}) \cdots$ which corresponds to a trace $(q_0, \sigma_0/\alpha_0, q_1) \cdots (q_i, \sigma_i/\alpha_i, q_{i+1}) \cdots$ on $\mathscr{A}_{\downarrow\Pi}$ with $q_0 = q_{\text{init}}^{\mathscr{A}_{\downarrow\Pi}}$. We distinguish depending on whether the sequence σ satisfies Π or not.

- The first case is $\Pi(\sigma)$. We know that the automaton $\mathscr{A}_{\Pi}$ accepts σ, let us distinguish whether σ is finite or not.
  - * If $\sigma \in \Sigma^*$, then $\phi(\sigma)$. Let $n = |\sigma|$. As σ is accepted by $\mathscr{A}_{\Pi}$, and according to the acceptance criterion (Definition 3), there exists a state $q \in R$ reachable from $q_{\text{init}}^{\mathscr{A}_{\Pi}}$ s.t. the run of $\mathscr{A}_{\Pi}$ on σ ends in a R-state (we have $P = \emptyset$ since $\mathscr{A}_{\Pi}$ is a response automaton).

    If $\sigma = \varepsilon$, then we have (5) as $\varepsilon \Downarrow_{\mathscr{A}_{\downarrow\Pi}} \varepsilon$. Moreover, $Pref_{\prec}(\phi, \varepsilon) = \emptyset$, which gives (7).

    If $(\sigma \neq \varepsilon)$, according to the constraints of the transition relation of a response automaton, the run and the trace of σ on $\mathscr{A}_{\Pi}$ are such that $q_n \in R$. According to (TRESP1) and (TRESP2), the trace of σ on $\mathscr{A}_{\downarrow\Pi}$ is such that $\alpha_n \in \{off, dump\}$.

    From the execution trace on $\mathscr{A}_{\downarrow\Pi}$ and the definition of the enforcement operations, we deduce the following derivations of configurations:

$$(q_{\text{init}}^{\mathscr{A}_{\downarrow\Pi}}, \sigma, \varepsilon) \overset{o_0}{\hookrightarrow} (q_1, \sigma_{1\ldots}, m_1) \cdots \overset{o_{n-2}}{\hookrightarrow} (q_{n-1}, \sigma_{n-1\ldots}, m_{n-1}) \overset{o_{n-1}}{\hookrightarrow} (q_n, \varepsilon, \varepsilon)$$

    with $o_0 \cdot o_1 \cdots o_{n-1} = \sigma$ since the last enforcement operation (α_{n-1}) is either *off* or *dump*.

    By deduction, using the multistep derivations, we have $(q_{\text{init}}^{\mathscr{A}_{\downarrow\Pi}}, \sigma, \varepsilon) \overset{\sigma}{\Rightarrow} (q_n, \varepsilon, \varepsilon)$. That is, $\sigma \Downarrow_{\mathscr{A}_{\downarrow\Pi}} \sigma$, which ensures (5). Besides, according to the acceptance criterion of e-properties, we have $\phi(\sigma)$, which permits to deduce (6), as $\sigma = \sigma$.
  - * If $\sigma \in \Sigma^{\omega}$, then $\varphi(\sigma)$. Using Definition 3 and the definition of a response automaton, we have $inf(\sigma, \mathscr{A}_{\Pi}) \cap R \neq \emptyset$. Formally, $\forall i \in \mathbb{N}, \exists j \in \mathbb{N}, j \geq i \wedge q_j \in R$. It follows that the trace of σ on $\mathscr{A}_{\Pi}$ satisfies $\forall i \in \mathbb{N}, \exists j \in \mathbb{N}, j \geq i \wedge (q_{j-1}, \sigma_{j-1}, q_j) \in trace(\sigma, \mathscr{A}_{\Pi}) \wedge q_j \in R$. Then, we deduce that the trace on the enforcement monitor $\mathscr{A}_{\downarrow\Pi}$ (using the definition of TransResponse, Definition 17) satisfies the property: $\forall i \in \mathbb{N}, \exists j \in \mathbb{N}, j \geq i \wedge (q_{j-1}, \sigma_{j-1}/dump, q_j) \in trace(\sigma, \mathscr{A}_{\downarrow\Pi})$. That is: $\forall i \in \mathbb{N}, \exists j \in \mathbb{N}, j \geq i, \alpha_j \in \{off, dump\}$. Thus we deduce that (using Property 2) $\sigma \Downarrow_{\mathscr{A}_{\downarrow\Pi}} \sigma$, i.e., (5). Moreover, we have (6) as $\varphi(\sigma) \wedge \sigma = \sigma$.
- The second case is $\neg\Pi(\sigma)$. The sequence σ is not accepted by $\mathscr{A}_{\Pi}$, let us distinguish whether σ is finite or not.
  - $\sigma \in \Sigma^*$ and then $\neg\phi(\sigma)$. Let $n = |\sigma|$. There are two cases depending on $Pref_{\prec}(\phi, \sigma) = \emptyset$ or not.
    - *If* $Pref_{\prec}(\phi, \sigma) = \emptyset$, according to the acceptance criterion of response automata, $\mathscr{A}_{\Pi}$ starts in $\overline{R}$ and stays in. We deduce that the execution trace of σ on $\mathscr{A}_{\downarrow\Pi}$ is s.t. $\forall i \geq 0, q_i \notin R$. Using the definition of TransResponse we can find $trace(\sigma, \mathscr{A}_{\downarrow\Pi})$.

