Version submitted to and accepted at NFM 2016.
Camera ready version under preparation.

Compositional Runtime Enforcement

Srinivas Pinisetty[1], Stavros Tripakis[1,2]

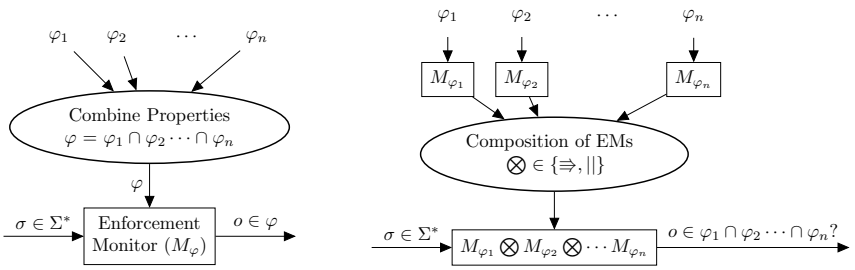
[1] Aalto University, Finland First.Last@aalto.fi
[2] University of California, Berkeley, USA

Abstract. Runtime enforcement is a methodology used to enforce that the output of a running system satisfies a desired property. Given a property, an enforcement monitor modifies an (untrusted) sequence of events into a sequence that complies to that property. In practice, we may have not one, but many properties to enforce. Moreover, new properties may arise as new capabilities are added to the system. It then becomes interesting to be able to build not a single, monolithic monitor that enforces all the properties, but rather several monitors, one for each property. The question is to what extent such monitors can be composed, and how. This is the topic of this paper. We study two monitor composition schemes, serial and parallel composition, and show that, while enforcement under these schemes is generally not compositional, it is for certain subclasses of regular properties.

1 Introduction

Runtime enforcement (RE) is a technique [14,5,11,12,13] to monitor the execution of a system at runtime and ensure its compliance against a set of formal requirements. An enforcement monitor (EM) is generally synthesized from a property expressed in a high-level formalism [14,5,11,12,13]. Similar to enforcement mechanisms in [13], we focus on online enforcement of regular properties defined as automata, where an EM is placed between an event emitter and an event receiver, operating at runtime. An EM takes as input a sequence of events σ (modeling an untrustworthy execution) and transforms it into a sequence of events o that complies with a given property φ that we want to enforce. The monitor is equipped with internal memory and is able to store some input events and release them later, after it receives some expected input.

For any complex system, we generally have several critical properties to enforce. Suppose that we want to enforce properties $\varphi_1, \varphi_2, \cdots \varphi_n$. An obvious approach is to take the conjunction of all these properties, $\varphi := \varphi_1 \wedge \cdots \wedge \varphi_n$, and synthesize an EM for the resulting property φ (illustrated in Figure 1a). This *monolithic* approach has several drawbacks. First and foremost, is not *modular*. As the functionality of the system evolves, additional properties may be added to the set of properties we want to enforce. With a monolithic approach, a new EM needs to be constructed from scratch every time a new property is added. In addition to problems of *performance* and *scalability*, this also has implications to system *security*. Indeed, in order to construct a monolithic EM for the conjunction $\varphi_1 \wedge \cdots \wedge \varphi_n$, *every* φ_i needs to be known. It may be the case, however, that some of these properties are unknown, or even secret. In that case, we would like to add new monitors that enforce new properties, without affecting the currently enforced properties, some of which may be unknown. We call this methodology *compositional runtime enforcement* (CRE).



(a) Monolithic approach. (b) Composing enforcement monitors.

Fig. 1: Monolithic and compositional runtime enforcement approaches.

