Unité Mixte de Recherche 5104 CNRS - INPG - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

Synthesizing Enforcement Monitors wrt. the Safety-Progress Classification of Properties

Yliès Falcone, Jean-Claude Fernandez, Laurent Mounier

Verimag Research Report n^o TR-2008-7

December 20, 2009

Synthesizing Enforcement Monitors wrt. the Safety-Progress Classification of Properties

Yliès Falcone, Jean-Claude Fernandez, Laurent Mounier

December 20, 2009

Abstract

Runtime enforcement is a powerful technique to ensure that a program will respect a given set of properties. We extend previous works 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 wrt. the *general* safety-progress classification of properties. Furthermore, we propose a *systematic* technique to produce an enforcing 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, property, safety-progress, synthesis

Notes:

How to cite this report:

@techreport { ,
title = { Synthesizing Enforcement Monitors wrt. the Safety-Progress Classification of Properties},
authors = { Yliès Falcone, Jean-Claude Fernandez, Laurent Mounier},
institution = { Verimag Research Report },
number = {TR-2008-7},
year = { 2008},
note = { }
}

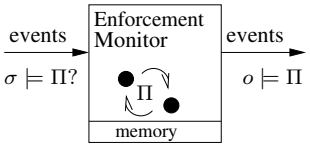
Contents

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 raises. Using formal methods appears as a solution to provide techniques to regain the needed confidence. However, these techniques should remain practical enough to be adopted by software engineers.

Runtime monitoring (see [HG08, LS09] for brief overviews) falls it this category. It consists in supervising at runtime the execution of an underlying program against a set of expected properties. With an appointed monitor, one is able to detect any occurrence of specific property violations. Such a detection might be a sufficient assurance. However, for certain 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 [Sch00] on what has been called *security automata*. In this work the monitors watch the current execution sequence and halt the underlying program whenever it deviates from the desired property. Such security automata are able to enforce the class of safety properties [HMS06], stating that *something bad can never happen*. Later, Viswanathan [Vis00] noticed that the class of enforceable properties is impacted by the computational power of the enforcement monitor. As the enforcement mechanism can implement no more than computable functions, the enforceable properties are included in the decidable ones. More recently, Ligatti and Al. [LBW09, LBW05] showed that it is possible to enforce at runtime more than safety properties. Using a more powerful enforcement mechanism called *edit-automata*, it is possible to enforce the larger class of *infinite renewal properties*, able to express some kinds of *obligations* used in security policies. To better cope with practical resource constraints, Fong [Fon04] studied the effect of memory limitations on enforcement mechanisms.The various mechanisms and operated controls should usually remain *transparent*, meaning that it should always output the *longest correct prefix* of the original execution sequences. This means that the initial sequence is altered in a minimal way.



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 depicted 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 actions (possibly using the memory) which are allowed on the input sequence. Some constraints (*e.g.*, transparency) may exist between σ and o that influence the actions performed by the monitor while reading σ. For instance, let consider a transactional property Π to be enforced, telling 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 been logged yet). On the other hand, whenever the property Π is satisfied, the monitor just need to dump 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 coming events, the input sequence will *never* (respectively will *always*) satisfy the property in the future. In such situation this input sequence can be definitely blocked (respectively the monitor can be switched off, since it not required anymore).

Our contributions. In this paper, we propose to extend these previous works in several directions. Firstly, we study the problem of enforcement relatively to the so-called *Safety-Progress* hierarchy of regular properties [MP87, CMP92b]. This classification differs from the more classical safety-liveness classification [Lam77, AS85] 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 regular property Π can be characterized by a particular kind of (finite-state) recognizing automaton $\mathcal{A}_\Pi$. 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 $\mathcal{A}_\Pi$.

A preliminary version of this paper appeared in ICISS'08 [FFM08]. 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 works 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 Section 3 we recall briefly the necessary elements from the Safety-Progress classification of properties. We add also additional results to this classification. Then, we present our notion of enforcement monitor and their properties in Section 4. We address the problem of enforcement monitor composition in Section 5. Section 6 first exposes the enforcement monitor synthesis and the proofs of its correctness, and then studies the enforcement capability wrt. the classes of the Safety-Progress classification and Section 7 compares these results and the enforcement monitors with similar works. Finally, Section 8 exposes some concluding remarks.

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

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

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 wrt. 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 powersets of execution sequences (like fairness, for instance). Moreover, for a finitary property ϕ and an execution sequence $\sigma \in \Sigma^\infty$, we denote by $\mathrm{Pref}_\prec(\phi, \sigma)$ the set of all (strict) prefixes of σ satisfying ϕ, *i.e.*, $\mathrm{Pref}_\prec(\phi, \sigma) = \{\sigma' \in \phi \mid \sigma' \prec \sigma\}$. This set is a chain (*i.e.*, a totally ordered set) regarding the order relation $\prec$. The (unique) maximal element of the set $\mathrm{Pref}_\prec(\phi, \sigma)$, namely the longest prefix of σ satisfying ϕ (noted $\mathrm{Max}(\mathrm{Pref}_\prec(\phi, \sigma))$) is the maximal element regarding $\prec$ if $\mathrm{Pref}_\prec(\phi, \sigma) \neq \emptyset$. 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 sequence 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 decision procedure reading a finite prefix and examining the satisfaction of this prefix wrt. 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 $\sigma \in Exec(\mathcal{P}_\Sigma)$, we say that σ satisfies (ϕ, φ) when $\sigma \in \Sigma^* \wedge \phi(\sigma) \vee \sigma \in \Sigma^\omega \wedge \varphi(\sigma)$. For an *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 [CMP92b, MP87] 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 *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.

[1] This is the distinction, made by Schneider [Sch00], between properties and (general) policies. The set of properties (defined over single execution sequences) is a subset of the set of policies (defined over sets of execution sequences).

[2] We advocate the fact that using 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.

- *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 more general class containing all linear temporal properties [MP87]. In this paper, we will focus on sub-classes of reactivity to characterize the set of enforceable properties.

The following example introduces informally the aforementioned properties. In Example 3.3, we formalize those properties into *e*-properties.

EXAMPLE 3.1 *Let us consider an operating system where a given operation* op *is allowed only when an authorization* auth *has been granted before. The system is also endowed with three primitives related to authentication:* r_auth *(requesting authentication),* g_auth *(granting authentication),* d_auth *(denying authentication). Then,*

- *the property* Π_1 *stating that "each occurrence of* op *should be preceded by a distinct occurrence of* g_auth*" is a* safety *property;*
- *the property* Π_2 *stating that "In this session, the user should perform an authorization request* r_auth *which should be eventually followed by a grant (*g_auth*) or a deny (*d_auth*)" is a* guarantee *property;*
- *the property* Π_3 *stating that "the system should run forever, unless a* d_auth *is issued and then the user should be disconnected (*disco*) and the system should terminate (*end*)" is an* obligation *property;*
- *the property* 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 *in the meantime" is a* response *property;*
- *the property* Π_5 *stating that "after a* d_auth*, a (forbidden) use of operation* op *should imply that at some point any future call to* r_auth *will always result in a* d_auth *answer" is a* persistence *property.*

The Safety-Progress classification is an alternative to the more classical safety-liveness [Lam77, AS85] 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* [MP87]: 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. We will consider here only the automata view dedicated to *e*-properties.

3.2 The automata view of *e*-properties

For the automata view of te Safety-Progress classification, we follow [MP87, CMP92a] and define *e*-properties using Streett automata. Furthermore, 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 [Str81] and used in [CMP92a]) 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 [MP87] and also only consider finite-state automata in the remainder.

DEFINITION 3.1 (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 a 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 $\mathcal{A} = (Q^{\mathcal{A}}, {q_{\text{init}}}^{\mathcal{A}}, \Sigma, \longrightarrow_{\mathcal{A}}, \{(R_1, P_1), \ldots, (R_m, P_m)\})$ designates a Streett m-automaton.

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

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

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

DEFINITION 3.2 (ACCEPTANCE CONDITION (INFINITE SEQUENCES)) *For* $\sigma \in \Sigma^{\omega}$*, we say that* $\mathcal{A}$ *accepts* σ *if* $\forall i \in \{1, \ldots, m\}$, $vinf(\sigma, \mathcal{A}) \cap R_i \neq \emptyset \vee vinf(\sigma, \mathcal{A}) \subseteq P_i$.

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

DEFINITION 3.3 (ACCEPTANCE CONDITION (FINITE SEQUENCES)) *For a finite-length execution sequence* $\sigma \in \Sigma^*$ *such that* $|\sigma| = n$*, we say that the* m*-automaton* $\mathcal{A}$ *accepts* σ *if* $(\exists q_0, \ldots, q_n \in Q^{\mathcal{A}}, run(\sigma, \mathcal{A}) = q_0 \cdots q_n \wedge q_0 = {q_{\text{init}}}^{\mathcal{A}}$ *and* $\forall i \in \{1, \ldots, m\}, q_n \in P_i \cup R_i)$.

The hierarchy of automata. By setting syntactic restrictions on a Streett automaton, we modify the kind of properties recognized by such an 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 $\{1, \ldots, m\}$:
  - 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.

EXAMPLE 3.2 (CLASSES OF AUTOMATA AND SYNTACTIC RESTRICTIONS) *In Fig. 1 are represented the schematic illustrations for each basic class of automata. The sets of persistent and recurrent states are represented by squares. Transitions between the different kinds of states are represented using arrows between squares.*

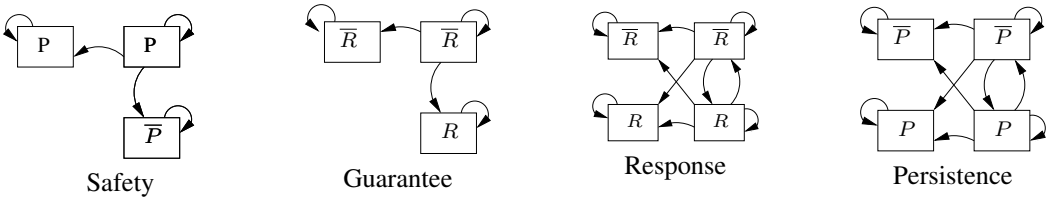


Figure 1: Schematic illustrations of the shapes of Streett automata for basic classes

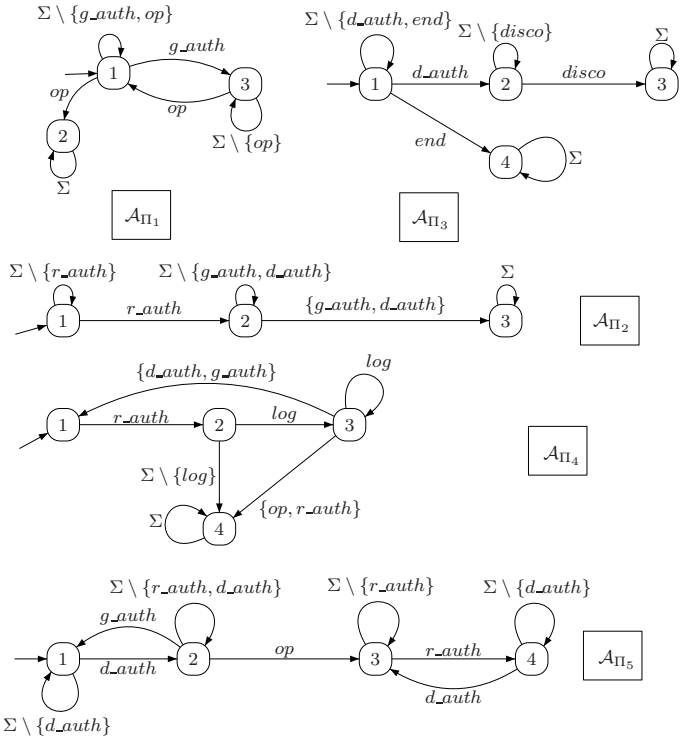


Figure 2: Streett Automata for *e*-properties of Example 3.1

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

EXAMPLE 3.3 (SPECIFYING *e*-PROPERTIES BY STREETT AUTOMATA) *The e-properties previously introduced in Example 3.1 can be specified by Streett automata represented on Fig. 2. Property Π_i is specified by automaton $\mathcal{A}_{\Pi_i}$, $i \in \{1, 2, 3, 4, 5\}$, which initial state is* 1.

- *For $\mathcal{A}_{\Pi_i}$, the set of states is $\{1, 2, 3\}$, $R = \emptyset$, and $P = \{1, 3\}$.*
- *For $\mathcal{A}_{\Pi_2}$, the set of states is $\{1, 2, 3\}$, $P = \emptyset$, and $R = \{3\}$.*
- *For $\mathcal{A}_{\Pi_3}$, the set of states is $\{1, 2, 3, 4\}$, $P = \{1\}$, and $R = \{3\}$.*
- *For $\mathcal{A}_{\Pi_4}$, the set of states is $\{1, 2, 3, 4\}$, $P = \emptyset$, and $R = \{1\}$.*
- *For $\mathcal{A}_{\Pi_5}$, the set of states is $\{1, 2, 3, 4\}$, $P = \{3\}$, and $R = \emptyset$.*

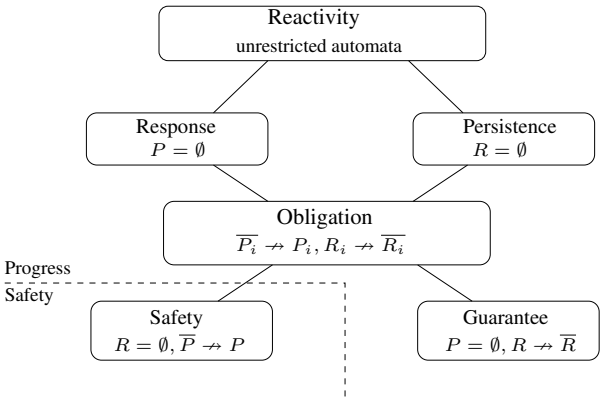


Figure 3: The Safety-Progress classification of *e*-properties

It is possible to link the syntactic characterizations on the automata to the semantic characterization of the properties they specify. This is stated by the following definition (transposed from the initial definition mentioned in [CMP92a]).

DEFINITION 3.4 *An e-property (ϕ, φ) is a regular κ-e-property iff it is specifiable by a finite state κ-automaton, where* $\kappa \in \{\text{safety}, \text{guarantee}, \text{obligation}, \text{response}, \text{persistence}, \text{reactivity}\}$

Given a set of events Σ, we note $\text{Safety}(\Sigma)$ (resp. $\text{Guarantee}(\Sigma)$, $\text{Obligation}(\Sigma)$, $\text{Response}(\Sigma)$, $\text{Persistence}(\Sigma)$) the set of safety (resp. guarantee, obligation, response, persistence) *e*-properties defined over Σ. A graphical representation of the Safety-Progress classification of properties is depicted on Fig. 3.

3.3 Some useful facts about *e*-properties

Properties of automata. Now we give a property of Streett automata related to their accepting pairs. Indeed given a Streett m-obligation automaton (with m accepting pairs), it is possible to characterize the language accepted by the automaton resulting in "forgetting" some accepting pairs of the initial automaton. This is formalized as follows.

LEMMA 3.1 (FORGETTING ACCEPTING PAIRS FOR OBLIGATION PROPERTIES) *Given an m-automaton $\mathcal{A}_\Pi = (Q, q_{\text{init}}, \Sigma, \longrightarrow, \{(R_1, P_1), \ldots, (R_m, P_m)\})$ recognizing an e-property Π. Following [CMP92a], Π can be expressed as $\bigcap_{i=1}^{m} \Pi_i$ where the Π_i are obligation e-properties.*
Given a subset $X \subseteq \{1, \ldots, m\}$, the automaton $\mathcal{A}_{\Pi/X} = (Q, q_{\text{init}}, \Sigma, \longrightarrow, \{(R_i, P_i) \mid i \in X\})$ recognizes the property $\bigcap_{i \in X} \Pi_i$.

Proof. For infinite execution sequences, this proof has been done in [CMP92a]. For finite execution sequences, the proof is a straightforward adaptation. ■

We expose some straightforward consequences of definitions of safety and guarantee *e*-properties.

PROPERTY 3.1 (CLOSURE OF *e*-PROPERTIES) *Considering an e-property $\Pi = (\phi, \varphi)$ defined over an alphabet Σ built from a finitary property ψ, the following facts hold:*

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

  Indeed, we have either $\phi(\sigma)$ or $\varphi(\sigma)$, i.e., all prefixes σ' of σ belong to ψ. Necessarily, all prefixes σ'' of σ' also belong to ψ, that is $\psi(\sigma'')$. By definition, that means $\sigma' \in A_f(\psi)$, i.e., $\phi(\sigma')$ and $\Pi(\sigma')$.

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

  Indeed, $\Pi(\sigma)$ *implies that* σ *has at least one prefix* $\sigma_0 \preceq \sigma$ *belonging to* ψ*:* $\sigma \in E_f(\psi)$. *Then, any continuation of* σ *built using any finite or infinite sequence* σ' *has at least the same prefix belonging to* ψ. *If* $\sigma' \in \Sigma^*$, *we have* $\sigma_0 \preceq \sigma \preceq \sigma \cdot \sigma'$ *and* $\sigma \cdot \sigma' \in E_f(\psi)$. *If* $\sigma' \in \Sigma^\omega$, *we have* $\sigma_0 \preceq \sigma \prec \sigma \cdot \sigma'$ *and* $\sigma \cdot \sigma' \in E(\psi)$.

The following lemma (inspired from [CMP92a]) provides a decomposition of each obligation properties in a normal form. This lemma is proved in Appendix A.1.

LEMMA 3.2 *Any obligation e-property can be represented as the intersection*

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

for some $n > 0$, *where* Safety_i *and* Guarantee_i *are respectively safety and guarantee e-properties. We refer to this presentation as the conjunctive normal form of obligation e-properties.*

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

4 Property enforcement via enforcement monitors

Considering a program $\mathcal{P}_\Sigma$, we aim to build an enforcement monitor for an *e*-property (ϕ, φ), defined 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. On each input event its state evolves and an enforcement operation is performed. Enforcement monitors are parameterized by a set of enforcement operations Ops.