Then, the enforcement operation performed by $\mathscr{A}_{\downarrow\Pi}$ is always *halt* or *store*. That is $\sigma \Downarrow_{\mathscr{A}_{\downarrow\Pi}} \varepsilon$ (5). Then $Pref_{\prec}(\phi, \sigma) = \emptyset$ implies that $\forall \sigma' \prec \sigma, \neg\phi(\sigma)$. We have (7).

- *If* $(Pref_{\prec}(\phi, \sigma) \neq \emptyset)$, there is at least one prefix of σ satisfying ϕ. Let us note σ_{good} the longest prefix of σ satisfying ϕ: $\sigma_{good} = Max(Pref_{\prec}(\sigma, \phi))$. Let $k = |\sigma_{good}|$. Then the run and the trace of $\mathscr{A}_{\Pi}$ on σ are s.t. $q_k \in R \wedge \forall i \in [k+1, n], q_i \in \overline{R}$. According to the TransResponse transformation, the trace of σ on $\mathscr{A}_{\downarrow\Pi}$ is s.t. $\alpha_{k-1} = dump \wedge \forall i \in [k, n-1], \alpha_i \in \{store, halt\}$. From the execution trace on $\mathscr{A}_{\downarrow\Pi}$ and the definition of the enforcement operations, we deduce the following derivations of configurations:

$$(q_{\text{init}}^{\mathscr{A}_{\downarrow\Pi}}, \sigma, \varepsilon) \overset{o_0}{\hookrightarrow} \cdots \overset{o_{k-2}}{\hookrightarrow} (q_{k-1}, \sigma_{k-1\cdots}, m_{k-1}) \overset{o_{k-1}}{\hookrightarrow} (q_k, \sigma_{k\cdots}, \varepsilon)$$
$$(q_k, \sigma_{k\cdots}, \varepsilon) \overset{\varepsilon}{\hookrightarrow} (q_{k+1}, \sigma_{k+1\cdots}, m_{k+1}) \overset{\varepsilon}{\hookrightarrow} \cdots \overset{\varepsilon}{\hookrightarrow} (q_n, \varepsilon, m_n)$$

with $\sigma_{good} = \sigma_{\cdots k-1} = o_0 \cdot o_1 \cdots o_{k-1}$. Indeed we have $dump(\sigma_{k-1}, m_{k-1}) = (m_{k-1} \cdot \sigma_{k-1}, \varepsilon)$ and $\forall i \geq k, \alpha_i \in \{store, halt\}$, $\mathscr{A}_{\downarrow\Pi}$ produces ε in output (for $k \leq i \leq n-1$). That is $\sigma_{\cdots k-1} \Downarrow_{\mathscr{A}_{\downarrow\Pi}} \sigma_{\cdots k-1}$ and $\sigma \Downarrow_{\mathscr{A}_{\downarrow\Pi}} \sigma_{\cdots k-1}$. Which ensures (5). Besides, according to the acceptance criterion of *e*-properties, we have $\neg\phi(\sigma)$, which proves (8), as $\sigma_{\cdots k-1} = Max(Pref_{\prec}(\phi, \sigma))$.