CRE is particularly relevant in a security context. For instance, when an attack is detected, one may consider adding an additional property to the set of properties to be enforced. The additional property tries to prevent the detected attack. When a new property is added, it is natural to consider adding an additional layer on top of an existing one, i.e., a new enforcer on top of an existing one. This naturally leads to considering the composition of EMs. To continue this example, suppose that we already have an EM for some property φ_1, but φ_1 is secret and unknown. We are now given a new property φ_2, and we are asked to develop a system that enforces φ_2, without compromising φ_1. Therefore, the resulting system must enforce *both* φ_1 *and* φ_2. Since we don't know φ_1, we have no way of computing a monolithic EM for the conjunction $\varphi_1 \wedge \varphi_2$. Therefore, a compositional approach is necessary. We need to build an EM for φ_2, and somehow compose it with the existing EM for φ_1, so that the resulting system satisfies $\varphi_1 \wedge \varphi_2$.

How to compose two or more enforcement monitors? As illustrated in Figure 1b, we consider *serial* and *parallel* composition of EMs, denoted by $\Rrightarrow$ and $||$, respectively. Serial composition means that the output of one enforcer is fed as input to a next enforcer in a chain. Parallel composition means that all the EMs run in parallel and receive the same input, and that their outputs are *merged* somehow. Our results are as follows:

- We show that runtime enforcement is not compositional for general regular properties, neither w.r.t. serial nor parallel composition. This means that if EMs M_1 and M_2 enforce properties φ_1 and φ_2, respectively, neither $M_1 \Rrightarrow M_2$ nor $M_1||M_2$ generally enforce $\varphi_1 \wedge \varphi_2$.
- We show that compositionality holds for certain subclasses of regular properties, for instance when both φ_1 and φ_2 are safety (or co-safety) properties.
- We also investigate whether the order of serial composition of enforcers matters. By definition, the order does not matter in the case of parallel composition.
- Surprisingly, we show that in a serial composition setting $M_1 \Rrightarrow M_2$, where M_1 enforces φ_1 and M_2 enforces φ_2, using the *predictive* runtime enforcement methodology [13] for constructing M_2 does not have any advantage over standard RE. Predictive RE is generally useful when the input to an enforcer is known to satisfy a certain property [13]. In the case of $M_1 \Rrightarrow M_2$, the input to M_2 is known to satisfy φ_1 (since M_1 enforces that). Despite this, we show that using predictive RE to construct M_2 is equivalent to using standard RE.
- In all above cases we also investigate whether the final result produced by the composite enforcers is the same as what a monolithic enforcer would produce.

Outline. In Section 2 we introduce preliminaries and notation, and recall the runtime enforcement framework from previous work. In Section 3 we discuss the three enforcement approaches, namely, monolithic, using composition in series, and using composition in parallel. We show by example that RE is generally not compositional w.r.t. neither serial nor parallel composition. We also show that predictive RE does not help in a serial composition setting. In Section 4 we consider subclasses of regular properties, and show that the serial and parallel composition approaches work for these subclasses. Section 6 presents conclusions and future work.

2 Background

2.1 Preliminaries and Notation

Languages. A (finite) word over a finite alphabet Σ is a finite sequence $w = a_1 a_2 \cdots a_n$ of elements of Σ. The *length* of w is n and is noted $|w|$. The empty word over Σ is denoted by ϵ_Σ, or ϵ when clear from the context. The sets of *all words* and *all non-empty words* are denoted by Σ^* and Σ^+, respectively. A *language* or a *property* over Σ is any subset $\mathcal{L}$ of Σ^*.

The *concatenation* of two words w and w' is noted $w \cdot w'$. A word w' is a *prefix* of a word w, noted $w' \preccurlyeq w$, whenever there exists a word w'' such that $w = w' \cdot w''$, and $w' \prec w$ if additionally $w' \neq w$; conversely w is said to be an *extension* of w'.

A language $\mathcal{L}$ over Σ is *prefix-closed* if all prefixes of all words from $\mathcal{L}$ are also in $\mathcal{L}$: $\mathcal{L} = \{w \mid \exists w' \in \mathcal{L} : w \preccurlyeq w'\}$. Similarly, a language $\mathcal{L}$ over Σ is *extension-closed* if all extensions of all words from $\mathcal{L}$ are in $\mathcal{L}$: $\mathcal{L} = \{w \mid \exists w' \in \mathcal{L} : w' \preccurlyeq w\}$.