DEFINITION 4.1 (ENFORCEMENT OPERATIONS Ops) *Enforcement operations are aimed to operate a modification of the internal memory of the enforcement monitor and potentially produce an output. More specifically, they 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 4.2 (GENERIC ENFORCEMENT MONITOR (EM(OPS))) *An* enforcement monitor $\mathcal{A}_\downarrow$ *is a 4-tuple* $(Q^{\mathcal{A}_\downarrow}, {q_{\text{init}}}^{\mathcal{A}_\downarrow}, \longrightarrow_{\mathcal{A}_\downarrow}, Ops)$ *defined relatively to a set of events* Σ *and parameterized by a set of enforcement operations* Ops. *The finite set* $Q^{\mathcal{A}_\downarrow}$ *denotes the control states,* ${q_{\text{init}}}^{\mathcal{A}_\downarrow} \in Q^{\mathcal{A}_\downarrow}$ *is the initial state.The complete function* $\longrightarrow_{\mathcal{A}_\downarrow}$: $Q^{\mathcal{A}_\downarrow} \times \Sigma \to Q^{\mathcal{A}_\downarrow} \times Ops$ *is the transition function. In the following we abbreviate* $\longrightarrow_{\mathcal{A}_\downarrow} (q, a) = (q', \alpha)$ *by* $q \xrightarrow{a/\alpha}_{\mathcal{A}_\downarrow} q'$.

Notions of *run* and *trace* (see Section 3.2) are naturally transposed from Streett automata. In the remainder of this section, $\sigma \in \Sigma^\infty$ designates an execution sequence of a program, and $\mathcal{A}_\downarrow = (Q^{\mathcal{A}_\downarrow}, {q_{\text{init}}}^{\mathcal{A}_\downarrow}, \longrightarrow_{\mathcal{A}_\downarrow}, Ops)$ designates an EM(Ops).

DEFINITION 4.3 (RUN AND TRACE) *The* run *of* σ *on* $\mathcal{A}_\downarrow$ *is the sequence of states involved by the execution of* $\mathcal{A}_\downarrow$ *when* σ *is input. It is formally defined as* $run(\sigma, \mathcal{A}_\downarrow) = q_0 \cdot q_1 \cdots$ *where* $q_0 = {q_{\text{init}}}^{\mathcal{A}_\downarrow} \wedge \forall i, (q_i \in Q^{\mathcal{A}_\downarrow} \wedge q_i \xrightarrow{\sigma_i/\alpha_i}_{\mathcal{A}_\downarrow} q_{i+1})$. *The* trace *resulting in the execution of* σ *on* $\mathcal{A}_\downarrow$ *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* $run(\sigma, \mathcal{A}_\downarrow) = q_0 \cdot q_1 \cdots$ *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 4.4 (EM(OPS) CONFIGURATIONS AND DERIVATIONS) *For an EM(Ops) $\mathcal{A}_\downarrow = (Q^{\mathcal{A}_\downarrow}, {q_{\text{init}}}^{\mathcal{A}_\downarrow}, \longrightarrow_{\mathcal{A}_\downarrow}, Ops)$, a* configuration *is a triplet $(q, \sigma, m) \in Q^{\mathcal{A}_\downarrow} \times \Sigma^* \times \Sigma^*$ 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 $(q, \sigma, m) \overset{o}{\hookrightarrow} (q', \sigma', m')$ if and only if $\sigma = a.\sigma' \wedge q \overset{a/\alpha}{\longrightarrow}_{\mathcal{A}_\downarrow} q' \wedge \alpha(a, m) = (o, m')$;*

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 $C \overset{o}{\Longrightarrow}_{\mathcal{A}_\downarrow} C'$, if and only if there exists $k \geq 0$ and configurations $C_0, C_1, \ldots, C_k$ such that $C = C_0$, $C' = C_k$, $C_i \overset{o_i}{\hookrightarrow} C_{i+1}$ 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 4.5 (SEQUENCE TRANSFORMATION) *We define the transformation performed by an EM(Ops) while reading an input sequence $\sigma \in \Sigma^\infty$ (produced by a program $\mathcal{P}_\Sigma$) and producing an output sequence $o \in \Sigma^\infty$. The relation $\Downarrow_{\mathcal{A}_\downarrow} \subseteq \Sigma^\infty \times \Sigma^\infty$ is defined as follows:*

- *The empty sequence ϵ is transformed into itself by $\mathcal{A}_\downarrow$, i.e., $\epsilon \Downarrow_{\mathcal{A}_\downarrow} \epsilon$. This is the case when the underlying program does not produce any event.*
- *The sequence $\sigma \in \Sigma^*$ is transformed by $\mathcal{A}_\downarrow$ from the state $q \in Q^{\mathcal{A}_\downarrow}$ into the sequence $o \in \Sigma^*$, which is noted $(q, \sigma) \Downarrow_{\mathcal{A}_\downarrow} o$, if $\exists q' \in Q^{\mathcal{A}_\downarrow}, m \in \Sigma^*$ such that $(q, \sigma, \epsilon) \overset{o}{\Longrightarrow}_{\mathcal{A}_\downarrow} (q', \epsilon, m)$. That is, if there exists a derivation starting from a configuration which state is q and producing o.*
- *The sequence $\sigma \in \Sigma^*$ is transformed by $\mathcal{A}_\downarrow$ into the sequence $o \in \Sigma^*$, when $({q_{\text{init}}}^{\mathcal{A}_\downarrow}, \sigma) \Downarrow_{\mathcal{A}_\downarrow} o$, which we abbreviate by $\sigma \Downarrow_{\mathcal{A}_\downarrow} o$. That is, if the sequence is transformed from the initial state of the EM.*
- *The sequence $\sigma \in \Sigma^\omega$ is transformed by $\mathcal{A}_\downarrow$ into the sequence $o \in \Sigma^*$, which is noted $\sigma \Downarrow_{\mathcal{A}_\downarrow} o$, if $\exists \sigma' \prec \sigma, \sigma' \Downarrow_{\mathcal{A}_\downarrow} o \wedge \forall \sigma'' \in \Sigma^*, \sigma' \prec \sigma'' \cdot \sigma'' \Downarrow_{\mathcal{A}_\downarrow} o$. 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 $\mathcal{A}_\downarrow$ into the sequence $o \in \Sigma^\omega$, which is noted $\sigma \Downarrow_{\mathcal{A}_\downarrow} o$, if*

$$\begin{array}{ll} & \forall o' \in \Sigma^*, o' \prec o \Rightarrow \exists \sigma'', o'' \in \Sigma^*, o' \prec o'' \wedge \sigma'' \Downarrow_{\mathcal{A}_\downarrow} o'' \\ \wedge & \forall \sigma', o' \in \Sigma^*, \sigma' \prec \sigma \wedge \sigma' \Downarrow o' \Rightarrow o' \prec o. \end{array}$$

That is, if for all prefixes of o, there exists a longer prefix which can be produced.

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 should satisfy 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 verifies Π it should remain unchanged (up to a given equivalence relation), otherwise its *longest prefix* satisfying Π should be issued.

A consequence of transparency is that the *e*-property (ϕ, φ) will be considered as *enforceable* only if each incorrect sequence has a *longest* correct prefix. This means that any infinite incorrect sequence should have only a finite number of correct prefixes, as stated below:

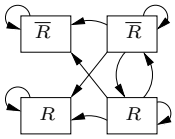
$$\forall \sigma \in \Sigma^{\omega}, \left(\neg\varphi(\sigma) \Rightarrow (\exists \sigma' \in \Sigma^{*}, \sigma' \prec \sigma, \forall \sigma'' \in \Sigma^{*}, \sigma' \prec \sigma'' \prec \sigma \Rightarrow \neg\phi(\sigma''))\right)$$

The set of enforceable *e*-properties is denoted EP. Note also 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* output o produced should be sound and transparent with respect to Π and σ. Furthermore, deciding whether a finite sequence σ verify Π or not should be computable in a finite amount of time (and 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 wrt. the Safety-Progress Classification [FFM09]. We have proved in [FFM09] that the set of response *e*-properties is the set of enforceable properties. The formal proof has been done in [FFM09], here we give a sketch of proof in the automata view for the sake of completeness.



Consider a response *e*-property $\Pi = (\phi, \varphi)$ ($\Pi \in Response(\Sigma)$) recognized by a response automaton. The shape of a response automaton is depicted on the left. Now let us consider an infinite execution sequence $\sigma \in \Sigma^{\omega}$. Let suppose that $\neg\varphi(\sigma)$. It means that, according to the acceptance criterion for infinite sequences (Definition. 3.3), the R-states are not visited infinitely often. In other words, σ has finitely many prefixes belonging to ψ. And, according to the acceptance criterion for finite sequences (Definition. 3.2), finitely many prefixes end in R-states. In order to explain the fact that response properties are *exactly* the set of enforceable properties, the reader is referred to [FFM09] or to the examples in Section 6.2.2 presenting (non enforceable) persistence properties.

Note that, as a straightforward consequence, safety, guarantee, and obligation *e*-properties are enforceable.

Property-enforcement. We define now the notion of property-enforcement by an EM(Ops). The notion of enforcement relates the input sequence produced by the program and given to the EM(Ops) and the output sequence allowed by the EM(Ops) (correct wrt. 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 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 monitor events are not considered). Note that the considered equivalence relation should preserve the *e*-property under consideration.

DEFINITION 4.6 (PROPERTY-ENFORCEMENT$_{\approx}$) *Let us consider an enforceable e-property $\Pi = (\phi, \varphi) \in EP$, we say that $\mathcal{A}_{\downarrow}$ enforces the property (ϕ, φ), relatively to an equivalence relation $\approx$, on a program $\mathcal{P}_{\Sigma}$ (noted $Enf_{\approx}(\mathcal{A}_{\downarrow}, (\phi, \varphi), \mathcal{P}_{\Sigma})$) iff for all $\sigma \in Exec(\mathcal{P}_{\Sigma})$, there exists $o \in \Sigma^{\infty}$, such that the following constraints hold:*

[5] This latter case is avoided in [LBW09] by assuming that properties under consideration always contain ϵ.

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

$$\sigma \Downarrow_{\mathcal{A}_\downarrow} o \quad (1)$$

$$\Pi(\sigma) \Rightarrow \sigma \approx o \quad (2)$$

$$\neg\Pi(\sigma) \wedge \mathit{Pref}_\prec(\phi,\sigma) = \emptyset \Rightarrow o \approx \epsilon \quad (3)$$

$$\neg\Pi(\sigma) \wedge \mathit{Pref}_\prec(\phi,\sigma) \neq \emptyset \Rightarrow o \approx \mathit{Max}(\mathit{Pref}_\prec(\phi,\sigma)) \quad (4)$$

(1),(2),(3) and (4) ensure soundness and transparency of $\mathcal{A}_\downarrow$: (1) stipulates that the sequence σ is transformed by $\mathcal{A}_\downarrow$ into a sequence o; (2) ensures that if σ satisfied already 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 ϕ. Transparency is ensured by the fact that correct execution sequence are not changed, and incorrect ones are restricted to their longest correct prefix.

One may remark that we could have set $\mathit{Max}(\mathit{Pref}_\prec(\phi,\sigma))$ to ϵ when $\mathit{Pref}_\prec(\phi,\sigma) = \emptyset$ and merge the two last constraints. However, we choose to distinguish explicitly the case in which $\mathit{Pref}_\prec(\phi,\sigma) = \emptyset$ 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 be also an incorrect sequences wrt. 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 to some particular, but expressive enough (regarding enforcement), enforcement monitors. This form of monitor is suitable for enforcement under the transparency constraint.

The considered enforcement operations allow enforcement monitors either:

- to **halt** the target program (when the current input sequence irreparably violates the property),
- or to **store** the current event in a *memory device* (when a decision has to be postponed),
- or to **dump** the content of the memory device (when the target program went back to a correct behavior),
- or to switch **off** permanently the monitor (when the property is satisfied for ever).

We give a more precise definition of such enforcement operations.

DEFINITION 4.7 (ENFORCEMENT OPERATIONS $Ops = \{halt, store, dump, off\}$) *In the following we consider a set $Ops = \{halt, store, dump, off\}$ defined as follows: $\forall a \in \Sigma \cup \{\epsilon\}, \forall m \in \Sigma^*$,*

$$halt(a,m) = (\epsilon, m) \quad store(a,m) = (\epsilon, m.a) \quad dump(a,m) = (m.a, \epsilon) \quad off(a,m) = (m.a, \epsilon)$$

(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, this 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 of interest to know when the monitor is not needed anymore.

Furthermore, we assume that, after performing a $halt$ (resp. off operation), an EM cannot perform another operation than $halt$ (resp. off). We also distinguish two subsets of the set of states of an EM(Ops): the states in $Halt$ (resp. Off) are used to represent the states in which the program (resp. monitor) should be stopped. Formally, for the $halt$ operation (it is similar for the off operation), it can be stated as:

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

In the remainder of this article we designate by EM, an instantiated EM(Ops) which respects these constraints.

EXAMPLE 4.1 (ENFORCEMENT MONITOR) *We illustrate the enforcement of some of the e-properties introduced in example 3.1 with EMs.*

- *The right-hand side of Fig. 5 is an EM $\mathcal{A}_{\downarrow\Pi_1}$ for the safety e-property Π_1. $\mathcal{A}_{\downarrow\Pi_1}$ has one halting state, $Halt^{\mathcal{A}_{\downarrow\Pi_1}} = \{2\}$ (filled with gray scale), and its initial state is 1. From this initial state it simply* dumps *a first occurrence of g_auth and moves to state 3, where op operation is allowed (i.e.,* dumped*) and goes back to state 1. Otherwise, if event op precedes g_auth, then it moves to state 2 and halts the underlying program forever.*
- *In the bottom of Fig. 6 is shown an EM $\mathcal{A}_{\downarrow\Pi_2}$ for the guarantee e-property Π_2. The initial state of $\mathcal{A}_{\downarrow\Pi_2}$ is state 1, and it has no halting states. Its behavior is the following: occurrences of events different from a* req_auth *are* stored *in memory as long as a req_auth does not occur. Then events different from a g_auth or a d_auth until one of these events occur. Then the whole memory content is* dumped*. This ensures that the output sequence always satisfies the e-property under consideration.*

4.4 Properties of enforcement monitors

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

PROPERTY 4.1 (ABOUT SEQUENCE TRANSFORMATION) *Given an execution sequence $\sigma \in Exec(\mathcal{P}_\Sigma) \cap \Sigma^\omega$ and an EM $\mathcal{A}_\downarrow$, s.t. the run of σ on $\mathcal{A}_\downarrow$ 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:

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

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

PROPERTY 4.2 (RELATION BETWEEN INPUT, MEMORY, AND OUTPUT) *Input execution sequence, content in the memory, and produced output are related by the following property: $\forall \sigma \in \Sigma^+, \forall \sigma' \in \Sigma^*$,*

$$\begin{array}{c} \exists q \in Q^{\mathcal{A}_\downarrow}, ({q_{\text{init}}}^{\mathcal{A}_\downarrow}, \sigma \cdot \sigma', \epsilon) \overset{o}{\Longrightarrow}_{\mathcal{A}_\downarrow} (q, \sigma', m) \\ \Longrightarrow \\ (\sigma = o \cdot m \wedge q \in Q^{\mathcal{A}_\downarrow} \setminus Halt^{\mathcal{A}_\downarrow}) \vee (o \prec \sigma \wedge q \in Halt^{\mathcal{A}_\downarrow}) \end{array}$$

Proof. The proof is done by induction on the length of the input sequence σ and distinguishing several cases according to the last enforcement operation performed by the EM $\mathcal{A}_\downarrow$. Complete proof can be found in Appendix A.2. ■

It follows that the equivalence relation considered previously for enforcement $\approx$, becomes the equality relation. This is due to the semantics of the enforcement operations we considered. Thus the enforcement predicate $Enf_\approx(\mathcal{A}_\downarrow, (\phi, \varphi), \mathcal{P}_\Sigma)$ becomes $Enf_=(\mathcal{A}_\downarrow, (\phi, \varphi), \mathcal{P}_\Sigma)$ (abbreviated $Enf(\mathcal{A}_\downarrow, (\phi, \varphi), \mathcal{P}_\Sigma)$ in the remainder of this article) when the *e*-property is enforced by $\mathcal{A}_\downarrow$ on $\mathcal{P}_\Sigma$. The following property is a straightforward consequence of property 4.2 and the definition of enforcement operations.

PROPERTY 4.3 (LAST ENFORCEMENT OPERATION) *Given an enforcement monitor $\mathcal{A}_{\downarrow}$, an e-property Π s.t. Enf$(\mathcal{A}_{\downarrow}, (\phi, \varphi), \mathcal{P}_{\Sigma})$ and a finite execution sequence $\sigma \in Exec(\mathcal{P}_{\Sigma}) \cap \Sigma^{+}$ ($|\sigma| = n+1$) which run on $\mathcal{A}_{\downarrow}$ is expressed $(q_0, \sigma_0/\alpha_0, q_1) \cdots (q_n, \sigma_n/\alpha_n, q_{n+1})$, we have:*

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

Meaning that, considering an EM which correctly 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.

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

5 Operations on enforcement monitors

Current development of information systems makes specifications going more and more complex. 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 wrt. Boolean operators: negation, conjunction and disjunction. We also prove their correctness.

5.1 Preliminary notations

We define the complete lattice $(Ops, \sqsubseteq)$ over enforcement operations, where $halt \sqsubseteq store \sqsubseteq dump \sqsubseteq \mathit{off}$ ($\sqsubseteq$ 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 $store$, $\overline{\mathit{off}}$ 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 EM are based on product constructions performed by combining enforcement operations with respect to the complete lattice $(Ops, \sqsubseteq)$.