- $\sigma \in \Sigma^{\omega}$ and then $\neg\varphi(\sigma)$. This case is similar to the case $\neg\varphi(\sigma)$ for guarantee properties. The acceptance criterion for response automata implies that $vinf(\sigma, \mathscr{A}_{\Pi}) \cap R = \emptyset$. We deduce that there exists n such that the run of σ on $\mathscr{A}_{\Pi}$ is expressed as $run(\sigma, \mathscr{A}_{\Pi}) = q_0 \cdots q_n \cdots$ with $q_0 = q_{\text{init}}^{\mathscr{A}_{\Pi}} \wedge (\forall i \geq n, q_i \in \overline{R})$. Let us consider n_{min} the smallest integer n verifying this property. For $k \leq n_{min}$, it is then possible to apply the previous reasoning (the case $\phi(\sigma)$) for $\sigma_{\cdots k}$. Hence we find an alternation in the run of the execution sequence $\sigma_{\cdots n_{min}}$ between states belonging to R and $\overline{R}$. We find in a similar way that for $k > n_{min}$, $\sigma_{\cdots k} \Downarrow_{\mathscr{A}_{\downarrow\Pi}} \sigma_{\cdots n_{min}}$ and $\phi(\sigma_{\cdots n_{min}})$. It is easy to see that $\sigma_{\cdots n_{min}}$ is the longest prefix (by definition of n_{min}) satisfying ϕ $(\sigma_{\cdots n_{min}} = Max(Pref_{\prec}(\phi, \sigma_{\cdots k})))$.

A.6 Correctness of the TransObligation transformation (Theorem 4 continued, p. 22)

We rely on showing that the EM obtained by applying the TransResponse, Union and Intersection transformations (this EM is correct by construction), and the EM obtained by applying directly TransObligation, are equivalent. To do so, we perform an induction on k where Π is a k-obligation *e*-property. Let us note $\mathscr{A}_{\downarrow\Pi} = \text{TransObligation}(\mathscr{A}_{\Pi})$.

Induction basis We take $k = 1$, Π is a 1-obligation. Let $\mathscr{A}_{\Pi} = (Q^{\mathscr{A}_{\Pi}}, q_{\text{init}}^{\mathscr{A}_{\Pi}}, \Sigma, \longrightarrow_{\mathscr{A}_{\Pi}}, \{(R, P)\})$. Let $\sigma \in \Sigma^{\infty}$. Π can be expressed $\Pi = \Pi_S \cup \Pi_G$ where Π_S (resp. Π_G) is a safety (resp. guarantee) *e*-property recognized by the safety (resp. guarantee) automaton $\mathscr{A}_{\Pi_S} = (Q^{\mathscr{A}_{\Pi}}, q_{\text{init}}^{\mathscr{A}_{\Pi}}, \Sigma, \longrightarrow_{\mathscr{A}_{\Pi}}, \{(\emptyset, P)\})$ (resp. $\mathscr{A}_{\Pi_G} = (Q^{\mathscr{A}_{\Pi}}, q_{\text{init}}^{\mathscr{A}_{\Pi}}, \Sigma, \longrightarrow_{\mathscr{A}_{\Pi}}, \{(R, \emptyset)\})$). These automata differ from $\mathscr{A}_{\Pi}$ only on their accepting states. We can apply the TransResponse transformation on $\mathscr{A}_{\Pi_S}$ seen as a response automaton, and on $\mathscr{A}_{\Pi_G}$ directly. It yields two enforcement monitors $\mathscr{A}_{\downarrow\Pi_S}$ and $\mathscr{A}_{\downarrow\Pi_G}$.

Now, using the definition of TransResponse for $\mathscr{A}_{\downarrow\Pi_S}$, we have $Q^{\mathscr{A}_{\downarrow\Pi_S}} = Q^{\mathscr{A}_{\Pi_S}}$. Moreover, for any transition $q \xrightarrow{a/\alpha}_{\mathscr{A}_{\downarrow\Pi_S}} q'$ in $\mathscr{A}_{\downarrow\Pi_S}$, the enforcement operation α satisfies:

$$(q' \in P \wedge Reach_{\mathscr{A}_{\Pi_S}}(q') \subseteq P) \Rightarrow \alpha = off$$
$$\wedge\, (q' \in P \wedge Reach_{\mathscr{A}_{\Pi_S}}(q') \not\subseteq P) \Rightarrow \alpha = dump$$
$$\wedge\, (q' \in \overline{P}) \Rightarrow \alpha = halt$$