Given an n-tuple of symbols $e = (e_1, \ldots, e_n)$, for $i \in [1, n]$, $\Pi_i(e)$ is the projection of e on its i-th element ($\Pi_i(e) \stackrel{\text{def}}{=} e_i$).

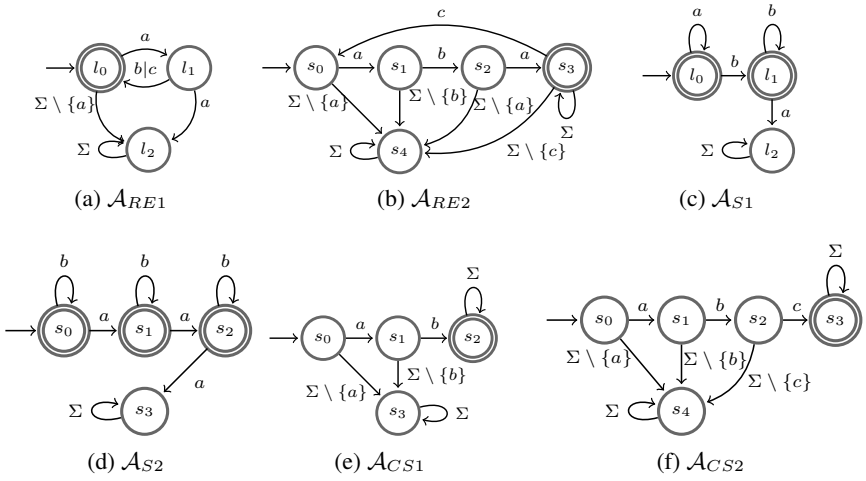
Deterministic and complete automata. A *deterministic and complete automaton* $\mathcal{A} = (Q, q_0, \Sigma, \delta, F)$ is a tuple where, Q is the set of *locations*, $q_0 \in Q$ is the initial location, Σ is the alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function and $F \subseteq Q$ is the set of accepting locations [3].

Function δ is extended to words by setting $\delta(q, \epsilon) = \epsilon$, and $\delta(q, a{\cdot}\sigma) = \delta(\delta(q, a), \sigma)$. A word σ is *accepted* by $\mathcal{A}$ *starting from location* q if $\delta(q, \sigma) \in F$, and σ is *accepted* by $\mathcal{A}$ if σ is accepted starting from the initial location q_0. The *language* of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is is the set of all accepted words from location q_0.

Classification of properties. A *regular property* is a language accepted by an automaton. In the sequel, we consider only regular properties and we refer to them as just properties. Safety (res. co-safety) properties are sub-classes of regular properties. Safety properties are the prefix-closed languages that can be accepted by an automaton. The set of safety properties is denoted as ρ_s. Co-safety properties are the extension-closed languages that can be accepted by an automaton. The set of co-safety properties is denoted as ρ_{cs}. We define another subset of regular properties $\rho = \rho_s \cup \rho_{cs}$. A regular property that belongs to this subset is either a safety or a co-safety property.

Thus, an automaton $\mathcal{A} = (Q, q_0, \Sigma, \delta, F)$ is a *safety* automaton if $\forall a \in \Sigma, q \notin F : \delta(q, a) \notin F$, and is a *co-safety* automaton if $\forall a \in \Sigma, q \in F : \delta(q, a) \in F$.

[3] In the rest of the paper the term automaton refers to a deterministic and complete automaton.



(a) $\mathcal{A}_{RE1}$ (b) $\mathcal{A}_{RE2}$ (c) $\mathcal{A}_{S1}$

(d) $\mathcal{A}_{S2}$ (e) $\mathcal{A}_{CS1}$ (f) $\mathcal{A}_{CS2}$

Fig. 2: Automata defining properties $RE1$, $RE2$, $S1$, $S2$, $CS1$ and $CS2$.