DEFINITION 5.1 (UNION OF EMS) *Given two EMs $\mathcal{A}_{\downarrow 1} = (Q^{\mathcal{A}_{\downarrow 1}}, {q_{\text{init}}}^{\mathcal{A}_{\downarrow 1}}, \longrightarrow_{\mathcal{A}_{\downarrow 1}}, Ops)$, $\mathcal{A}_{\downarrow 2} = (Q^{\mathcal{A}_{\downarrow 2}}, {q_{\text{init}}}^{\mathcal{A}_{\downarrow 2}}, \longrightarrow_{\mathcal{A}_{\downarrow 2}}, Ops)$ defined relatively to a same input language Σ, we define $\mathcal{A}_{\downarrow\sqcup} = Union(\mathcal{A}_{\downarrow 1}, \mathcal{A}_{\downarrow 2})$ with $Q^{\mathcal{A}_{\downarrow\sqcup}} = (Q^{\mathcal{A}_{\downarrow 1}} \times Q^{\mathcal{A}_{\downarrow 2}})$, ${q_{\text{init}}}^{\mathcal{A}_{\downarrow\sqcup}} = ({q_{\text{init}}}^{\mathcal{A}_{\downarrow 1}}, {q_{\text{init}}}^{\mathcal{A}_{\downarrow 2}})$ and $Halt^{\mathcal{A}_{\downarrow\sqcup}} = Halt^{\mathcal{A}_{\downarrow 1}} \times Halt^{\mathcal{A}_{\downarrow 2}}$. The transition relation of this enforcement monitor is defined by getting the supremum ($\sqcup$) of enforcement operations. More formally $\rightarrow_{\mathcal{A}_{\downarrow\sqcup}}$: $Q^{\mathcal{A}_{\downarrow\sqcup}} \times \Sigma \times Ops \rightarrow Q^{\mathcal{A}_{\downarrow\sqcup}}$ is defined as $\forall a \in \Sigma, \forall q = (q_1, q_2) \in Q^{\mathcal{A}_{\downarrow\sqcup}}$,*

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

Note that this construction does not introduce non-determinism in the resulting EM. 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 EM. However, one can notice that the resulting EM may not be minimal (like the one illustrated in the following example).

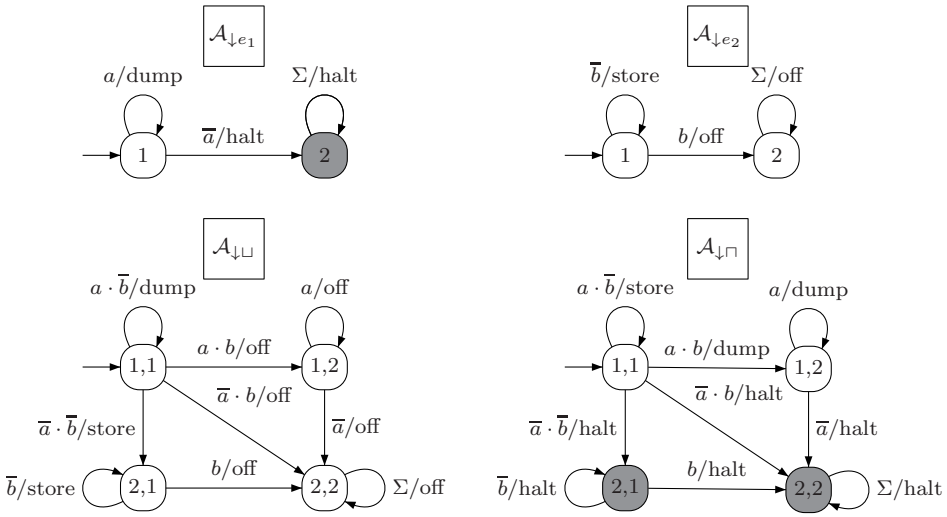


Figure 4: Union and intersection of two enforcement monitors: $\mathcal{A}_{\downarrow e_1}$ and $\mathcal{A}_{\downarrow e_2}$

EXAMPLE 5.1 (UNION OF EMS) *Let 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 consider the following requirement: "Either a is always true or a 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 will use two EMs, one for the requirement "a is always true", and the second for the requirement "b will be eventually true". Next, we use the union of EMs in order to obain an EM for the initial requirement. The underling vocabulary of the EMs is made of the possible events which are observed from the system, i.e., $\Sigma = \{\overline{a}\overline{b}, \overline{a}b, a\overline{b}, ab\}$. The vocabulary represents all possible evaluations of the propositional variables a and b. We use a boolan notation, e.g., the event $a\overline{b}$ represents that a is evaluated to true and b to false, the event a represents that a is true and b is either true or false.

The EMs we consider are depicted in Fig. 4, halting states are in gray scale:

- *$\mathcal{A}_{\downarrow e_1}$ enforces the requirement "a is always true". State 2 is a halting state.*
- *$\mathcal{A}_{\downarrow e_2}$ enforces the requirement "b is eventually true".*
- *$\mathcal{A}_{\downarrow \sqcup}$ enforces the requirement "a is always true* or *b is eventually true". It is the EM union $\mathcal{A}_{\downarrow \sqcup}$ built from the EMs $\mathcal{A}_{\downarrow e_1}, \mathcal{A}_{\downarrow e_2}$. Following the definition of the construction, the set of states is the cartesian product $Q^{\mathcal{A}_{\downarrow e_1}} \times Q^{\mathcal{A}_{\downarrow e_2}}$. The initial state is $(1,1)$. Note that there is no halting state in this resulting EM since $Halt^{\mathcal{A}_{\downarrow e_1}} \times Halt^{\mathcal{A}_{\downarrow e_2}} = \emptyset$. This EM is not minimal and can be easily minimized by merging the states $(1,2)$ and $(2,2)$.*

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

DEFINITION 5.2 (INTERSECTION OF EMS) *Given two EM $\mathcal{A}_{\downarrow 1} = (Q^{\mathcal{A}_{\downarrow 1}}, {q_{\text{init}}}^{\mathcal{A}_{\downarrow 1}}, \longrightarrow_{\mathcal{A}_{\downarrow 1}}, Ops)$, $\mathcal{A}_{\downarrow 2} = (Q^{\mathcal{A}_{\downarrow 2}}, {q_{\text{init}}}^{\mathcal{A}_{\downarrow 2}}, \longrightarrow_{\mathcal{A}_{\downarrow 2}}, Ops)$ defined relatively to a same input language Σ and enforcement operations Ops, we define the EM intersection $\mathcal{A}_{\downarrow \sqcap} = (Q^{\mathcal{A}_{\downarrow \sqcap}}, {q_{\text{init}}}^{\mathcal{A}_{\downarrow \sqcap}}, \longrightarrow_{\mathcal{A}_{\downarrow \sqcap}}, Ops)$ with $Q^{\mathcal{A}_{\downarrow \sqcap}} = (Q^{\mathcal{A}_{\downarrow 1}} \times Q^{\mathcal{A}_{\downarrow 2}})$, ${q_{\text{init}}}^{\mathcal{A}_{\downarrow \sqcap}} = ({q_{\text{init}}}^{\mathcal{A}_{\downarrow 1}}, {q_{\text{init}}}^{\mathcal{A}_{\downarrow 2}})$ and $Halt^{\mathcal{A}_{\downarrow \sqcap}} = Halt^{\mathcal{A}_{\downarrow 1}} \times Q^{\mathcal{A}_{\downarrow 2}} \cup Q^{\mathcal{A}_{\downarrow 1}} \times Halt^{\mathcal{A}_{\downarrow 2}}$. The transition relation of this enforcement monitor is defined by getting the supremum ($\sqcap$) of enforcement operations. More formally $\rightarrow_{\mathcal{A}_{\downarrow \sqcap}}$: $Q^{\mathcal{A}_{\downarrow \sqcap}} \times \Sigma \times Ops \rightarrow Q^{\mathcal{A}_{\downarrow \sqcap}}$ is defined as $\forall a \in \Sigma, \forall q = (q_1, q_2) \in Q^{\mathcal{A}_{\downarrow \sqcap}}$,*

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

EXAMPLE 5.2 (INTERSECTION OF TWO ENFORCEMENT MONITORS) *Similarly to Example 5.1, 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 $\mathcal{A}_{\downarrow\sqcap}$ is shown in Fig. 4. It has two halting states: $(2,1)$ and $(2,2)$. This EM is not minimal and can be easily minimized by merging the states $(2,1)$ and $(2,2)$.*

THEOREM 5.1 *Given $\mathcal{A}_{\downarrow\Pi_1} = (Q^{\mathcal{A}_{\downarrow\Pi_1}}, {q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi_1}}, \longrightarrow_{\mathcal{A}_{\downarrow\Pi_1}}, Ops)$ and $\mathcal{A}_{\downarrow\Pi_2} = (Q^{\mathcal{A}_{\downarrow\Pi_2}}, {q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi_2}}, \longrightarrow_{\mathcal{A}_{\downarrow\Pi_2}}, Ops)$, two enforcement monitors enforcing two enforceable properties $\Pi_1, \Pi_2 \in EP$ on a program $\mathcal{P}_\Sigma$, the property $\Pi_1 \vee \Pi_2$ (resp. $\Pi_1 \wedge \Pi_2$) is enforced by the enforcement monitor union (resp. intersection). More formally: $\forall \Pi_1, \Pi_2 \subseteq \Sigma^\infty$,*

$$Enf(\mathcal{A}_{\downarrow\Pi_1}, \Pi_1, \mathcal{P}_\Sigma) \wedge Enf(\mathcal{A}_{\downarrow\Pi_2}, \Pi_2, \mathcal{P}_\Sigma) \Rightarrow Enf(Union(\mathcal{A}_{\downarrow\Pi_1}, \mathcal{A}_{\downarrow\Pi_2}), \Pi_1 \vee \Pi_2, \mathcal{P}_\Sigma)$$
$$Enf(\mathcal{A}_{\downarrow\Pi_1}, \Pi_1, \mathcal{P}_\Sigma) \wedge Enf(\mathcal{A}_{\downarrow\Pi_2}, \Pi_2, \mathcal{P}_\Sigma) \Rightarrow Enf(Intersection(\mathcal{A}_{\downarrow\Pi_1}, \mathcal{A}_{\downarrow\Pi_2}), \Pi_1 \wedge \Pi_2, \mathcal{P}_\Sigma)$$

Proof. We tackle the proof of the *Union* operator. For $i \in \{1,2\}$, we have $Enf(\mathcal{A}_{\downarrow\Pi_i}, \Pi_i, \mathcal{P}_\Sigma)$, *i.e.*, for all $\sigma \in Exec(\mathcal{P}_\Sigma)$, there exists $o_i \in \Sigma^*$:

$$\sigma \Downarrow_{\mathcal{A}_{\downarrow\Pi_i}} o_i \tag{5}$$
$$\Pi_i(\sigma) \Rightarrow \sigma = o_i \tag{6}$$
$$\neg\Pi_i(\sigma) \wedge Pref_\prec(\phi_i, \sigma) = \emptyset \Rightarrow o_i = \epsilon \tag{7}$$
$$\neg\Pi_i(\sigma) \wedge Pref_\prec(\phi_i, \sigma) \neq \emptyset \Rightarrow o_i = Max(Pref_\prec(\phi_i, \sigma)) \tag{8}$$

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

$$\sigma \Downarrow_{\mathcal{A}_\sqcup} o \tag{9}$$
$$\Pi(\sigma) \Rightarrow \sigma = o \tag{10}$$
$$\neg\Pi(\sigma) \wedge Pref_\prec(\phi, \sigma) = \emptyset \Rightarrow o = \epsilon \tag{11}$$
$$\neg\Pi(\sigma) \wedge Pref_\prec(\phi, \sigma) \neq \emptyset \Rightarrow o = Max(Pref_\prec(\phi, \sigma)) \tag{12}$$

We first consider $\sigma \in \Sigma^*$, the following proof is done by induction on $|\sigma|$.

Induction basis. For the induction basis $|\sigma| = 0$; we have $\sigma = \epsilon$. Then we have easily (9) as $\epsilon \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \epsilon$. Moreover, $Pref_\prec(\phi, \epsilon) = \emptyset$, which gives (11).

Inductive step. Let $n \in \mathbb{N}$ and suppose that for all sequences σ s.t. $|\sigma| = n$, we have the existence of an output o of $\mathcal{A}_\sqcup$ s.t. (10) and (11).

As $\sigma \Downarrow_{\mathcal{A}_\sqcup} o$ (induction hypothesis), there exists a configuration $(q, \epsilon, m) \in Q \times \Sigma^* \times \Sigma^*$ such that $(q_{\text{init}}, \sigma, \epsilon) \overset{o}{\Rightarrow} (q, \epsilon, m)$. Which implies that $(q_{\text{init}}, \sigma \cdot a, \epsilon) \overset{o}{\Rightarrow} (q, a, m)$. That is, after reading σ, the EM $\mathcal{A}_\sqcup$ 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 $(q, a, m) \overset{o'}{\hookrightarrow} (q', \epsilon, m')$ with $\alpha(a, m) = (o', m'), \alpha \in Ops$. By reading $\sigma \cdot a$, $\mathcal{A}_\sqcup$ produces the output $o \cdot o'$. Also, the reading of $\sigma \cdot a$ on $\mathcal{A}_{\downarrow\Pi_i}$, $i \in \{1,2\}$, induces the following evolution of configurations:

$$(q_{\text{init}}, \sigma \cdot a, \epsilon) \overset{o_i}{\Rightarrow} (q_i, a, m_i) \overset{o_i'}{\hookrightarrow} (q_i', \epsilon, m_i'),$$

with $\alpha_i(a, m_i) = (o_i', m_i')$; $q_i, q_i' \in Q^{\mathcal{A}_{\downarrow\Pi_i}}$; $m_i, m_i', o_i, o_i' \in \Sigma^*$.

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

- The first case is $\phi(\sigma \cdot a)$. In this case, that means that either $\phi_1(\sigma \cdot a)$ or $\phi_2(\sigma \cdot a)$. Let treat the case $\phi_1(\sigma \cdot a)$, the case $\phi_2(\sigma \cdot a)$ is identical. As $Enf(\Pi_1, \mathcal{A}_{\downarrow\Pi_1}, \mathcal{P}_\Sigma)$, we have that $\exists o_1 \in \Sigma^*, \sigma \cdot a \Downarrow_{\mathcal{A}_{\downarrow\Pi_1}} o_1$. Moreover, $\phi_1(\sigma \cdot a)$ implies that $o_1 = \sigma \cdot a$. Inevitably the last enforcement operation of $\mathcal{A}_{\downarrow\Pi_1}$ is *dump* or *off*, *i.e.*, $\alpha_1 = dump \vee \alpha_1 = off$ (Prop. 4.3). It follows that $\alpha = \sqcup(\{\alpha_1, \alpha_2\}) = dump \vee \alpha = \sqcup(\{\alpha_1, \alpha_2\}) = off$. According to the definition of enforcement operation and Prop. 4.2, we obtain in both cases that $\sigma \cdot a \Downarrow_{\mathcal{A}_\sqcup} \sigma \cdot a$, *i.e.*, (9) and (10).

- The second case is $\neg\phi(\sigma \cdot a)$. It implies that $\neg\phi_1(\sigma \cdot a) \wedge \neg\phi_2(\sigma \cdot a)$. 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 $Enf(\Pi_i, \mathcal{A}_{\downarrow\Pi_i}, \mathcal{P}_\Sigma)$, $\neg\phi_i(\sigma \cdot a)$ gives us $\exists o_i \in \Sigma^*, o_i = Max(Pref_\prec(\phi_i, \sigma \cdot a))$. Now, we have either $o_1 \prec o_2$, $o_2 \prec o_1$ or $o_1 = o_2$.

    * Let us consider the case $o_1 \prec o_2$ ($o_2 \prec o_1$ is identical). In this case, we have $\forall o_1' \in \Sigma^* \cdot o_1 \prec o_1' \preceq \sigma \cdot a \cdot \neg\phi_1(o_1')$, and $\forall o_2' \in \Sigma^* \cdot o_2 \prec o_2' \preceq \sigma \cdot a \cdot \neg\phi_2(o_2')$. Then $o_1 \prec o_2$ implies that $o_2 = Max(Pref_\prec(\phi, \sigma \cdot a))$. We have now to show that $\sigma \cdot a \Downarrow_{\mathcal{A}_\sqcup} o_2$. Let us examine the sequence of enforcement operations performed by $\mathcal{A}_\sqcup$. We have that $o_2 \Downarrow_{\mathcal{A}_\sqcup} o_2$, as the last enforcement operation performed while reading $o_2 \preceq \sigma \cdot a$ is a *dump* ($\mathcal{A}_\sqcup$ is obtained by taking the upperbound of enforcement operations).
    * Similarly in the case $o_1 = o_2$, we have that $o_1 = o_2 = Max(Pref_\prec(\phi, \sigma \cdot a))$. 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 $Enf(\Pi_i, \mathcal{A}_{\downarrow\Pi_i}, \mathcal{P}_\Sigma)$, $\neg\phi_i(\sigma \cdot a)$ gives us $\sigma \cdot a \Downarrow_{\mathcal{A}_{\downarrow\Pi_i}} \epsilon, i \in \{1, 2\}$.
  - The third case is $Pref_\prec(\phi_1, \sigma \cdot a) = \emptyset \vee Pref_\prec(\phi_2, \sigma \cdot a) = \emptyset$. Since $Enf(\Pi_i, \mathcal{A}_{\downarrow\Pi_i}, \mathcal{P}_\Sigma), i \in \{1, 2\}$, 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 we can compare similarly to the first case.

For infinite sequences, the reasoning is similar to the proof of Theorem 5.2. It is done on the shape of the sequence of enforcement operations and by distinguishing according to $\varphi(\sigma)$. ■

The proof for the intersection operator is conducted similarly. A consequence of this theorem is that the class EP of enforceable properties is closed under union and intersection.