Similarly, using the definition of TransResponse for $\mathscr{A}_{\downarrow\Pi_G}$, for any transition $q \xrightarrow{a/\beta}_{\mathscr{A}_{\downarrow\Pi_G}} q'$ in $\mathscr{A}_{\downarrow\Pi_G}$, the enforcement operation β satisfies:

$$\begin{aligned}&(q' \in \overline{R} \wedge Reach_{\mathscr{A}_{\Pi_G}}(q') \subseteq \overline{R}) \Rightarrow \beta = halt\\ &\wedge (q' \in \overline{R} \wedge Reach_{\mathscr{A}_{\Pi_S}}(q') \not\subseteq \overline{R}) \Rightarrow \beta = store\\ &\wedge (q' \in R) \Rightarrow \beta = off\end{aligned}$$

Now, notice that every transition in $\mathscr{A}_{\sqcup} = \text{Union}(\mathscr{A}_{\downarrow\Pi_S}, \mathscr{A}_{\downarrow\Pi_G})$ is in the form $(q,q) \xrightarrow{a/\gamma}_{\mathscr{A}_{\sqcup}} (q',q')$ where $q, q' \in Q^{\mathscr{A}_{\downarrow\Pi_S}} = Q^{\mathscr{A}_{\downarrow\Pi_G}} = Q^{\mathscr{A}_{\Pi}}$. Moreover, γ satisfies $\gamma = \alpha \sqcup \beta$ where $q \xrightarrow{a/\alpha}_{\mathscr{A}_{\downarrow\Pi_S}} q'$ and $q \xrightarrow{a/\beta}_{\mathscr{A}_{\downarrow\Pi_G}} q'$. Furthermore, for any transition $q \xrightarrow{a/\gamma'}_{\mathscr{A}_{\downarrow\Pi}} q'$ in $\mathscr{A}_{\downarrow\Pi}$, the enforcement operation γ' satisfies the same previous condition ($\alpha \sqcup \beta$). Using the definition of TransObligation, there is a bijection between TransObligation($\mathscr{A}_{\Pi}$) and Union($\mathscr{A}_{\downarrow\Pi_S}, \mathscr{A}_{\downarrow\Pi_G}$): $\forall q \in Q^{\mathscr{A}_{\Pi}}$, the state q in TransObligation($\mathscr{A}_{\Pi}$) is in relation with the state (q,q) in Union($\mathscr{A}_{\downarrow\Pi_S}, \mathscr{A}_{\downarrow\Pi_G}$). This allows to state that TransObligation is correct for 1-obligation properties.

Induction step Let $n \in \mathbb{N}^*$ and suppose that for $k \leq n$, if Π is a k-obligation recognized by a k-obligation automaton $\mathscr{A}_{\Pi}$, then the EM $\mathscr{A}_{\downarrow\Pi} = \text{TransObligation}(\mathscr{A}_{\Pi})$ enforces Π, that is, we have $Enf(\mathscr{A}_{\downarrow\Pi}, \Pi, \mathscr{P}_{\Sigma})$.

Now consider a $(k+1)$-obligation Π, $\mathscr{A}_{\Pi}$ a recognizing $(k+1)$-obligation automaton, and $\mathscr{A}_{\downarrow\Pi} = \text{TransObligation}(\mathscr{A}_{\Pi})$. As Π is a $(k+1)$-obligation property, Π can be expressed as $\bigcap_{i=1}^{k+1} \Pi_i$ where the Π_i are 1-obligation properties (Lemma 1). The expression of Π can be rewritten as $\Pi = (\bigcap_{i=1}^{k} \Pi_i) \cap \Pi_{k+1}$. Using Lemma 1, one can find two recognizing automata $\mathscr{A}_{\Pi/[1,k]}$ recognizing $\bigcap_{i=1}^{k} \Pi_i$ and $\mathscr{A}_{\Pi/\{k+1\}}$ recognizing Π_{k+1}. Using the induction hypothesis, we can apply TransObligation to these two automata to obtain two EMs $\mathscr{A}_{\downarrow\Pi/[1,k]}$ enforcing $\bigcap_{i=1}^{k} \Pi_i$ and $\mathscr{A}_{\downarrow\Pi/\{k+1\}}$ enforcing Π_{k+1}. With the Intersection construction (Definition 15), we obtain the EM $\mathscr{A}'_{\downarrow\Pi} = \text{Intersection}(\mathscr{A}_{\downarrow\Pi/[1,k]}, \mathscr{A}_{\downarrow\Pi/\{k+1\}})$ enforcing (Theorem 2) $(\bigcap_{i=1}^{k} \Pi_i) \cap \Pi_{k+1} = \bigcap_{i=1}^{k+1} \Pi_i$, that is Π.