Example 1 (Properties defined as automata). Consider the following properties:
- $RE1$: Action a followed by b or c should alternate starting with an a.
- $RE2$: The first action should be an a, immediately followed by a b, then immediately followed by another a. This sequence can be repeated again with a c.
- $S1$: After a b occurs, it is forbidden to have an a.
- $S2$: We can have at most two a actions.
- $CS1$: The first two actions should be a followed by b.
- $CS2$: The first three actions should be a followed by b followed by c.

The set of actions $\Sigma = \{a, b, c\}$. The automata in Figure 2 define these properties. Properties $RE1$ and $RE2$ are regular properties (that are neither safety nor co-safety), and are defined by automata in Figures 2a and 2b respectively. Properties $S1$ and $S2$ are safety properties defined by safety automata in Figures 2c and 2d respectively. Properties $CS1$ and $CS2$ are co-safety properties defined by co-safety automata in Figures 2e and 2f respectively.

Intersection of automata. Let $\mathcal{A} = (Q, q_0, \Sigma, \delta, F)$ and $\mathcal{A}' = (Q', q'_0, \Sigma, \delta', F')$ be two automata over the same alphabet Σ. The *intersection* of $\mathcal{A}$ and $\mathcal{A}'$, denoted $\mathcal{A} \cap \mathcal{A}'$, is defined as $(Q \times Q', (q_0, q'_0), \Sigma, \delta \times \delta', F \times F')$, where $(\delta \times \delta')((q, q'), a) = (\delta(q, a), \delta'(q', a))$. We have $\mathcal{L}(\mathcal{A} \cap \mathcal{A}') = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}')$.

Example 2 (Intersection of automata). Consider properties $S1$ and $S2$ from Example 1. Intersection of properties $S1$ and $S2$ informally mean that *"We can have at most two "a" actions before a 'b" action occurs."* The automaton in Figure 3 defines the property $S1 \cap S2$. A word over Σ is accepted by the automaton $\mathcal{A}_{S1 \cap S2}$ if it is accepted by both automata $\mathcal{A}_{S1}$ and $\mathcal{A}_{S2}$.

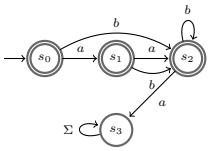


Fig. 3: $\mathcal{A}_{S1 \cap S2}$.

2.2 Runtime Enforcement

Several runtime enforcement frameworks exist, already mentioned in the introduction. In this paper, we follow the framework of [13], where enforcement monitors are syn-

thesized from regular properties modeled as automata. The input-output behavior of an enforcement monitor is specified by an enforcement function. The enforcement function E_φ transforms some input word σ which is possibly incorrect w.r.t. φ. Enforcement mechanism has the ability of blocking events when a violation is detected. The output $E_\varphi(\sigma)$ is a prefix of the input word σ. Some requirements are defined on the enforcement function: *soundness*, *transparency* and *monotonicity*. Soundness means that for any input word σ, if the output $E_\varphi(\sigma)$ is not empty ($\neq \epsilon$), then it must satisfy φ. Transparency expresses how an enforcement mechanism is allowed to correct the input sequence: the output word is a prefix of the input, and if the input word satisfies the property, the output should be equal to the input. Monotonicity is related to online behavior of the enforcement mechanism, that it cannot undo what is already released as output during the incremental computation, and new events can be only appended to the tail of the output. Formal constraints are recalled in Appendix A, page 16 and are detailed in [13]. Let us see a definition of an enforcement function that incrementally builds the output.