5.3 Negation

Considering a safety or guarantee (enforceable) *e*-property[7], we show how an EM can be transformed so as the *e*-property of the resulting EM enforces the negation of the original enforced *e*-property.

DEFINITION 5.3 (NEGATION OF AN EM) *Given $\mathcal{A}_{\downarrow\Pi} = (Q^{\mathcal{A}_{\downarrow\Pi}}, {q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi}}, \longrightarrow_{\mathcal{A}_{\downarrow\Pi}}, Ops)$ with input language Σ and enforcing Π, a safety or guarantee e-property, we define the $Negation(\mathcal{A}_{\downarrow\Pi}) = \overline{\mathcal{A}_{\downarrow\Pi}} = (Q^{\overline{\mathcal{A}_{\downarrow\Pi}}}, {q_{\text{init}}}^{\overline{\mathcal{A}_{\downarrow\Pi}}}, \longrightarrow_{\overline{\mathcal{A}_{\downarrow\Pi}}}, Ops)$ as:*

- $Q^{\overline{\mathcal{A}_{\downarrow\Pi}}} = Q^{\mathcal{A}_\downarrow\Pi}$, ${q_{\text{init}}}^{\overline{\mathcal{A}_{\downarrow\Pi}}} = {q_{\text{init}}}^{\mathcal{A}_\downarrow\Pi}$,
- $Halt^{\overline{\mathcal{A}_{\downarrow\Pi}}} = \{q \in Q^{\overline{\mathcal{A}_{\downarrow\Pi}}} \mid \exists q' \in Q^{\overline{\mathcal{A}_{\downarrow\Pi}}}, \exists a \in \Sigma, q' \xrightarrow{a/halt}_{\overline{\mathcal{A}_{\downarrow\Pi}}} q\}$,
- $\rightarrow_{\overline{\mathcal{A}_{\downarrow\Pi}}}$ *is the smallest relation verifying* $q \xrightarrow{a/\overline{\alpha}}_{\overline{\mathcal{A}_{\downarrow\Pi}}} q'$ *if* $q \xrightarrow{a/\alpha}_{\mathcal{A}_{\downarrow\Pi}} q'$.

EXAMPLE 5.3 *When restraining the vocabulary of the previous example to $\Sigma = \{a, b\}$, the EM $\mathcal{A}_{\downarrow e_1}$ is the negation of $\mathcal{A}_{\downarrow e_2}$. In this case, exactly either a or b is true on a same event, and $\overline{a} = b$, $\overline{b} = a$.*

[7] It is only useful to deal with safety and guarantee *e*-properties. Indeed, the negation of a response *e*-property is a persistence (thus not enforceable) and obligation *e*-properties are Boolean combination of safety and guarantee *e*-properties.

THEOREM 5.2 (Negation of an EM) *Given* $\mathcal{A}_{\downarrow\Pi} = (Q^{\mathcal{A}_{\downarrow\Pi}}, {q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi}}, \longrightarrow_{\mathcal{A}_{\downarrow\Pi}}, Ops)$ *an EM defined relatively to an input language* Σ *enforcing* Π*, the EM obtained using the* Negation *transformation enforces* $\overline{\Pi}$*. More formally:* $\forall \Pi \subseteq \Sigma^* \times \Sigma^\omega$

$$Enf(\mathcal{A}_{\downarrow\Pi}, \Pi, \mathcal{P}_\Sigma) \Rightarrow Enf(\text{Negation}(\mathcal{A}_{\downarrow\Pi}), \overline{\Pi}, \mathcal{P}_\Sigma)$$

The proof principle follows the one used for the proof of the Union construction. The complete proof can be found in Appendix A.3.

6 Enforcement wrt. the Safety-Progress classification

We now study how to practically enforce *e*-properties of the Safety-Progress hierarchy (Section 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. This construction technique is specific to each class of properties. This synthesis technique provides also a characterization of the set of enforceable properties wrt. the Safety-Progress classification of properties.

6.1 From a recognizing automaton to an enforcement monitor

We define four general operations those purpose is to transform a Streett automaton recognizing a safety (resp. guarantee, obligation, response) *e*-property into an enforcement monitor enforcing the same *e*-property.

6.1.1 Safety *e*-properties

DEFINITION 6.1 (SAFETY TRANSFORMATION) *Given a Streett safety-automaton* $\mathcal{A}_\Pi = (Q^{\mathcal{A}_\Pi}, {q_{\text{init}}}^{\mathcal{A}_\Pi}, \Sigma, \longrightarrow_{\mathcal{A}_\Pi}, (\emptyset, P))$ *recognizing a safety (enforceable) e-property* $\Pi \in Safety(\Sigma)$*. We define the transformation* $\text{TransSafety}(\mathcal{A}_\Pi) = \mathcal{A}_{\downarrow\Pi} = (Q^{\mathcal{A}_{\downarrow\Pi}}, {q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi}}, \longrightarrow_{\mathcal{A}_{\downarrow\Pi}}, Ops)$ *such that:*

- $Q^{\mathcal{A}_{\downarrow\Pi}} = Q^{\mathcal{A}_\Pi}, {q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi}} = {q_{\text{init}}}^{\mathcal{A}_\Pi}$
- $\rightarrow_{\mathcal{A}_{\downarrow\Pi}}$ *is defined as the smallest relation verifying:*
  - $q \xrightarrow{a/off}_{\mathcal{A}_{\downarrow\Pi}} q'$ *if* $q' \in P \wedge Reach(q') \subseteq P \wedge q \xrightarrow{a}_{\mathcal{A}_\Pi} q'$ (TSAF1)
  - $q \xrightarrow{a/dump}_{\mathcal{A}_{\downarrow\Pi}} q'$ *if* $q' \in P \wedge Reach(q') \not\subseteq P \wedge q \xrightarrow{a}_{\mathcal{A}_\Pi} q'$ (TSAF2)
  - $q \xrightarrow{a/halt}_{\mathcal{A}_{\downarrow\Pi}} q'$ *if* $q' \notin P \wedge q \xrightarrow{a}_{\mathcal{A}_\Pi} q'$ (TSAF3)

Note that there is no transition from $q \in R$ *to* $q' \notin R$*. Here, one can notice that the halting states (resp. off states) are the* $\overline{P}$ *states (resp.* $\{q \in P \mid Reach_{\mathcal{A}_\Pi}(q) \subseteq P\}$*).*

Informally, the behavior of an EM $\mathcal{A}_{\downarrow\Pi}$ obtained from $\text{TransSafety}(\mathcal{A}_\Pi)$ can be understood as follows. While the current execution sequence satisfies the underlying property (*i.e.*, while $\mathcal{A}_\Pi$ remains in P-states), the EM "allows" the events: it switches off if the property is also satisfied forever ($Reach(q') \subseteq P$), else it dumps each input event if the may not be satisfied in the future ($Reach(q') \not\subseteq P$). Once the execution sequence deviates from the property (*i.e.*, when $\mathcal{A}_\Pi$ reaches a $\overline{P}$-state), then it halts immediately the underlying program with a $halt$ operation.

EXAMPLE 6.1 (SAFETY TRANSFORMATION) *The right-hand side of Fig. 5 shows the EM* $\mathcal{A}_{\downarrow\Pi_1}$ *obtained using transformation* TransSafety *applied to* $\mathcal{A}_{\Pi_1}$*.*

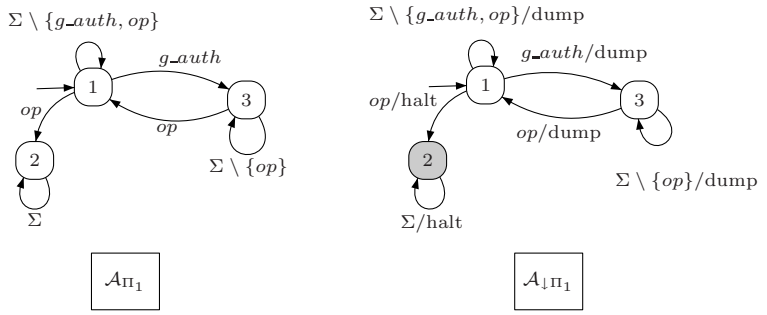


Figure 5: Recognizing automaton and EM for the safety *e*-property Π_1

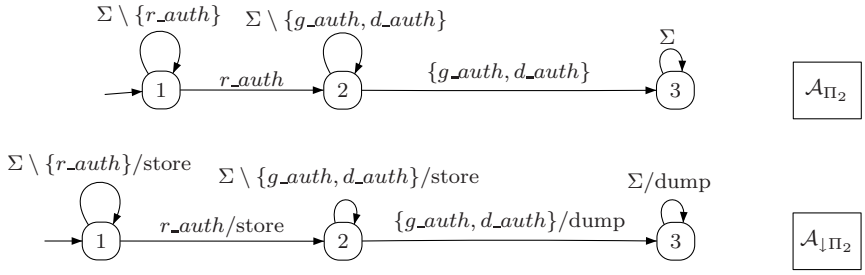


Figure 6: A guarantee-automaton and the corresponding EM for property Π_2

6.1.2 Guarantee *e*-properties

DEFINITION 6.2 (GUARANTEE TRANSFORMATION) *Given a Streett guarantee-automaton* $\mathcal{A}_\Pi = (Q^{\mathcal{A}_\Pi}, {q_{\text{init}}}^{\mathcal{A}_\Pi}, \Sigma, \longrightarrow_{\mathcal{A}_\Pi}, (R, \emptyset))$ *recognizing a guarantee (enforceable) e-property* $\Pi \in Guarantee(\Sigma)$*. We define the transformation* $\text{TransGuarantee}(\mathcal{A}_\Pi) = \mathcal{A}_{\downarrow\Pi} = (Q^{\mathcal{A}_{\downarrow\Pi}}, {q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi}}, \longrightarrow_{\mathcal{A}_{\downarrow\Pi}}, Ops)$ *s.t.:*

- $Q^{\mathcal{A}_{\downarrow\Pi}} = Q^{\mathcal{A}_\Pi}, {q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi}} = {q_{\text{init}}}^{\mathcal{A}_\Pi}$,
- $\rightarrow_{\mathcal{A}_{\downarrow\Pi}}$ *is defined as the smallest relation verifying:*
  - $q \xrightarrow{a/off}_{\mathcal{A}_{\downarrow\Pi}} q'$ *if* $q' \in R \wedge q \xrightarrow{a}_{\mathcal{A}_\Pi} q'$ (TGUAR1)
  - $q \xrightarrow{a/halt}_{\mathcal{A}_{\downarrow\Pi}} q'$ *if* $q' \notin R \wedge q \xrightarrow{a}_{\mathcal{A}_\Pi} q' \wedge Reach_{\mathcal{A}_\Pi}(q') \cap R = \emptyset$ (TGUAR2)
  - $q \xrightarrow{a/store}_{\mathcal{A}_{\downarrow\Pi}} q'$ *if* $q' \notin R \wedge q \xrightarrow{a}_{\mathcal{A}_\Pi} q' \wedge Reach_{\mathcal{A}_\Pi}(q') \cap R \neq \emptyset$ (TGUAR3)

Note that there is no transition from $q \in R$ *to* $q' \in \overline{R}$*. And, as* $P = \emptyset$*, we do not have transition from* $q \in P$ *to* $q' \in P$*. One can notice that* $Halt^{\mathcal{A}_{\downarrow\Pi}} = \{q \in Q^{\mathcal{A}_{\downarrow\Pi}} \mid \nexists q' \in Reach_{\mathcal{A}_\Pi}(q) \wedge q' \in R\}$ *and* $Off^{\mathcal{A}_{\downarrow\Pi}} = R$*. We note* $\mathcal{A}_{\downarrow\Pi} = \text{TransGuarantee}(\mathcal{A}_\Pi)$.

An EM $\mathcal{A}_{\downarrow\Pi}$ obtained from $\text{TransGuarantee}(\mathcal{A}_\Pi)$ behaves as follows. While the current execution sequence does not satisfy the underlying property (*i.e.*, while $\mathcal{A}_\Pi$ remains in $\overline{R}$-states), it stores each entered event in its memory. If $\mathcal{A}_\Pi$ reaches a state from which the property cannot be satisfied anymore, then the enforcement monitor halts the underlying program. Once, the execution sequence satisfies the property (*i.e.*, when $\mathcal{A}_\Pi$ reaches an R-state), it switches off and dumps the content of the memory and the current event. The following example illustrates this principle.

EXAMPLE 6.2 (GUARANTEE TRANSFORMATION) *Fig. 6 (down) shows the EM enforcing* Π_2*, obtained by the* TransGuarantee *on* $\mathcal{A}_{\Pi_2}$ *transformation. One can notice that this EM has no halting state since from all states an R-state is reachable.*

6.1.3 Obligation *e*-properties

Since obligation properties are defined as positive Boolean combinations of safety and guarantee properties [CMP92a], the following corollary is a straightforward consequence of theorems 5.1 and 6.1. The proof of this corollary provides a systematic construction when the automata recognizing the *e*-property is decomposable into a product of safety and guarantee automata.

COROLLARY 6.1 *Given a program $\mathcal{P}_\Sigma$, an* obligation *property Π is enforceable by an EM obtained by using the safety and guarantee transformations and the Union and Intersection operations.*

Proof. Consider a program $\mathcal{P}_\Sigma$, an obligation property Π on $\mathcal{P}_\Sigma$, we want to show that there exists an EM $\mathcal{A}_{\downarrow\Pi}$ s.t. $Enf(\mathcal{A}_{\downarrow\Pi}, \Pi, \mathcal{P}_\Sigma)$.

We know that there exists $n > 0$ s.t. Π can be expressed $\bigcap_{i=1}^{n}(\mathrm{Safety}_i \cup \mathrm{Guarantee}_i)$.

This proof is done by an induction on n. From each Safety_i (resp. $\mathrm{Guarantee}_i$) we build an EM using the TransSafety (resp. TransGuarantee) transformation. Then using the union and intersection construction we obtain the EM for the property. ■

We also define a direct transformation for obligation properties. Informally the TransObligation transformation combines the effects of the two previous transformations by using information of each accepting pair.

DEFINITION 6.3 (OBLIGATION TRANSFORMATION) *Given a Streett obligation-automaton $\mathcal{A}_\Pi = (Q^{\mathcal{A}_\Pi}, {q_{\text{init}}}^{\mathcal{A}_\Pi}, \Sigma, \longrightarrow_{\mathcal{A}_\Pi}, \{(R_1, P_1), \ldots, (R_m, P_m)\})$ recognizing an m-obligation (enforceable) e-property $\Pi \in Obligation(\Sigma)$. We define* $\mathrm{TransObligation}(\mathcal{A}_\Pi) = \mathcal{A}_{\downarrow\Pi} = (Q^{\mathcal{A}_{\downarrow\Pi}}, {q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi}}, \longrightarrow_{\mathcal{A}_{\downarrow\Pi}}, Ops)$ *s.t.:*

- $Q^{\mathcal{A}_{\downarrow\Pi}} = Q^{\mathcal{A}_\Pi}, {q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi}} = {q_{\text{init}}}^{\mathcal{A}_\Pi}$,
- $\rightarrow_{\mathcal{A}_{\downarrow\Pi}}$ *is defined as the smallest relation verifying:*

  $q \xrightarrow{a/\alpha}_{\mathcal{A}_{\downarrow\Pi}} q'$ *if* $q \xrightarrow{a}_{\mathcal{A}_\Pi} q'$ *and* $\alpha = \sqcap_{i=1}^{m}(\sqcup(\beta_i, \gamma_i))$ *where the* β_i *and* γ_i *are obtained in the following way:*

  - $\beta_i = off$ *if* $q' \in P_i \wedge Reach_{\mathcal{A}_\Pi}(q') \subseteq P_i$
  - $\beta_i = dump$ *if* $q' \in P_i \wedge Reach_{\mathcal{A}_\Pi}(q') \not\subseteq P_i$
  - $\beta_i = halt$ *if* $q' \notin P_i$
  - $\gamma_i = off$ *if* $q' \in R_i$
  - $\gamma_i = halt$ *if* $q' \notin R_i \wedge Reach_{\mathcal{A}_\Pi}(q') \subseteq \overline{R_i}$
  - $\gamma_i = store$ *if* $q' \notin R_i \wedge Reach_{\mathcal{A}_\Pi}(q') \not\subseteq \overline{R_i}$

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 that $Halt = \{q \in \bigcup_{i=1}^{m}(\overline{P_i} \cap \overline{R_i}) \mid Reach_{\mathcal{A}_\Pi}(q) \subseteq \bigcup_i(\overline{P_i} \cap \overline{R_i})\}$ and $Off = \{q \in \bigcap_{i=1}^{m}(P_i \cup R_i) \mid Reach_{\mathcal{A}_\Pi}(q) \subseteq \bigcap_i(P_i \cup R_i)\}$. We note $\mathcal{A}_{\downarrow\Pi} = \mathrm{TransGuarantee}(\mathcal{A}_\Pi)$.

EXAMPLE 6.3 (OBLIGATION TRANSFORMATION) *At the bottom of Fig. 7 is depicted the EM enforcing the 1-obligation property Π_3 of Example 3.3, obtained by the* TransObligation *transformation. One can notice that this EM has no halting state.*

6.1.4 Response *e*-properties

Finding a transformation for a *response* property Π needs to slightly extend the definition of TransGuarantee to deal with transitions of a Streett automaton leading from states belonging to R to states belonging to $\overline{R}$ (such transitions are absent when Π is a *guarantee* property). Therefore, we introduce a new transformation called TransResponse obtained from the TransGuarantee transformation by adding a rule to deal with the aforementioned difference.