Now let us examine the EM $\mathscr{A}_{\downarrow\Pi}$ obtained by applying directly the TransObligation transformation on $\mathscr{A}_{\Pi}$. We compare it with $\mathscr{A}'_{\downarrow\Pi}$ obtained by the induction hypothesis and the intersection construction; this EM is correct by construction.

- For $\mathscr{A}_{\downarrow\Pi}$, according to Definition 18 of TransObligation:
  - $Q^{\mathscr{A}_{\downarrow\Pi}} = Q^{\mathscr{A}_{\Pi}}$,
  - $q_{\text{init}}^{\mathscr{A}_{\downarrow\Pi}} = q_{\text{init}}^{\mathscr{A}_{\Pi}}$,
  - and $\forall a \in \Sigma, q \xrightarrow{a/\alpha}_{\mathscr{A}_{\downarrow\Pi}} q'$ where $\alpha = \bigsqcap_{i=1}^{k+1} \bigsqcup(\{\beta_i, \gamma_i\})$.
- For $\mathscr{A}_{\downarrow\Pi}{}'$, according to Definition 15 of the intersection between EMs:
  - $Q^{\mathscr{A}_{\downarrow\Pi}{}'} = Q^{\mathscr{A}_{\downarrow\Pi/\{k+1\}}} \times Q^{\mathscr{A}_{\downarrow\Pi/[1,k]}} = Q^{\mathscr{A}_{\Pi}} \times Q^{\mathscr{A}_{\Pi}}$,
  - $q_{\text{init}}^{\mathscr{A}_{\downarrow\Pi}{}'} = q_{\text{init}}^{\mathscr{A}_{\downarrow\Pi/\{k+1\}}} \times q_{\text{init}}^{\mathscr{A}_{\downarrow\Pi/[1,k]}} = q_{\text{init}}^{\mathscr{A}_{\Pi}} \times q_{\text{init}}^{\mathscr{A}_{\Pi}}$,
  - and $\forall a \in \Sigma, q \xrightarrow{a/\alpha}_{\mathscr{A}_{\downarrow\Pi}{}'} q'$ where $\alpha = \bigsqcap_{i=1}^{k} \bigsqcup(\{\beta_i, \gamma_i\}) \sqcap (\bigsqcup(\{\beta_{k+1}, \gamma_{k+1}))$, i.e., $\alpha = \bigsqcap_{i=1}^{k+1} \bigsqcup(\{\beta_i, \gamma_i\})$.
- where, $\forall i \in [1, k+1]$:
  - β_i is
    - *off* if $q' \in P_i \wedge Reach_{\mathscr{A}_{\Pi}}(q') \cap \overline{P_i} = \emptyset$
    - *dump* if $q' \in P_i \wedge Reach_{\mathscr{A}_{\Pi}}(q') \cap \overline{P_i} \neq \emptyset$
    - *halt* if $q' \notin P_i$.

- γ_i is
  - *off* if $q' \in R_i$
  - *halt* if $q' \notin R_i \wedge \nexists q'' \in R_i, q'' \in Reach_{\mathscr{A}_\Pi}(q')$
  - *store* if $q' \notin R_i \wedge \exists q'' \in R_i, q'' \in Reach_{\mathscr{A}_\Pi}(q')$.

That is, we can exhibit a bijection relation between $\mathscr{A}_{\downarrow\Pi}{}'$ and $\mathscr{A}_{\downarrow\Pi}$: for each state $q \in Q^{\mathscr{A}_\Pi}$, q in $\mathscr{A}_{\downarrow\Pi}$ is in relation with the state (q, q) in $\mathscr{A}_{\downarrow\Pi}{}'$. Formally, between the two EMs $\mathscr{A}_{\downarrow\Pi}$ and $\mathscr{A}_{\downarrow\Pi}{}'$, there is a relation $\mathscr{R} \subseteq (Q^{\mathscr{A}_\Pi} \times (Q^{\mathscr{A}_\Pi} \times Q^{\mathscr{A}_\Pi}))$ defined by $\mathscr{R} = \{(q, (q, q)) \mid q \in Q^{\mathscr{A}_\Pi}\}$. The two EMs are equal (they differ only by the name of their states). As a consequence, the EM produced by directly applying TransObligation on $\mathscr{A}_\Pi$, is correct. This concludes the proof for the TransObligation transformation and *Obligation* properties.