Definition 1 (Enforcement function). *Given a property $\varphi \subseteq \Sigma^*$, the enforcement function is $E_\varphi : \Sigma^* \to \Sigma^*$, and is defined as $E_\varphi(\sigma) = \Pi_1\Big(\mathrm{store}_\varphi(\sigma)\Big)$.*

where:

- $\kappa_\varphi(\sigma) = (\sigma \in \varphi)$
- $\mathrm{store}_\varphi : \Sigma^* \to \Sigma^* \times \Sigma^*$ *is defined as:*

$$\begin{aligned} \mathrm{store}_\varphi(\epsilon) &= (\epsilon, \epsilon) \\ \mathrm{store}_\varphi(\sigma \cdot a) &= \begin{cases} (\sigma_s \cdot \sigma_c \cdot a, \epsilon) & \textit{if } \kappa_\varphi(\sigma_s \cdot \sigma_c \cdot a), \\ (\sigma_s, \sigma_c \cdot a) & \textit{otherwise} \end{cases} \end{aligned}$$

with $(\sigma_s, \sigma_c) = \mathrm{store}_\varphi(\sigma)$.

Function store_φ takes a word over Σ as input and returns a pair of words over Σ, and the first element of the output of function store_φ is the output of the enforcement function. The first element of the output of function store_φ is a prefix of the input that satisfies property φ; and the second element is a suffix of the input that the enforcer cannot output yet. Function store_φ is defined inductively (see [13] for detailed explanation).

Predictive runtime enforcement. In predictive RE setting [13], instead of considering Σ^* as the language of possible inputs, another property $\psi \subseteq \Sigma^*$ defines the set of possible sequences that the EM receives as input at runtime. ψ is considered to be an abstract model or knowledge of the system obtained using some static-analysis techniques.

In addition to soundness[4], transparency and monotonicity constraints, a predictive enforcement function should satisfy an additional constraint called *urgency*. Constraint urgency expresses that if the input received so far does not satisfy the property φ, it can still be released as output if all possible inputs that the EM will receive in the future will allow to satisfy φ. A predictive enforcement function takes words that belong to

[4] In the predictive setting, soundness is restricted to input words that belong to ψ.

the input property ψ as input and outputs words that belong to φ. In addition to property φ, a predictive enforcement function also requires property ψ as input.

Definition 2 (Predictive enforcement function). *Given $\varphi, \psi \subseteq \Sigma^*$, the predictive enforcement function is $E_{\psi \triangleright \varphi} : \Sigma^* \to \Sigma^*$, defined as $E_{\psi \triangleright \varphi}(\sigma) = \Pi_1\Big(\text{store}_{\psi \triangleright \varphi}(\sigma)\Big)$.*

The only difference in $\text{store}_{\psi \triangleright \varphi}$ compared to store_{φ} (Definition 1) is in the condition that is checked upon receiving a new event a (to output events that were not released earlier (σ_c) followed by the received event a). In the non-predictive case function κ_φ is used for this purpose, which checks whether the input sequence received so far belongs to property φ. In the predictive case, it is replaced with the following function $\kappa_{\psi \triangleright \varphi}$.

$$\kappa_{\psi \triangleright \varphi}(\sigma) = (\forall \sigma_{\text{con}} \in \Sigma^* : \sigma \cdot \sigma_{\text{con}} \in \psi \implies \exists \sigma' \in \Sigma^* : \sigma' \preccurlyeq \sigma_{\text{con}} \wedge \sigma \cdot \sigma' \in \varphi).$$

Function $\kappa_{\psi \triangleright \varphi}$ uses input property ψ to anticipate the future and release the received input earlier. If every possible continuation of the received input according to ψ allows to satisfy φ in the future, then it returns true. It returns false if the received input sequence does not satisfy φ and there is a continuation of the received input that will not allow to satisfy φ.

Remark 1. In [13] we prove that the predictive enforcement function (Definition 2) satisfies soundness, transparency, monotonicity and urgency constraints. When $\psi = \Sigma^*$, we show that the urgency constraint reduces to one of the transparency constraints and the function $\kappa_{\psi \triangleright \varphi}(\sigma)$ can be simplified as $\kappa_{\psi \triangleright \varphi}(\sigma) = (\sigma \in \varphi)$. Thus, Definition 2 reduces to Definition 1 when $\psi = \Sigma^*$.