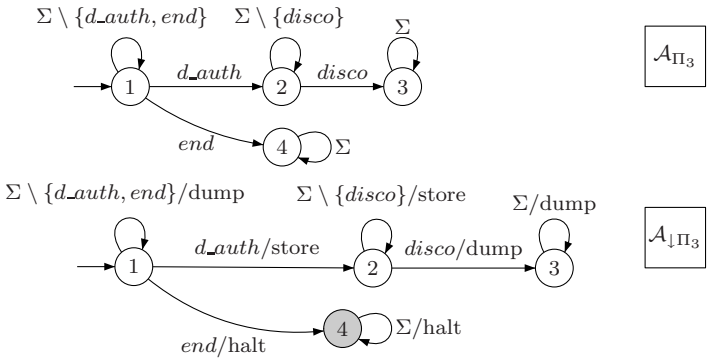


Figure 7: A 1-obligation-automaton and the corresponding EM for property Π_3

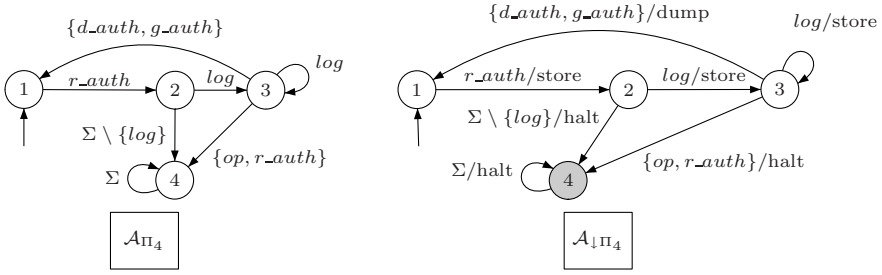


Figure 8: A response-automaton and the corresponding EM for property Π_4

DEFINITION 6.4 (RESPONSE TRANSFORMATION) *Given a Streett response-automaton* $\mathcal{A}_\Pi = (Q^{\mathcal{A}_\Pi}, q_{\text{init}}{}^{\mathcal{A}_\Pi}, \Sigma, \longrightarrow_{\mathcal{A}_\Pi}, (R, \emptyset))$ *recognizing a response (enforceable) e-property* $\Pi \in \mathit{Response}(\Sigma)$*. We define the transformation* $\text{TransResponse}(\mathcal{A}_\Pi) = \mathcal{A}_{\downarrow\Pi} = (Q^{\mathcal{A}_{\downarrow\Pi}}, q_{\text{init}}{}^{\mathcal{A}_{\downarrow\Pi}}, \longrightarrow_{\mathcal{A}_{\downarrow\Pi}}, \mathit{Ops})$ *using the following rules for* $\rightarrow_{\mathcal{A}_{\downarrow\Pi}}$*:*

- $q \xrightarrow{a/store}_{\mathcal{A}_{\downarrow\Pi}} q'$ *if* $q \in R \wedge q' \notin R \wedge q \xrightarrow{a}_{\mathcal{A}_\Pi} q' \wedge \mathit{Reach}_{\mathcal{A}_\Pi}(q') \not\subseteq \overline{R}$ (TRESP1)
- $q \xrightarrow{a/halt}_{\mathcal{A}_{\downarrow\Pi}} q'$ *if* $q \in R \wedge q' \notin R \wedge q \xrightarrow{a}_{\mathcal{A}_\Pi} q' \wedge \mathit{Reach}_{\mathcal{A}_\Pi}(q') \subseteq \overline{R}$ (TRESP2)
- $q \xrightarrow{a/dump}_{\mathcal{A}_{\downarrow\Pi}} q'$ *if* $q \in R \wedge q' \in R \wedge q \xrightarrow{a}_{\mathcal{A}_\Pi} q' \wedge \mathit{Reach}_{\mathcal{A}_\Pi}(q') \not\subseteq R$ (TRESP3)
- $q \xrightarrow{a/off}_{\mathcal{A}_{\downarrow\Pi}} q'$ *if* $q \in R \wedge q' \in R \wedge q \xrightarrow{a}_{\mathcal{A}_\Pi} q' \wedge \mathit{Reach}_{\mathcal{A}_\Pi}(q') \subseteq R$ (TRESP4)

An EM $\mathcal{A}_{\downarrow\Pi}$ obtained via the $\text{TransResponse}(\mathcal{A}_\Pi)$ transformation processes the entered execution sequence and enforces the originally recognized property. Informally the principle is similar to the one of guarantee enforcement, except that there might be an alternation in the run between states of R and $\overline{R}$. While the current execution sequence does not satisfy the underlying property (the current state is in $\overline{R}$), it stores each event of the input sequence (or halts the underlying program if the property can not be satisfied in the future). Once, the execution sequence satisfies the property (the current state is in R), it dumps the content of the memory and the events stored so far (or switches off if the property is satisfied for ever).

EXAMPLE 6.4 (RESPONSE TRANSFORMATION) *The right-hand side of Fig. 8 shows the EM enforcing the response e-property* Π_4 *introduced in example 3.3, obtained by the* TransResponse *transformation. There is one halting state: state* 4*.*

6.2 Enforcement wrt. the Safety-Progress classification

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

6.2.1 Enforceable properties.

Now, we tackle the question of monitor synthesis for the enforceable properties, namely the safety, guarantee, obligation, and response properties. Given *any* safety (resp. guarantee, obligation, response) property Π, and a Streett automaton recognizing Π, one can *synthesize* from this automaton an enforcing monitor for Π using systematic transformations. The following theorem proves the correctness of these transformations. It also proves alternatively that safety, guarantee, obligation, and response properties are enforceable.

THEOREM 6.1 (Correctness of the transformations) *Given a program $\mathcal{P}_\Sigma$, a* regular *safety (resp. guarantee, obligation, response) e-property $\Pi \in \mathrm{Safety}(\Sigma)$ (resp. $\Pi \in \mathrm{Guarantee}(\Sigma), \Pi \in \mathrm{Obligation}(\Sigma), \Pi \in \mathrm{Response}(\Sigma)$) is enforceable on $\mathcal{P}_\Sigma$ by an EM obtained by the application of the safety (resp. guarantee, obligation, response) transformation on the automaton recognizing Π. More formally, given $\mathcal{A}_\Pi$ recognizing Π, we have:*

$$
\begin{aligned}
(\Pi \in \mathrm{Safety}(\Sigma) \wedge \mathcal{A}_{\downarrow\Pi} = \mathrm{TransSafety}(\mathcal{A}_\Pi)) &\Rightarrow \mathit{Enf}(\mathcal{A}_{\downarrow\Pi}, \Pi, \mathcal{P}_\Sigma),\\
(\Pi \in \mathrm{Guarantee}(\Sigma) \wedge \mathcal{A}_{\downarrow\Pi} = \mathrm{TransGuarantee}(\mathcal{A}_\Pi)) &\Rightarrow \mathit{Enf}(\mathcal{A}_{\downarrow\Pi}, \Pi, \mathcal{P}_\Sigma).\\
(\Pi \in \mathrm{Obligation}(\Sigma) \wedge \mathcal{A}_{\downarrow\Pi} = \mathrm{TransObligation}(\mathcal{A}_\Pi)) &\Rightarrow \mathit{Enf}(\mathcal{A}_{\downarrow\Pi}, \Pi, \mathcal{P}_\Sigma).\\
(\Pi \in \mathrm{Response}(\Sigma) \wedge \mathcal{A}_{\downarrow\Pi} = \mathrm{TransResponse}(\mathcal{A}_\Pi)) &\Rightarrow \mathit{Enf}(\mathcal{A}_{\downarrow\Pi}, \Pi, \mathcal{P}_\Sigma).
\end{aligned}
$$

For each proof, we have to show for Π that for $\mathcal{A}_{\downarrow\Pi}$, result of the transformation from a recognizing automaton $\mathcal{A}_\Pi$ of Π, we have: for all execution sequences $\sigma \in \mathit{Exec}(\mathcal{P}_\Sigma)$, there exists $o \in \Sigma^*$ s.t. the following constraints hold:

$$\sigma \Downarrow_{\mathcal{A}_{\downarrow\Pi}} o \tag{13}$$

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

$$\neg\Pi(\sigma) \wedge \mathit{Pref}_\prec(\phi, \sigma) = \emptyset \Rightarrow o = \epsilon \tag{15}$$

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

Proof.
For these proofs we note $\mathcal{A}_{\downarrow\Pi} = (Q^{\mathcal{A}_{\downarrow\Pi}}, {q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi}}, \longrightarrow_{\mathcal{A}_{\downarrow\Pi}}, \mathit{Ops})$ the EM obtained using a transformation. The proof for safety properties is fully presented. Then for the others classes of properties, we only sketch the proofs; full versions can be found in appendix.

For the class of safety properties. We note $\mathcal{A}_\Pi = (Q^{\mathcal{A}_\Pi}, {q_{\text{init}}}^{\mathcal{A}_\Pi}, \Sigma, \longrightarrow_{\mathcal{A}_\Pi}, (\emptyset, P))$. Let us consider an execution sequence of the program $\sigma \in \mathit{Exec}(\mathcal{P}_\Sigma)$. We study the effect of the submission of σ to $\mathcal{A}_{\downarrow\Pi}$. We will associate the execution of σ on $\mathcal{A}_\Pi$ to the execution of σ on $\mathcal{A}_{\downarrow\Pi}$. The execution of σ on $\mathcal{A}_\Pi$ 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 $\mathcal{A}_{\downarrow\Pi}$ with $q_0 = {q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi}}$. We distinguish depending on whether the sequence σ satisfies the property Π or not.

- *The first case is* $\Pi(\sigma)$. The automaton $\mathcal{A}_\Pi$ accepts σ, let us distinguish whether σ is finite or not.
  - If $\sigma \in \Sigma^*$, let $n = |\sigma|$. As σ is accepted by $\mathcal{A}_\Pi$, and according to (Def. 3.2), we have that while reading σ, the automaton "stays in P-states" (we have $R = \emptyset$ and no transition from $\overline{P}$-states to P-states since $\mathcal{A}_\Pi$ is a safety automaton).
    If $\sigma = \epsilon$, we have (13) as $\epsilon \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \epsilon$. Moreover, $\mathit{Pref}_\prec(\phi, \epsilon) = \emptyset$, which gives (15).
    Else ($\sigma \neq \epsilon$), according to the constraints of the transition relation of a safety-automaton, (TRANSSAFETY1), and (TRANSSAFETY2), the trace of σ on $\mathcal{A}_{\downarrow\Pi}$ is such that $\forall i \leq n, \alpha_i = \mathit{dump} \vee \alpha_i = \mathit{off}$.
    From the execution trace on $\mathcal{A}_{\downarrow\Pi}$ and the definition of the enforcement operations ($\forall i < n$, $\mathit{dump}(\sigma_i, \epsilon) = (\sigma_i, \epsilon)$), we deduce the following derivation of configurations:
    $({q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi}}, \sigma, \epsilon) \overset{\sigma_0}{\hookrightarrow} (q_1, \sigma_{1\ldots}, \epsilon) \cdots \overset{\sigma_{n-2}}{\hookrightarrow} (q_{n-1}, \sigma_{n-1\ldots}, \epsilon) \overset{\sigma_{n-1}}{\hookrightarrow} (q_n, \sigma_n, \epsilon)$.

By deduction, using the multistep derivations, we have $(q_{\text{init}}{}^{\mathcal{A}_{\downarrow\Pi}}, \sigma, \epsilon) \overset{\sigma}{\Rightarrow}_{\mathcal{A}_{\downarrow\Pi}} (q_{n-1}, \epsilon, \epsilon)$. That is $\sigma \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \sigma$. Which ensures (13). Besides, according to the acceptance criterion of e-properties, we have $\phi(\sigma)$, which permits us to deduce (14), as $\sigma = \sigma$.

– If $\sigma \in \Sigma^\omega$, then using (Def. 3.2) of a Streett automaton and the definition of a safety-automaton, we have $vinf(\sigma) \subseteq P$. Meaning that the only states visited infinitely often are P-states. As there is no transition from states in $\overline{P}$ to states in R, no state of $\overline{P}$ was visited. The run of σ on $\mathcal{A}_\Pi$ is s.t. $\forall i \in \mathbb{N}, q_i \in P$. Which by a similar reasoning leads us to find the trace of σ on $\mathcal{A}_{\downarrow\Pi}$. The sequence of enforcement operations complies to the regular expression $dump^\omega + dump^* \cdot off^\omega$. We have $\forall \sigma' \prec \sigma, \sigma' \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \sigma'$. It follows that $\sigma \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \sigma$. We have then (13) and (14).

- *The second case is* $\neg\Pi(\sigma)$. The sequence σ is not accepted by $\mathcal{A}_\Pi$. There are two cases depending on $Pref_\prec(\phi, \sigma) = \emptyset$ or not.

  – *If* $Pref_\prec(\phi, \sigma) = \emptyset$. According to the constraints on safety automata, $\mathcal{A}_\Pi$ starts in a $\overline{P}$-state and stays in (there is no transition from $\overline{P}$-states to P-states. We deduce that the execution trace of σ on $\mathcal{A}_\Pi$ is s.t. $\forall i > 0, q_i \notin P$. Using the definition of the $\mathrm{TransSafety}$ transformation (TRANSSAFETY3) we can find $trace(\sigma, \mathcal{A}_{\downarrow\Pi})$. Then, the enforcement operation performed by $\mathcal{A}_{\downarrow\Pi}$ is always $halt$. That is $\sigma \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \epsilon$. (13). Then $Pref_\prec(\phi, \sigma) = \emptyset$ implies that $\forall \sigma' \prec \sigma, \neg\phi(\sigma)$. We have (15).

  – *Else (*$Pref_\prec(\phi, \sigma) \neq \emptyset$*)*, there is at least one prefix of σ satisfying ϕ. As Π is a safety property, we can decompose σ into $\sigma_{good} \cdot \sigma_{bad}$ where σ_{good} is the longest prefix of σ satisfying ϕ (and hence Π) and $\sigma_{bad} \neq \epsilon$.

  Using a reasoning similar to the first case, we can find on $\mathcal{A}_\Pi$ a trace $(q_0, \sigma_0, q_1) \cdots (q_{s-1}, \sigma_{s-1}, q_s)$, with $s = |\sigma_{good}|$. From which we can associate a trace on $\mathcal{A}_{\downarrow\Pi}$: $(q_0, \sigma_0/dump, q_1) \cdots (q_{s-1}, \sigma_{s-1}/dump, q_s)$.

  The execution of σ_{bad} produces a trace $(q_s, \sigma_s, q_{s+1}) \cdots (q_{s+j}, \sigma_{s+j}, q_{s+j+1}) \cdots$, with $j > 0$. Then, $\mathcal{A}_\Pi$ switches from $q_{s-1} \in P$ to $q_s \notin P$. Afterwards, according to (TSAF2), we obtain for $i > s$, $(q_0, \sigma_0/dump, q_1) \cdots (q_{s-1}, \sigma_{s-1}/dump, q_s) \cdot (q_s, \sigma_s/halt, q_{s+1}) \cdots (q_i, \sigma_i/halt, q_{i+1}) \cdots$ (with $\forall i > s, q_i \notin P$). As in the first case, since $dump(\sigma_i, \epsilon) = (\sigma_i, \epsilon)$ we find that $(q_{\text{init}}{}^{\mathcal{A}_{\downarrow\Pi}}, \sigma, \epsilon) \overset{\sigma_{good}}{\Rightarrow} (q_{s-1}, \sigma_{bad}, \epsilon)$. Then, using that $\forall i > |\sigma_{good}|, halt(\sigma_i, \epsilon) = (\epsilon, \epsilon)$, $\forall i > |\sigma_{good}|, (q_{\text{init}}{}^{\mathcal{A}_{\downarrow\Pi}}, \sigma, \epsilon) \overset{\sigma_{good}}{\Rightarrow} (q_i, \sigma_{i\ldots}, \epsilon)$, *i.e.*, $\sigma_{\ldots i} \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \sigma_{good}$. Thereby we have:

    * For all $\sigma' \in \Sigma^*$ *s.t.* $\sigma' \preceq \sigma_{good} \prec \sigma$, we have $\sigma' \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \sigma'$, which gives (13). Then as $\phi(\sigma')$, we get (14).
    * For all $\sigma' \in \Sigma^*$ *s.t.* $\sigma_{good} \prec \sigma' \prec \sigma$, we have $\sigma' \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \sigma_{good}$ (13). Moreover $\neg\phi(\sigma')$, and σ_{good} is the longest prefix of σ satisfying ϕ, that is $Pref_\prec(\phi, \sigma) \neq \emptyset$, and $Max(Pref_\prec(\phi, \sigma)) = \sigma_{good}$ (which gives (15)).

For the class of guarantee properties. The proof for guarantee properties follows the same principle of the proof for safety properties. Thus we only sketch the proof, the compete proof can be found in Appendix A.4. Let us consider an execution sequence $\sigma \in Exec(\mathcal{P}_\Sigma)$. By examining the run of σ on $\mathcal{A}_\Pi$, we can find, using the definition of $\mathrm{TransGuarantee}$ transformation, the shape of the sequence of enforcement operations. We distinguish two cases: $\Pi(\sigma)$ or not.

- The first case is $\Pi(\sigma)$. It means that the run of σ on $\mathcal{A}_\Pi$ reached an R-state and stays in it. If σ is finite, then the sequence is of the form $store^* \cdot off^+$. Else (σ is infinite) the shape of the sequence is $store^* \cdot off^\omega$.

- The second case is $\neg\Pi(\sigma)$. It means that the run of σ on $\mathcal{A}_\Pi$ does not reach a R-state and stays in $\overline{R}$-states. If σ is finite, the sequence is of the form $(store + halt)^*$. Else (σ is infinite) the sequence is of the form $(store + halt)^\omega$.

For the class of obligation properties. To prove the theorem, we show that the following property holds: given $\mathcal{A}_\Pi$ recognizing Π,

$$\forall k \geq 1, \big(\Pi \in \mathrm{Obligation_k}(\Sigma) \wedge \mathcal{A}_{\downarrow\Pi} = \mathrm{TransObligation}(\mathcal{A}_\Pi)\big) \Rightarrow \mathit{Enf}(\mathcal{A}_{\downarrow\Pi}, \Pi, \mathcal{P}_\Sigma)$$

To do so, we perform an induction on k where the property Π is a k-obligation *e*-property. Complete proof can be found in Appendix A.5.