References

1. Havelund K, Goldberg A (2008) Verify your runs. In: Verified software: theories, tools, experiments: first IFIP TC 2/WG 2.3 conference, revised selected papers and discussions, VSTTE 2005, Zurich, Switzerland, October 10–13, 2005, pp 374–383
2. Leucker M, Schallhart C (2009) A brief account of runtime verification. J Log Algebr Program 78:293–303
3. Schneider FB (2000) Enforceable security policies. ACM Trans Inf Syst Secur 3:30–50
4. Hamlen KW, Morrisett G, Schneider FB (2006) Computability classes for enforcement mechanisms. ACM Trans Program Lang Syst 28:175–205
5. Viswanathan M (2000) Foundations for the run-time analysis of software systems. PhD thesis, University of Pennsylvania, Philadelphia, PA, USA, Supervisor-Sampath Kannan and Supervisor-Insup Lee
6. Ligatti J, Bauer L, Walker D (2009) Run-time enforcement of nonsafety policies. ACM Trans Inf Syst Secur 12
7. Ligatti J, Bauer L, Walker D (2005) Enforcing non-safety security policies with program monitors. In: ESORICS, pp 355–373
8. Fong PWL (2004) Access control by tracking shallow execution history. In: Proceedings of the 2004 IEEE symposium on security and privacy. IEEE Computer Society Press, Los Alamitos, pp 43–55
9. Manna Z, Pnueli A (1987) A hierarchy of temporal properties. In: PODC'87: proceedings of the sixth annual ACM symposium on principles of distributed computing. ACM, New York, pp 205–205
10. Chang EY, Manna Z, Pnueli A (1992) Characterization of temporal property classes. In: Automata, languages and programming, pp 474–486
11. Lamport L (1977) Proving the correctness of multiprocess programs. IEEE Trans Softw Eng 3:125–143
12. Alpern B, Schneider FB (1985) Defining liveness. Inf Process Lett 21:181–185
13. Falcone Y, Fernandez JC, Mounier L (2008) Synthesizing enforcement monitors wrt the safety-progress classification of properties. In: Sekar R, Pujari AK (eds) ICISS. Lecture notes in computer science, vol 5352, pp 41–55
14. Chang E, Manna Z, Pnueli A (1992) The safety-progress classification. Technical report, Stanford University, Dept of Computer Science
15. Streett RS (1981) Propositional dynamic logic of looping and converse. In: STOC'81: proceedings of the thirteenth annual ACM symposium on theory of computing. ACM, New York, pp 375–383
16. Falcone Y, Fernandez JC, Mounier L (2009) Runtime verification of safety-progress properties. In: Bensalem S, Peled D (eds) RV. Lecture notes in computer science, vol 5779. Springer, Berlin, pp 40–59
17. Hamlen KW (2006) Security policy enforcement by automated program-rewriting. PhD thesis, Cornell University
18. Ligatti JA (2006) Policy enforcement via program monitoring. PhD thesis, Princeton University
19. Bauer L, Ligatti J, Walker D (2009) Composing expressive runtime security policies. ACM Trans Softw Eng Methodol 18
20. Martinelli F, Matteucci I (2007) Through modeling to synthesis of security automata. Electron Notes Theor Comput Sci 179:31–46
21. Matteucci I (2007) Automated synthesis of enforcing mechanisms for security properties in a timed setting. Electron Notes Theor Comput Sci 186:101–120
22. Erlingsson U, Schneider FB (2000) IRM enforcement of Java stack inspection. In: IEEE symposium on security and privacy, pp 246–255

23. Erlingsson U, Schneider FB (2000) SASI enforcement of security policies: a retrospective. In: WNSP: new security paradigms workshop. ACM Press, New York
24. Kiczales G, Lamping J, Mendhekar A, Maeda C, Lopes C, Loingtier JM, Irwin J (1997) Aspect-oriented programming. Springer, Berlin
25. Falcone Y, Fernandez JC, Mounier L (2009) Enforcement monitoring wrt the safety-progress classification of properties. In: SAC'09: proceedings of the 2009 ACM symposium on applied computing. ACM, New York, pp 593–600
26. The Apache Jakarta Project: Byte Code Engineering Library (2008) http://jakarta.apache.org/bcel/
27. Nethercote N, Seward J (2007) Valgrind: a framework for heavyweight dynamic binary instrumentation. ACM SIGPLAN Not 42:89–100