- *Induction basis.* For the induction basis, $k = 1$, Π is a simple obligation. We note $\mathcal{A}_\Pi = (Q^{\mathcal{A}_\Pi}, {q_{\mathrm{init}}}^{\mathcal{A}_\Pi}, \Sigma, \longrightarrow_{\mathcal{A}_\Pi}, \{(R, P)\})$. Let $\sigma \in \Sigma^\infty$, the proof is done by studying the effect of the submission of σ to $\mathcal{A}_{\downarrow\Pi}$. We distinguish two cases depending on whether $\Pi(\sigma)$ or not. In both cases, the reasoning leads us to remark that for a simple obligation, using the constraints of simple-obligation automata, it amounts to the case where Π is either a safety or a guarantee property. Thus we can apply the previous reasonings.

- *Induction step.* The proof is done by showing that the two following EMs are equal:

  – The first EM is obtained by following the compositional way. First we decompose the (k+1)-obligation automaton into an intersection product of one simple obligation and one k-obligation automata. This decomposition uses the principle exposed in Lemma 3.1. It results into a product of automata recognizing the initial (k+1)-obligation property Π. Second, we apply the $\mathrm{TransObligation}$ transformation (on the two obligation automata) and $\mathrm{Intersection}$ operation. This EM is correct by construction.

  – The second EM is obtained by the direct application of $\mathrm{TransObligation}$ on the automaton recognizing Π the (k+1)-obligation property.

  The equality is shown by pointing out a bijection between those EMs.

For the class of response properties. Similarly to guarantee properties, we examine the run of an execution sequence $\sigma \in \mathit{Exec}(\mathcal{P}_\Sigma)$, and then, following the definition of the $\mathrm{TransResponse}$ transformation, we deduce the shape of enforcement operations. Complete proof can be found in Appendix A.6.

- The first case is $\Pi(\sigma)$. We distinguish according to whether σ is a finite sequence or not.

  – If σ is a finite sequence then it means that the run of σ on $\mathcal{A}_\Pi$ ends in a R-state. Hence, the last enforcement operation performed by $\mathcal{A}_{\downarrow\Pi}$ is dump. The shape of the sequence of enforcement operations is $(\mathit{store} + \mathit{dump})^* \cdot (\mathit{dump} + \mathit{off}^*)$.

  – If σ is an infinite sequence, then it means that an R-state is visited infinitely often. Hence, $\mathcal{A}_{\downarrow\Pi}$ performs regularly the dump operation or persistently a off operation. Then the shape of the sequence of enforcement operations is $(\mathit{store}^* \cdot \mathit{dump})^\omega + \big((\mathit{store} + \mathit{dump})^* \cdot \mathit{off}^\omega\big)$.

- The second case is $\neg\Pi(\sigma)$. We distinguish according to whether σ is a finite sequence or not.

  – If σ is a finite sequence then it means that the run of σ on $\mathcal{A}_\Pi$ ends in a $\overline{R}$-state. Hence, following the last enforcement operation performed by $\mathcal{A}_{\downarrow\Pi}$ is store or halt. The shape of the sequence of enforcement operations is $(\mathit{halt} + \mathit{store} + \mathit{dump})^* \cdot (\mathit{halt} + \mathit{store})$.

  – If σ is an infinite sequence, then it means that R-states are visited *finitely* often. Hence, $\mathcal{A}_{\downarrow\Pi}$ performs always halt or store operation from a certain prefix of σ. Then the shape of the sequence of enforcement operations is $(\mathit{halt} + \mathit{store} + \mathit{dump})^* \cdot (\mathit{halt} + \mathit{store})^\omega$.

■

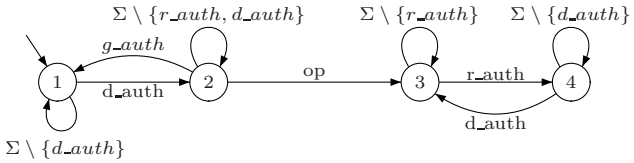


Figure 9: Automaton recognizing the persistence *e*-property Π_5

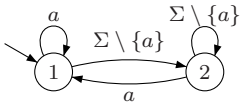


Figure 10: Non-enforceable persistence property

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 [FFM09]. We give two examples of pure persistence properties and explain with more details than in [FFM09] the enforcement limitation and why it is not desirable to enforce pure persistence properties in practice.

EXAMPLE 6.5 (NON-ENFORCEABLE PURE PERSISTENCE PROPERTIES) *This property is recognized by the Streett automaton depicted on Fig. 9 (with acceptance criterion $inf(\sigma, \Pi_5) \subseteq P$ and $P = \{1, 3\}$). This property is not enforceable since it has incorrect infinite sequences with an infinite number of correct prefixes. Indeed consider $\sigma_{bad} = d\_auth \cdot op \cdot (r\_auth \cdot d\_auth)^\omega$. Such an (infinite) execution sequence does not satisfy Π_5 since $inf(\sigma_{bad}, \Pi_5) = \{3, 4\} \not\subseteq \{1, 3\}$. Moreover according to the acceptance criterion for finite sequences, each prefix σ'_{bad} of the form $d\_auth \cdot op \cdot (r\_auth \cdot d\_auth)^*$ 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 6.6 (NON-ENFORCEABLE PURE PERSISTENCE PROPERTIES) *An example of (pure) persistence property (depicted in Fig. 10), defined on $\Sigma \supseteq \{a\}$ is $\Sigma^* \cdot a^\omega$ stating that "it will be eventually true that a always occurs". One can notice that this property is neither a safety, guarantee nor obligation property. Similarly to the property of the previous example, this property has infinite incorrect sequences with an infinite number of correct prefixes.*

One can understand the enforcement limitation intuitively with the following argument: if this property were enforceable it would imply that an enforcement monitor can decide from a certain point that the underlying program will always produce the event a. However such a decision can never be taken by a monitor without memorizing the entire execution sequence beforehand. This is unrealistic for an infinite sequence. From a more formal perspective, the enforcement limitation can be understood as follows. As stated in Section 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 wrt. 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 wrt. 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 wrt. the *e*-property and it is the longest correct prefix of the input sequence.

Thus the enforcement monitor would output the same input sequence: $(a \cdot b)^\omega$; which is not correct wrt. the considered *e*-property.

REMARK 6.1 *Note that, as a consequence, properties of the reactivity class (containing the persistence class) are not enforceable by our enforcement monitors.*

REMARK 6.2 *Due to the transparency constraint, the later properties still remain not enforceable even with more powerfull enforcement mechanisms (e.g., EM(Ops) with more expressive enforcement operations).*

7 Related works

This section provides a comparison with related works in runtime enforcement monitoring and shows the differences of the approach proposed here. Also, we refer to the comparison of enforcement mechanisms provided in [HMS06] as it sets up enforcement at runtime wrt. other enforcement mechanisms from a computational point of view.

Schneider automata. Schneider introduced security automata as the first runtime mechanism for enforcing properties. In [Sch00], he defined a variant of Büchi automaton which runs in parallel with an underlying program. These automata were endowed of the ability to halt the program whenever the security automaton detects a violation of the property under scrutiny. Schneider announced in his paper that the set of enforceable properties with this kind of security automata is the set of safety properties. Then [HMS06] Schneider, Hamlen, and Morisett refined the set of enforceable properties using such a mechanism. They notably shown that these security automata were in fact restrained by some computational limits. Indeed, Viswanathan [Vis00] noticed that the class of enforceable properties is impacted by the computational power of the enforcement monitor. As the enforcement mechanism can implement no more than computable functions, the enforceable properties are included in the decidable ones. Hence, they showed in [HMS06] that the set of safety properties is a strict superior limit to the power of enforcement execution monitor.

Edit-automata. Ligatti and al. [LBW09, LBW05, Lig06, BLW09] 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-automata 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).

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 [LBW09]. The set of renewal properties is a superset of safety properties, contains some liveness properties (but not all).

Shallow history automata. Fong [Fon04] studied the effect of restraining the capacity of the runtime execution monitor and the effect on the enforcement power. Shallow History Automata keep as history a set of access events the underlying program made. Fong shown that these automata can enforce a set of properties stricly contained in the set of properties enforceable by Schneider's automata. The result has been generalized by using abstraction mechanisms on Schneider's automata. Fong's classification has a practical interest, in the sense that it studies the effect of practical programming constraint (limited memory). It also shows that some classical security policies remain enforceable using such shallow automata.

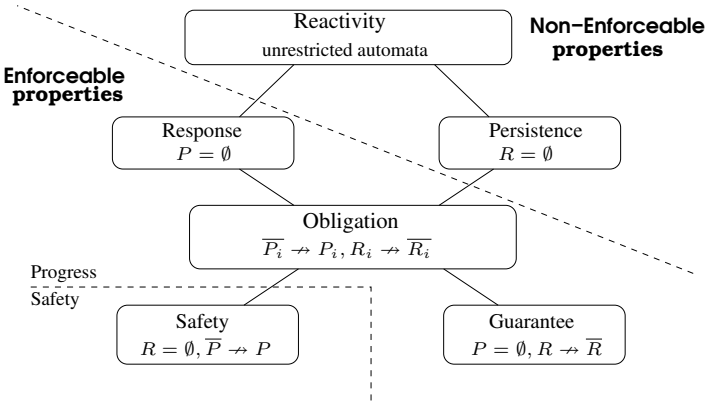


Figure 11: Enforceable properties and enforcement mechanisms wrt. the Safety-Progress classification of properties

Computability power of enforcement mechanisms. Hamlen, Morisett, and Schneider proposed [Ham06, HMS06] a classification of enforceable properties with the regard of a program as a Turing machine. Properties are classified according to the modification the enforcement mechanism can perform on the underlying program. Notably each employed mechanism corresponds to a certain 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 [HMS06]). However we believe that the proposed mechanism can be appointed 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 the more enforcement capability to our mechanism without any modification of the underlying program.

8 Conclusion and perspectives

Conclusion. In this paper our purpose was to extend previous works 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 wrt. 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 wrt. 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 security 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 the case 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 *controlable* 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 of interest 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 to define other relations between the input and the output sequences; and thus define other enforcement primitives so as 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 [KLM+97] as an underlying technique. Taking, as input, an e-property Π specified by a Streett automaton $\mathcal{A}_\Pi$, 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 plateform to investigate the impact of the aforementioned practical constraints. Also, we are currently studying alternative rewriting techniques (non based on aspects) to replace the tool for monitor integration in the underlying program (such as BCEL [The08] technology, or dynamic binary code insertion [NS07]). The benefits would be to perform runtime enforcement from binary versions of the target program.

References

[AS85] Bowen Alpern and Fred B. Schneider. Defining liveness. *Inf. Process. Lett.*, 21(4):181–185, 1985. 1, 3.1

[BLW09] Lujo Bauer, Jay Ligatti, and David Walker. Composing expressive runtime security policies. *ACM Trans. Softw. Eng. Methodol.*, 18(3), 2009. 7

[CMP92a] Edward Chang, Zohar Manna, and Amir Pnueli. The safety-progress classification. Technical report, Stanford University, Dept. of Computer Science, 1992. 3.2, 3.2, 3.1, 3.3, 3.3, 6.1.3, A.1

[CMP92b] Edward Y. Chang, Zohar Manna, and Amir Pnueli. Characterization of temporal property classes. In *Automata, Languages and Programming*, pages 474–486, 1992. 1, 3

[FFM08] Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier. Synthesizing enforcement monitors wrt. the safety-progress classification of properties. In R. Sekar and Arun K. Pujari, editors, *ICISS*, volume 5352 of *Lecture Notes in Computer Science*, pages 41–55, 2008. 1

[FFM09] Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier. Runtime Verification of Safety-Progress Properties. In Saddek Bensalem and Doron Peled, editors, *RV*, volume 5779 of *Lecture Notes in Computer Science*, pages 40–59. Springer, 2009. 4.2, 6.2.2

[Fon04] Philip W. L. Fong. Access control by tracking shallow execution history. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, pages 43–55. IEEE Computer Society Press, 2004. 1, 7

[Ham06] Kevin W. Hamlen. *Security Policy Enforcement By Automated Program-Rewriting*. PhD thesis, Cornell University, 2006. 7

[HG08] Klaus Havelund and Allen Goldberg. Verify your runs. *Verified Software: Theories, Tools, Experiments: First IFIP TC 2/WG 2.3 Conference, VSTTE 2005, Zurich, Switzerland, October 10-13, 2005, Revised Selected Papers and Discussions*, pages 374–383, 2008. 1

[HMS06] Kevin W. Hamlen, Greg Morrisett, and Fred B. Schneider. Computability classes for enforcement mechanisms. *ACM Trans. Program. Lang. Syst.*, 28(1):175–205, 2006. 1, 7, 7, 7

[KLM+97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. *Aspect-oriented programming*. Springer-Verlag, 1997. 8

[Lam77] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Softw. Eng.*, 3(2):125–143, 1977. 1, 3.1

[LBW05] Jay Ligatti, Lujo Bauer, and David Walker. Enforcing non-safety security policies with program monitors. In *ESORICS*, pages 355–373, 2005. 1, 7

[LBW09] Jay Ligatti, Lujo Bauer, and David Walker. Run-time enforcement of nonsafety policies. *ACM Trans. Inf. Syst. Secur.*, 12(3), 2009. 1, 5, 7

[Lig06] Jarred Adam Ligatti. *Policy Enforcement via Program Monitoring*. PhD thesis, Princeton University, June 2006. 7

[LS09] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *J. Log. Algebr. Program.*, 78(5):293–303, 2009. 1

[MP87] Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 205–205, New York, NY, USA, 1987. ACM. 1, 3, 3.1, 3.1, 3.2, 4

[NS07] Nicholas Nethercote and Julian Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. *SIGPLAN Not.*, 42(6):89–100, 2007. 8

[Sch00] Fred B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, 2000. 1, 1, 7

[Str81] Robert S. Streett. Propositional dynamic logic of looping and converse. In *STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 375–383, New York, NY, USA, 1981. ACM. 3.2

[The08] The Apache Jakarta Project. Byte Code Engineering Library. http://jakarta.apache.org/bcel/, 2008. 8

[Vis00] Mahesh Viswanathan. *Foundations for the run-time analysis of software systems*. PhD thesis, University of Pennsylvania, Philadelphia, PA, USA, 2000. Supervisor-Sampath Kannan and Supervisor-Insup Lee. 1, 7

A Proofs

A.1 Proof of Lemma 3.2

This proof is inspired from the proof done in [CMP92a]. An obligation *e*-property is defined as a boolean combination of safety and guarantee *e*-properties. Moreover, any boolean combination of safety and guarantee *e*-properties can be brought, using the distributive rule, to the following form:

$$\bigcap_{i=1}^{n} \left(\Pi_0^i \cup \cdots \cup \Pi_{k_i-1}^i \cup \Pi_{k_i}^i \cup \cdots \cup \Pi_{m_i-1}^i\right)$$

where $\Pi_0^i, \ldots, \Pi_{k_i-1}^i$ are safety properties, and $\Pi_{k_i}^i, \ldots, \Pi_{m_i-1}^i$ are guarantee properties. Using the closure of safety and guarantee properties under union, $\Pi_0^i, \ldots, \Pi_{k_i-1}^i$ (resp. $\Pi_{k_i}^i \cup \cdots \cup \Pi_{m_i-1}^i$) can be

replaced by a single safety (resp. guarantee) e-property Π^i_S (resp. Π^i_G). Thus, for any obligation e-property, there exists $n \in \mathbb{N}$ s.t. this e-property can be written as a n-obligation:

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

A.2 Proof of Property 4.2

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

Induction basis. For the induction basis $|\sigma| = 1$; we have $\sigma = a$ with $a \in \Sigma$. Using the definition of evolution of configurations (Def. 4.4), we have $\exists q \in Q^{\mathcal{A}_\downarrow}, ({q_{\text{init}}}^{\mathcal{A}_\downarrow}, \sigma\cdot\sigma', \epsilon) \overset{o}{\Longrightarrow}_{\mathcal{A}_\downarrow} (q, \sigma', m)$ with $\alpha(\sigma, \epsilon) = (o, m)$ and ${q_{\text{init}}}^{\mathcal{A}_\downarrow} \overset{\sigma/\alpha}{\longrightarrow} q$.

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

Induction step. Let us consider that the property is verified for every execution sequences of length n and consider an execution sequence $\sigma \cdot a$ of length $n + 1$, where $a \in \Sigma$. By reading σ, $\mathcal{A}_\downarrow$ enters a state $q \in Q^{\mathcal{A}_\downarrow}$, produces an output o, and has m in its memory. More formally, we have $\exists q \in Q^{\mathcal{A}_\downarrow}, ({q_{\text{init}}}^{\mathcal{A}_\downarrow}, \sigma \cdot a \cdot \sigma', \epsilon) \overset{o}{\Longrightarrow}_{\mathcal{A}_\downarrow} (q, a \cdot \sigma', m)$. Moreover, the induction hypothesis gives us: $(q \in Halt^{\mathcal{A}_\downarrow} \wedge o \preceq \sigma) \vee (q \notin Halt^{\mathcal{A}_\downarrow} \wedge \sigma = o \cdot m)$. As $\mathcal{A}_\downarrow$ is complete wrt. $Q^{\mathcal{A}_\downarrow} \times \Sigma$ (definition of EMs), $\exists \alpha \in Ops, \exists q' \in Q^{\mathcal{A}_\downarrow}, q \overset{a/\alpha}{\longrightarrow}_{\mathcal{A}_\downarrow} q'$. So, $\exists o', m' \in \Sigma^*, (q, a \cdot \sigma', m) \overset{o'}{\hookrightarrow}_{\mathcal{A}_\downarrow} (q', \sigma', m')$ with $\alpha(a, m) = (o', m')$. Which results in $({q_{\text{init}}}^{\mathcal{A}_\downarrow}, \sigma \cdot a \cdot \sigma', \epsilon) \overset{o\cdot o'}{\Longrightarrow}_{\mathcal{A}_\downarrow} (q', \sigma', m')$ and $(q \in Halt^{\mathcal{A}_\downarrow} \wedge o \preceq \sigma) \vee (q \notin Halt^{\mathcal{A}_\downarrow} \wedge \sigma = o \cdot m)$. We want to show that $(q' \in Halt^{\mathcal{A}_\downarrow} \wedge o \cdot o' \preceq \sigma \cdot a) \vee (q' \notin Halt^{\mathcal{A}_\downarrow} \wedge \sigma \cdot a = o \cdot o' \cdot m')$. Let us treat the three cases for the enforcement operation α.

- Case $\alpha = halt$. We have $\alpha(a, m) = (\epsilon, m)$. So $o' = \epsilon$ and $m = m'$. And we have also, according to the definition of EMs (Def. 4.2), $q' \in Halt^{\mathcal{A}_\downarrow}$. Then, we apply the induction hypothesis with σ, and depending on the membership of q in $Halt^{\mathcal{A}_\downarrow}$. If $q \in Halt^{\mathcal{A}_\downarrow}$, $o \preceq \sigma \Rightarrow o \cdot \epsilon \preceq \sigma \cdot a$. Else ($q \notin Halt^{\mathcal{A}_\downarrow}$), we have $o \cdot \epsilon \preceq \sigma \cdot a$.
- Case $\alpha = store$. We have $q \notin Halt^{\mathcal{A}_\downarrow}$, and $\alpha(a, m) = (\epsilon, m \cdot a)$, so $o' = \epsilon$ and $m' = m \cdot a$. By induction hypothesis, $q' \notin Halt^{\mathcal{A}_\downarrow}$ (Def. 4.2) 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 $q \notin Halt^{\mathcal{A}_\downarrow}$, and $\alpha(a, m) = (m.a, \epsilon)$. Then $o' = m \cdot a$ and $m' = \epsilon$. By induction hypothesis, we have necessarily $q' \notin Halt^{\mathcal{A}_\downarrow}$ (Def. 4.2), 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 Negation transformation

Proof. Let the e-property Π be (ϕ, φ), with $\phi \subseteq \Sigma^*$ and $\varphi \subseteq \Sigma^\omega$. Let us note $\overline{\mathcal{A}_{\downarrow\Pi}} = \text{Negation}(\mathcal{A}_{\downarrow\Pi})$, and $\Rightarrow$ the multistep derivation relation defined over configurations of $\overline{\mathcal{A}_{\downarrow\Pi}}$ and $\longrightarrow$. Also, since $Q^{\mathcal{A}_{\downarrow\Pi}} = Q^{\overline{\mathcal{A}_{\downarrow\Pi}}}$, 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 $Enf(\overline{\mathcal{A}_{\downarrow\Pi}}, \overline{\Pi}, \mathcal{P}_\Sigma)$, meaning that, for all $\sigma \in Exec(\mathcal{P}_\Sigma)$, we have to prove that $\exists o \in \Sigma^\infty$ s.t.

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

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

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

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

The proof is in two stages. The first one is for finite sequences. The second one is for infinite sequences.

Finite sequences. The proof for finite sequences is done by induction on $|\sigma|$.

Induction basis. For the induction basis $|\sigma| = 0$; we have $\sigma = \epsilon$, then we have easily (17) as $\epsilon \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \epsilon$. Moreover, $Pref_{\prec}(\phi, \epsilon) = \emptyset$, which gives (19).

Inductive step. Let $n \in \mathbb{N}$ and suppose that for all sequences σ s.t. $|\sigma| = n$, we have the existence of an output $o \in \Sigma^*$ s.t. the constraints (17), (18), (19) and (20) hold. Now consider $a \in \Sigma$, 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 want to prove that there exists a new output s.t. the same constraints hold.

As $\sigma \Downarrow_{\overline{\mathcal{A}_{\downarrow\Pi}}} o$ (induction hypothesis), there exists a configuration $(q, \epsilon, m) \in Q \times \Sigma^* \times \Sigma^*$ such that $(q_{\text{init}}, \sigma, \epsilon) \stackrel{o}{\Rightarrow} (q, \epsilon, m)$. Which implies that $(q_{\text{init}}, \sigma \cdot a, \epsilon) \stackrel{o}{\Rightarrow} (q, a, m)$. That is, after reading σ, $\overline{\mathcal{A}_{\downarrow\Pi}}$ 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 $(q, a, m) \stackrel{o'}{\hookrightarrow} (q', \epsilon, m')$ with $\alpha(a, m) = (o', m'), \alpha \in Ops$. The reading of $\sigma \cdot a$ on $\mathcal{A}_{\downarrow\Pi}$, induces a similar evolution of configurations:

$$(q_{\text{init}}, \sigma \cdot a, \epsilon) \stackrel{o}{\Rightarrow} (q, a, m) \stackrel{o'}{\hookrightarrow} (q', \epsilon, m')$$
$$(q_{\text{init}}, \sigma \cdot a, \epsilon) \stackrel{p}{\Rightarrow}_{\mathcal{A}_{\downarrow\Pi}} (q, a, n) \stackrel{p'}{\hookrightarrow}_{\mathcal{A}_{\downarrow\Pi}} (q', \epsilon, n'),$$

with:

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

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

- The first case is $\phi(\sigma \cdot a)$. As $Enf(\Pi, \mathcal{A}_{\downarrow\Pi}, \mathcal{P}_\Sigma)$, $\mathcal{A}_{\downarrow\Pi}$ produces $\sigma \cdot a$, that is $\sigma \cdot a \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \sigma \cdot a$. Necessarily, $\alpha' \in \{dump, \mathit{off}\}$. It corresponds to an operation $\alpha \in \{store, halt\}$ on $\overline{\mathcal{A}_{\downarrow\Pi}}$. Now we distinguish according to $\phi(\sigma)$ or not.
  - If $\phi(\sigma)$, using the induction hypothesis ($|\sigma| = n$), we have either $o = \epsilon$ (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 have also $Pref_{\prec}(\overline{\phi}, \sigma \cdot a) = \emptyset$. The output of $\overline{\mathcal{A}_{\downarrow\Pi}}$ is still ϵ, *i.e.*, $o \cdot o' = \epsilon$ (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 $o = Max(Pref_{\prec}(\overline{\phi}, \sigma \cdot a))$ (20).
  - If $\neg\phi(\sigma)$, *i.e.*, $\overline{\phi}(\sigma)$. Using the induction hypothesis, we have that $\sigma \Downarrow_{\overline{\mathcal{A}_{\downarrow\Pi}}} o$ with $\sigma = o$. Then $\sigma = Max(Pref_{\prec}(\overline{\phi}, \sigma))$ since $\overline{\phi}(\sigma)$. We obtain also (20).
- The second case is $\overline{\phi}(\sigma \cdot a)$. We have either $Pref_{\prec}(\phi, \sigma \cdot a) \neq \emptyset$ or $Pref_{\prec}(\phi, \sigma \cdot a) = \emptyset$.
  - If $Pref_{\prec}(\phi, \sigma \cdot a) \neq \emptyset$, then as $Enf(\Pi, \mathcal{A}_{\downarrow\Pi}, \mathcal{P}_\Sigma)$, we have $p = Max(Pref_{\prec}(\phi, \sigma \cdot a))$. We know that $p \prec \sigma \cdot a$ (as by hypothesis $\overline{\phi}(\sigma \cdot a)$). It follows that $\alpha' \in \{store, halt\}$. As a consequence $\alpha \in \{dump, \mathit{off}\}$ and $\sigma \cdot a \Downarrow_{\overline{\mathcal{A}_{\downarrow\Pi}}} \sigma \cdot a$, and by hypothesis, we have $\overline{\phi}(\sigma \cdot a)$. We have (17) and (18).
  - If $Pref_{\prec}(\phi, \sigma \cdot a) = \emptyset$. Necessarily, $\alpha' \in \{store, halt\}$ and $\alpha \in \{dump, \mathit{off}\}$. Thus $\sigma \cdot a \Downarrow_{\overline{\mathcal{A}_{\downarrow\Pi}}} \sigma \cdot a$.

Infinite sequences. We now deal with the proof for infinite sequences $\sigma \in \Sigma^\omega$. In the following we distinguish according to the class of the property 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 $Enf(\Pi, \mathcal{A}_{\downarrow\Pi}, \mathcal{P}_\Sigma)$, we have that $\sigma \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \sigma$. Moreover as Π is a safety *e*-property, all prefixes of σ satisfy ϕ (Property 3.1), that is $\forall \sigma' \prec \sigma, \phi(\sigma')$, and consequently $\sigma' \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \sigma'$. It follows (Prop. 4.3) that the sequence of enforcement operations on $\mathcal{A}_{\downarrow\Pi}$ belongs to $(dump)^\omega + dump^* \cdot off^\omega$. Then using the definition of Negation, we find that the sequence of enforcement operations on $\overline{\mathcal{A}_{\downarrow\Pi}}$ is $(store + halt)^\omega$. It follows that $\sigma \Downarrow_{\overline{\mathcal{A}_{\downarrow\Pi}}} \epsilon$, *i.e.*, (17). As $Pref_\prec(\overline{\phi}, \sigma) = \emptyset$, we obtain (19).
  - $\overline{\varphi(\sigma)}$. As $Enf(\Pi, \mathcal{A}_{\downarrow\Pi}, \mathcal{P}_\Sigma)$, we have that either $Pref_\prec(\phi, \sigma) = \emptyset \wedge o = \epsilon$ or $\exists o \in \Sigma^* \cdot o = Max(Pref_\prec(\phi, \sigma))$.
    - Let deal first with the case in which $Pref_\prec(\phi, \sigma) = \emptyset$. In this case we have $\forall \sigma' \in \Sigma^*, \sigma' \prec \sigma, \neg\phi(\sigma')$. It follows that the sequence of enforcement operations on $\mathcal{A}_{\downarrow\Pi}$ belongs to $halt^\omega$. Using the definition of Negation, the sequence of enforcement operations belongs to $\overline{\mathcal{A}_{\downarrow\Pi}}$ is off^ω. It follows that $\sigma \Downarrow_{\overline{\mathcal{A}_{\downarrow\Pi}}} \sigma$. We obtain (17).
    - Now $Pref_\prec(\phi, \sigma) \neq \emptyset$. Let $n = |o|$. As Π is a safety *e*-property, we have that $\forall i \leq n, \phi(\sigma_{\ldots i-1}) \wedge \forall i > n, \neg\phi(\sigma_{\ldots i})$. Then using the property Prop. 4.3, we can find the sequence of enforcement operations performed by $\mathcal{A}_{\downarrow\Pi}$: $(dump)^n \cdot halt^\omega$. On $\overline{\mathcal{A}_{\downarrow\Pi}}$, using the definition of the transformation Negation, the sequence of enforcement operations becomes $store^n \cdot off^\omega$. It follows that $\sigma \Downarrow_{\overline{\mathcal{A}_{\downarrow\Pi}}} \sigma$ (17). Then, $\overline{\varphi(\sigma)}$ and $\sigma = \sigma$ ensure (18).

- Π *is a guarantee e-property.* We have two cases, depending on $\varphi(\sigma)$ or not.
  - $\varphi(\sigma)$. As $Enf(\Pi, \mathcal{A}_{\downarrow\Pi}, \mathcal{P}_\Sigma)$, we have that $\sigma \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \sigma$. Moreover as Π is a guarantee *e*-property, there exists a prefix σ' of σ s.t. $\forall \sigma'' \in \Sigma^*, \sigma' \preceq \sigma'' \Rightarrow \phi(\sigma'') \wedge \forall \sigma'' \prec \sigma', \neg\phi(\sigma'')$. Let us note $n = |\sigma'|$. Consequently, as Π is enforced by $\mathcal{A}_{\downarrow\Pi}$, we have $\forall \sigma'' \in \Sigma^*, \sigma' \preceq \sigma'' \Rightarrow \sigma'' \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \sigma'' \wedge \forall \sigma'' \prec \sigma', \sigma'' \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \epsilon$. It follows that the sequence of enforcement operations on $\mathcal{A}_{\downarrow\Pi}$ is $store^{n-1} \cdot off^\omega$. Then, using the definition of the transformation Negation, we find that the sequence of enforcement operations on $\overline{\mathcal{A}_{\downarrow\Pi}}$ is $dump^{n-1} \cdot halt^\omega$. It follows that $\sigma \Downarrow_{\overline{\mathcal{A}_{\downarrow\Pi}}} \sigma'$ (17). Moreover we have seen that $\phi(\sigma')$, we have (20).
  - $\neg\varphi(\sigma)$. Knowing that Π is a guarantee *e*-property, $\neg\varphi(\sigma)$ implies that there is no prefix of σ satisfying ϕ. As $Enf(\Pi, \mathcal{A}_{\downarrow\Pi}, \mathcal{P}_\Sigma)$, we have that $\forall \sigma' \prec \sigma, \sigma \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \epsilon$. The sequence of enforcement operations performed by $\mathcal{A}_{\downarrow\Pi}$ belongs to $store^* \cdot halt^\omega$. Using the definition of the Negation transformation, the sequence of enforcement operations on $\overline{\mathcal{A}_{\downarrow\Pi}}$ belongs to $dump^* \cdot off^\omega$. It follows that $\sigma \Downarrow_{\overline{\mathcal{A}_{\downarrow\Pi}}} \sigma$. We have (17) and (18).

■

A.4 Correctness of the TransGuarantee transformation

We note $\mathcal{A}_\Pi = (Q^{\mathcal{A}_\Pi}, {q_{\text{init}}}^{\mathcal{A}_\Pi}, \Sigma, \longrightarrow_{\mathcal{A}_\Pi}, (R, \emptyset))$. Let us consider an execution sequence of the program $\sigma \in Exec(\mathcal{P}_\Sigma)$. We study the effect of the submission of σ to $\mathcal{A}_{\downarrow\Pi}$. We will, as previously, associate the execution of σ on $\mathcal{A}_\Pi$ to the execution of σ on $\mathcal{A}_{\downarrow\Pi}$. The execution of σ on $\mathcal{A}_\Pi$ 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 $\mathcal{A}_{\downarrow\Pi}$ with $q_0 = {q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi}}$. We distinguish again depending on whether the sequence σ satisfies the property Π or not.

- *The first case is* $\Pi(\sigma)$. We know that the automaton $\mathcal{A}_\Pi$ accepts σ, let distinguish whether σ is finite or not.

* If $\sigma \in \Sigma^*$, then $\phi(\sigma)$. Let $n = |\sigma|$. As σ is accepted by $\mathcal{A}_\Pi$, and according to the acceptance criterion (Def. 3.2), we have that by reading σ, there exists a state $q \in R$ reachable from ${q_{\text{init}}}^{\mathcal{A}_\Pi}$, from which we stay in R-states (we have $P = \emptyset$ and no transition from R-states to $\overline{R}$-states since $\mathcal{A}_\Pi$ is a guarantee-automaton).

  If $\sigma = \epsilon$, then we have easily (13) as $\epsilon \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \epsilon$. Moreover, $Pref_{\prec}(\phi, \epsilon) = \emptyset$, which gives (15).

  Else ($\sigma \neq \epsilon$), according to the constraints of the transition relation of a guarantee-automaton, the run and the trace of σ on $\mathcal{A}_\Pi$ are such that there exists $0 \leq k \leq n$ such that $\forall i \leq k \leq n, q_i \in \overline{R} \wedge \forall n \geq i > k, q_i \in R$.

  According to (TGUAR1) and (TGUAR2), the trace of σ on $\mathcal{A}_{\downarrow\Pi}$ is such that $\forall i < k, \alpha_i = store \wedge \forall i \geq k, \alpha_i = dump$.

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

  * $({q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi}}, \sigma, \epsilon) \overset{\epsilon}{\hookrightarrow} (q_1, \sigma_1\ldots, \sigma_0) \cdots \overset{\epsilon}{\hookrightarrow} (q_{k-1}, \sigma_{k-1}\ldots, \sigma_{\ldots k-2}) \overset{\epsilon}{\hookrightarrow} (q_k, \sigma_{k}\ldots, \sigma_{\ldots k-1})$ as $\forall i < k, store(\sigma_i, \sigma_{\ldots i-1}) = (\epsilon, \sigma_{\ldots i})$;
  * $(q_k, \sigma_{k}\ldots, \sigma_{\ldots k-1}) \overset{\sigma_{\ldots k}}{\hookrightarrow} (q_{k+1}, \sigma_{k+1}\ldots, \epsilon) \cdots \overset{\sigma_{n-2}}{\hookrightarrow} (q_{n-1}, \sigma_{n-1}, \epsilon) \overset{\sigma_{n-1}}{\hookrightarrow} (q_n, \epsilon, \epsilon)$ as $dump(\sigma_k, \sigma_{\ldots k-1}) = (\sigma_{\ldots k}, \epsilon)$ and $\forall i > k$, $dump(\sigma_i, \epsilon) = (\sigma_i, \epsilon)$.

  By deduction, using the multistep derivations, we have $({q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi}}, \sigma, \epsilon) \overset{\epsilon}{\Rightarrow} (q_k, \sigma_{k}\ldots, \sigma_{\ldots k-1})$ and $(q_k, \sigma_{k}\ldots, \sigma_{\ldots k-1}) \overset{\sigma}{\Rightarrow} (q_n, \epsilon, \epsilon)$. That is $\sigma_{\ldots k-1} \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \epsilon$, and then $\sigma_{\ldots k} \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \sigma_k$, and at last $\sigma \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \sigma$. Which ensures (13). Besides, according to the acceptance criterion of e-properties, we have $\phi(\sigma)$, wich permits us to deduce (14), as $\sigma = \sigma$.

* If $\sigma \in \Sigma^\omega$, and then $\varphi(\sigma)$. Using Def. 3.2 and the definition of a guarantee-automaton, we have $vinf(\sigma) \cap R \neq \emptyset$. Meaning that there exists a state of R which is visited infinitely often. As there is no transition from states in R to states in $\overline{R}$, the states of $\overline{R}$ are visited at most a finite number of times. That is, we can find an index k and two states q, q' s.t. $q \xrightarrow{\sigma_k}_{\mathcal{A}_{\downarrow\Pi}} q'$ with $q \notin R \wedge q' \in R$. The run of σ on $\mathcal{A}_\Pi$ is s.t. $\forall i < k, q_i \notin R \wedge \forall i \geq k, q_i \in R$. Which by a similar reasoning leads us to find the trace of σ on $\mathcal{A}_{\downarrow\Pi}$. The sequence of enforcement operations is then $(store)^{k-1} \cdot (dump)^\omega$. It follows that $\sigma \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \sigma$. We have then (13) and (14).

- *The second case is* $\neg\Pi(\sigma)$. The sequence σ is not accepted by $\mathcal{A}_\Pi$. As $\neg\varphi(\sigma)$ and Π is guarantee e-property, we have $Pref_{\prec}(\phi, \sigma) = \emptyset$. Indeed, as Π is a guarantee e-property, the existence of a prefix of σ satisfying φ would have implied that $\varphi(\sigma)$.

  According to the constraints on a guarantee-automaton, $\mathcal{A}_\Pi$ starts and ends in $\overline{R}$-states and stays in (there exists no transition from $q \in R$ to $q' \notin R$). We deduce that the execution trace of σ on $\mathcal{A}_\Pi$ is s.t. $\forall i > 0, q_i \notin R$. Using the definition of the TransGuarantee transformation we can find $trace(\sigma, \mathcal{A}_{\downarrow\Pi})$. Then, the enforcement operation performed by $\mathcal{A}_{\downarrow\Pi}$ is always $halt$ or $store$. There exists two possible shapes for the sequence of the α_i: either $\forall i > 0, \alpha_i = store$, or $\exists k > 0, (\forall i \leq k, \alpha_i = store \wedge \forall k > i, \alpha_i = halt)$. In both cases, using definitions of $store$ and $halt$, we can find easily that $\sigma \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \epsilon$ (13).

  Thus, as $\sigma \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \epsilon$, we have (13) and (15).

A.5 Correctness of the TransObligation transformation

In order to prove the property, we rely on showing that: the EM obtained by the application of TransSafety, TransGuarantee, and the *Intersection* transformation (this EM is correct by construction); and the EM obtained by the direct application of TransObligation, are equal.

To prove the theorem, we show that the following property holds: given $\mathcal{A}_\Pi$ recognizing Π

$$\forall k \geq 1, \big(\Pi \in \text{Obligation}_k(\Sigma) \wedge \mathcal{A}_{\downarrow\Pi} = \text{TransObligation}(\mathcal{A}_\Pi)\big) \Rightarrow Enf(\mathcal{A}_{\downarrow\Pi}, \Pi, \mathcal{P}_\Sigma)$$

To do so, we perform an induction on k where the property Π is a k-obligation.

- *Induction basis.* For the induction basis, $k = 1$, Π is a simple obligation. We note $\mathcal{A}_\Pi = (Q^{\mathcal{A}_\Pi}, {q_{\text{init}}}^{\mathcal{A}_\Pi}, \Sigma, \longrightarrow_{\mathcal{A}_\Pi}, \{(R, P)\})$. Let $\sigma \in \Sigma^\infty$. Let study the effect of the submission of σ to $\mathcal{A}_{\downarrow\varphi}$. There are two cases depending on $\Pi(\sigma)$.

  - *The first case is* $\Pi(\sigma)$. In this case, the sequence σ is accepted by $\mathcal{A}_\Pi$. According to the constraints on obligation-automata, the execution of σ on $\mathcal{A}_\Pi$ produces an execution trace $(q_0, \sigma_0, q_1) \cdot (q_1, \sigma_1, q_2) \cdots (q_i, \sigma_i, q_{i+1}) \cdots$ of σ such that $q_0 = {q_{\text{init}}}^{\mathcal{A}_\Pi}$ and
    - either each state q_i is in P,
    - or there exists $0 \leq k$ s.t. $\forall i < k, q_i \in \overline{R} \wedge \forall i \geq k, q_i \in R$

    (there is no transition from $q \in \overline{P}$ to $q' \in P$ and from $q \in R$ to $q' \in \overline{R}$).

    In both cases, we can follow the previous reasoning for safety and guarantee properties to obtain the trace on $\mathcal{A}_{\downarrow\Pi}$. Note that if the two previous conditions hold simultaneously, the shape of the execution trace is the same.
  - *The second case is* $\neg\Pi(\sigma)$. Similarly in this case, it amounts to the cases where Π is either a safety or a guarantee property.

- *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 $\mathcal{A}_\Pi$, then the EM $\mathcal{A}_{\downarrow\Pi} = \text{TransObligation}(\mathcal{A}_\Pi)$ enforces Π, that is, we have $\mathit{Enf}(\mathcal{A}_{\downarrow\Pi}, \Pi, \mathcal{P}_\Sigma)$.

  Now consider a (k+1)-obligation Π, $\mathcal{A}_\Pi$ a recognizing (k+1)-obligation automaton, and $\mathcal{A}_{\downarrow\Pi} = \text{TransObligation}(\mathcal{A}_\Pi)$. 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 simple obligation properties (3.2). The expression of Π can be rewritten as $\Pi = (\bigcap_{i=1}^{k} \Pi_i) \cap \Pi_{k+1}$. Using Lemma 3.1, one can find two recognizing automata $\mathcal{A}_{\Pi/\{1,\ldots,k\}}$ recognizing $\bigcap_{i=1}^{k} \Pi_i$ and $\mathcal{A}_{\Pi/\{k+1\}}$ recognizing Π_{k+1}. Now using the induction hypothesis, we can apply $\text{TransObligation}()$ to these two automata to obtain two EMs $\mathcal{A}_{\downarrow\Pi/\{1,\ldots,k\}}$ enforcing $\bigcap_{i=1}^{k} \Pi_i$ and $\mathcal{A}_{\downarrow\Pi/\{k+1\}}$ enforcing Π_{k+1}. Using the Intersection construction (Def. 5.2), one can obtain the EM $\mathcal{A}_{\downarrow\Pi}{}' = \text{Intersection}(\mathcal{A}_{\downarrow\Pi/\{1,\ldots,k\}}, \mathcal{A}_{\downarrow\Pi/\{k+1\}})$ enforcing (Theorem 5.1) $(\bigcap_{i=1}^{k} \Pi_i) \cap \Pi_{k+1} = \bigcap_{i=1}^{k+1} \Pi_i$, that is Π.

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

  - For $\mathcal{A}_{\downarrow\Pi}$, according to Def. 6.3 of TransObligation:
    - $Q^{\mathcal{A}_{\downarrow\Pi}} = Q^{\mathcal{A}_\Pi}$,
    - ${q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi}} = {q_{\text{init}}}^{\mathcal{A}_\Pi}$,
    - and $\forall a \in \Sigma, q \xrightarrow{a/\alpha}_{\mathcal{A}_{\downarrow\Pi}} q'$ where $\alpha = \sqcap_{i=1}^{k+1} \sqcup (\{\beta_i, \gamma_i\})$.
  - For $\mathcal{A}_{\downarrow\Pi}{}'$, according to Def. 5.2 of the intersection between EMs:
    - $Q^{\mathcal{A}_{\downarrow\Pi}{}'} = Q^{\mathcal{A}_{\downarrow\Pi/\{k+1\}}} \times Q^{\mathcal{A}_{\downarrow\Pi/\{1,\ldots,k\}}} = Q^{\mathcal{A}_\Pi} \times Q^{\mathcal{A}_\Pi}$,
    - ${q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi}{}'} = {q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi/\{k+1\}}} \times {q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi/\{1,\ldots,k\}}} = {q_{\text{init}}}^{\mathcal{A}_\Pi} \times {q_{\text{init}}}^{\mathcal{A}_\Pi}$,
    - and $\forall a \in \Sigma, q \xrightarrow{a/\alpha}_{\mathcal{A}_{\downarrow\Pi}{}'} q'$ where $\alpha = \sqcap_{i=1}^{k} \sqcup (\{\beta_i, \gamma_i\}) \sqcap (\sqcup(\{\beta_{k+1}, \gamma_{k+1})\)$, *i.e.*, $\alpha = \sqcap_{i=1}^{k+1} \sqcup (\{\beta_i, \gamma_i\})$.
  - where, $\forall i \in \{1, \ldots, k+1\}$:
    - β_i is
      - *off* if $q' \in P_i \wedge \mathit{Reach}_{\mathcal{A}_\Pi}(q') \cap \overline{P_i} = \emptyset$
      - *dump* if $q' \in P_i \wedge \mathit{Reach}_{\mathcal{A}_\Pi}(q') \cap \overline{P_i} \neq \emptyset$
      - *halt* if $q' \notin P_i$
    - γ_i is
      - *off* if $q' \in R_i$

· *halt* if $q' \notin R_i \wedge \nexists q'' \in R_i, q'' \in Reach_{\mathcal{A}_\Pi}(q')$
· *store* if $q' \notin R_i \wedge \exists q'' \in R_i, q'' \in Reach_{\mathcal{A}_\Pi}(q')$

That is we can exhibit a bijection relation between $\mathcal{A}_{\downarrow\Pi}{}'$ and $\mathcal{A}_{\downarrow\Pi}$: $\forall q \in Q^{\mathcal{A}_\Pi}$, the state q in $\mathcal{A}_{\downarrow\Pi}$ is in relation with the state (q,q) in $\mathcal{A}_{\downarrow\Pi}{}'$. Formally, between the two EMs $\mathcal{A}_{\downarrow\Pi}$ and $\mathcal{A}_{\downarrow\Pi}{}'$, there is a relation $\mathcal{R} \subseteq (Q^{\mathcal{A}_\Pi} \times (Q^{\mathcal{A}_\Pi} \times Q^{\mathcal{A}_\Pi}))$ defined by $\mathcal{R} = \{(q,(q,q)) \mid q \in Q^{\mathcal{A}_\Pi}\}$. 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 $\mathcal{A}_\Pi$ is correct. This concludes the proof for the TransObligation transformation and *Obligation* properties.

A.6 Correctness of the TransResponse transformation

The proof follows the same principle of the proof for the guarantee properties. 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 $\mathcal{A}_\Pi = (Q^{\mathcal{A}_\Pi}, {q_{\text{init}}}^{\mathcal{A}_\Pi}, \Sigma, \longrightarrow_{\mathcal{A}_\Pi}, (R,\emptyset))$. Let us consider an execution sequence of the program $\sigma \in Exec(\mathcal{P}_\Sigma)$. We study the effect of the submission of σ to $\mathcal{A}_{\downarrow\Pi}$. We will, as previously, associate the execution of σ on $\mathcal{A}_\Pi$ to the execution of σ on $\mathcal{A}_{\downarrow\Pi}$. The execution of σ on $\mathcal{A}_\Pi$ 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 $\mathcal{A}_{\downarrow\Pi}$ with $q_0 = {q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi}}$. We distinguish again depending on whether the sequence σ satisfies the property Π or not.

- The first case is $\Pi(\sigma)$. We know that the automaton $\mathcal{A}_\Pi$ accepts σ, let distinguish whether σ is finite or not.

  * $\sigma \in \Sigma^*$, and then $\phi(\sigma)$. Let $n = |\sigma|$. According to the acceptance criterion (Def. 3.2), the run of $\mathcal{A}_\Pi$ on σ ends in a R-state.
    If $\sigma = \epsilon$, then we have easily (13) as $\epsilon \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \epsilon$. Moreover, $Pref_{\prec}(\phi,\epsilon) = \emptyset$, which gives (15).
    Else ($\sigma \neq \epsilon$), according to the constraints of the transition relation of a response-automaton, the run and the trace of σ on $\mathcal{A}_\Pi$ are such that $q_n \in R$. According to (TGUAR1), the trace of σ on $\mathcal{A}_{\downarrow\Pi}$ is such that $\alpha_n = dump$. From the execution trace on $\mathcal{A}_{\downarrow\Pi}$ and the definition of the enforcement operations, we deduce the following derivations of configurations:

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

    with $o_0 \cdot o_1 \cdots o_{n-1} = \sigma$ since the last enforcement operation (α_{n-1}) is $dump$.
    By deduction, using the multistep derivations, we have $({q_{\text{init}}}^{\mathcal{A}_{\downarrow\Pi}}, \sigma, \epsilon) \overset{\sigma}{\Rightarrow} (q_n, \epsilon, \epsilon)$. That is $\sigma \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \sigma$. Which ensures (13). Besides, according to the acceptance criterion of e-properties, we have $\phi(\sigma)$, which permits to deduce (14), as $\sigma = \sigma$.

  * If $\sigma \in \Sigma^\omega$, then according to the definition of a response-automaton, we find using the acceptance criterion that $vinf(\sigma, \mathcal{A}_\Pi) \cap R \neq \emptyset$. In other words, there exists a state $q \in Q^{\mathcal{A}_\Pi} \cap R$ visited infinitely often. As there is no restriction on the transition function of $\mathcal{A}_\Pi$, the run of σ on $\mathcal{A}_\Pi$ contains some states of $\overline{R}$, some states of R, but visits q infinitely often. 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 $\mathcal{A}_\Pi$ verifies $\forall i \in \mathbb{N}, \exists j \in \mathbb{N}, j \geq i \wedge (q_{j-1}, \sigma_{j-1}, q_j) \in trace(\sigma, \mathcal{A}_\Pi) \wedge q_j \in R$. Then we deduce that the trace on the enforcement monitor $\mathcal{A}_{\downarrow\Pi}$ (using the definition of TransResponse, Def. 6.4) verifies the property: $\forall i \in \mathbb{N}, \exists j \in \mathbb{N}, j \geq i \wedge (q_{j-1}, \sigma_{j-1}/dump, q_j) \in trace(\sigma, \mathcal{A}_{\downarrow\Pi})$. That is: $\forall i \in \mathbb{N}, \exists j \in \mathbb{N}, j \geq i, \alpha_j = dump$. Thus we deduce that (using Prop. 4.1) $\sigma \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \sigma$, *i.e.*, (13). Moreover, we have (14) as $\varphi(\sigma) \wedge \sigma = \sigma$.

- The second case is $\neg\Pi(\sigma)$. The sequence σ is not accepted by $\mathcal{A}_\Pi$, 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, $\mathcal{A}_\Pi$ starts in a $\overline{R}$-state and stays in. We deduce that the execution trace of σ on $\mathcal{A}_\Pi$ is s.t. $\forall i > 0, q_i \notin R$. Using the definition of the $\mathrm{TransResponse}$ transformation we can find $trace(\sigma, \mathcal{A}_{\downarrow\Pi})$. Then, the enforcement operation performed by $\mathcal{A}_{\downarrow\Pi}$ is always $halt$ or $store$. That is $\sigma \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \epsilon$ (13). Then $Pref_{\prec}(\phi, \sigma) = \emptyset$ implies that $\forall \sigma' \prec \sigma, \neg\phi(\sigma)$. We have (15).
  * *Else (*$Pref_{\prec}(\phi, \sigma) \neq \emptyset$*).* There is at least one prefix of σ satisfying ϕ. Let us note σ_{good} the longest prefix of σ satisfying ϕ, *i.e.*, $\sigma_{good} = Max(Pref_{\prec}(\sigma, \phi))$. Let $k = |\sigma_{good}|$. Then the run and the trace of $\mathcal{A}_\Pi$ on σ is s.t. $q_{k-1} \in R \wedge \forall i \in [k, n-1], q_i \in \overline{R}$. According to the $\mathrm{TransResponse}$ transformation, the trace of σ on $\mathcal{A}_{\downarrow\Pi}$ is s.t. $\alpha_{k-1} = dump \wedge \forall i \in [k, n-1], \alpha_i \in \{store, halt\}$. From the execution trace on $\mathcal{A}_{\downarrow\Pi}$ and the definition of the enforcement operations, we deduce the following derivations of configurations:

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

$$(q_k, \sigma_{k\cdots}, \epsilon) \overset{\epsilon}{\hookrightarrow} (q_{k+1}, \sigma_{k+1\cdots}, m_{k+1}) \overset{\epsilon}{\hookrightarrow} \cdots \overset{\epsilon}{\hookrightarrow} (q_n, \epsilon, m_n)$$

  with $\sigma_{good} = \sigma_{\cdots(k-1)} = o_0 \cdot o_1 \cdots o_{k-1}$. Indeed we have $dump(\sigma_{k-1}, m_{k-1}) = (m_{k-1} \cdot \sigma_{k-1}, \epsilon)$ and $\forall i \geq k, \alpha_i \in \{store, halt\}$, α_i makes that $\mathcal{A}_{\downarrow\Pi}$ produces ϵ in output (for $k \leq i \leq n-1$). That is $\sigma_{\cdots k-1} \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \sigma_{\cdots k-1}$ and $\sigma \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \sigma_{\cdots k-1}$. Which ensures (13). Besides, according to the acceptance criterion of e-properties, we have $\neg\phi(\sigma)$, which permits us to deduce (16), 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 $vinf(\sigma, \mathcal{A}_\Pi) \cap R = \emptyset$. We deduce that there exists a natural number n such that the run of σ on $\mathcal{A}_\Pi$ is expressed as $run(\sigma, \mathcal{A}_\Pi) = q_0 \cdots q_n \cdots$ with $q_0 = {q_{\text{init}}}^{\mathcal{A}_\Pi} \wedge (\forall i \geq n, q_i \in \overline{R})$. 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 $k > n_{min}$, $\sigma_{\cdots k} \Downarrow_{\mathcal{A}_{\downarrow\Pi}} \sigma_{\cdots n_{min}}$ 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 $\phi\left(\sigma_{\cdots n_{min}} = Max(Pref_{\prec}(\phi, \sigma_{\cdots k}))\right)$.