Algorithms and implementation. Algorithms describing how to implement the enforcement functions are detailed in [13]. An implementation of these algorithms in Python is available for download at: https://github.com/SrinivasPinisetty/PredictiveRE.

3 Monolithic vs. Compositional Runtime Enforcement Approaches

In this section we discuss three different approaches for enforcing multiple properties, namely, monolithic RE, and RE using serial or parallel composition of EMs. To simplify notation and explanations, we consider the enforcement of only two properties φ_1 and φ_2. The results generalize to any number of properties. Properties φ_1 and φ_2 are assumed to be regular properties defined by complete and deterministic automata $\mathcal{A}_{\varphi_1}$ and $\mathcal{A}_{\varphi_2}$ over some alphabet Σ.

3.1 Monolithic Approach

In the monolithic approach, properties are first combined using intersection and an EM for the resulting property is synthesized (See Figure 1a). Specifically, given any two regular properties φ_1 and φ_2, to enforce both these properties, we first compute $\varphi = \varphi_1 \cap \varphi_2$ (by computing the product of the automata for φ_1 and φ_2). Then we synthesize an EM for φ using the approach described in Section 2.2. For any input word σ, $E_\varphi(\sigma)$ is sound and transparent with respect to $\varphi_1 \cap \varphi_2$. Since $E_\varphi(\sigma)$ satisfies $\varphi_1 \cap \varphi_2$, $E_\varphi(\sigma)$

σ	$E_{RE1}(\sigma)$	$E_{RE2}(E_{RE1}(\sigma))$	$E_{RE2}(\sigma)$	$E_{RE1}(E_{RE2}(\sigma))$
a	ϵ	ϵ	ϵ	ϵ
ab	ab	ϵ	ϵ	ϵ
aba	ab	ϵ	aba	$ab \notin RE1 \cap RE2$
$abac$	$abac$	$aba \notin RE1 \cap RE2$	aba	$ab \notin RE1 \cap RE2$

Table 1: Counterexamples to compositionality of the serial approach.

obviously satisfies both φ_1 and φ_2. Regarding transparency, if the input satisfies only φ_1 (or only φ_2), then the output will not be equal to the input. The output will be equal to the input only if the input satisfies $\varphi_1 \cap \varphi_2$ (i.e., the input satisfies both φ_1 and φ_2).

Remark 2 (Maximality). From [13] we know that for any given regular property φ, for any sequence $\sigma \in \Sigma^*$, if $E_\varphi(\sigma) \neq \epsilon$ then it is the maximal prefix of σ that satisfies φ. Thus, for any given regular properties φ_1 and φ_2, for any input word $\sigma \in \Sigma^*$, if $E_{\varphi_1 \cap \varphi_2}(\sigma) \neq \epsilon$, it is the maximal prefix of σ that satisfies both properties φ_1 and φ_2 (i.e., maximal prefix that belongs to $\varphi_1 \cap \varphi_2$).

Example 3. Consider the property $S1 \cap S2$ defined by the automaton in Figure 3, where $\Sigma = \{a, b\}$. Consider input sequence $\sigma = abbb$. The output of the EM $E_{S1 \cap S2}(abbb) = abbb$. Notice that the word $abbb$ is accepted by the automaton $\mathcal{A}_{S1 \cap S2}$, and it is also accepted by automata $\mathcal{A}_{S1}$ and $\mathcal{A}_{S2}$. Consider another input sequence $\sigma = aaabb$. The output of the EM will be aa which is the maximal prefix of σ accepted by $\mathcal{A}_{S1 \cap S2}$. Notice that the input word $aaabb$ is accepted by $\mathcal{A}_{S1}$, but is not accepted by $\mathcal{A}_{S2}$.

3.2 Serial Composition of Enforcement Monitors

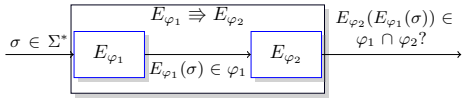


Fig. 4: Serial composition.

Given two properties φ_1 and φ_2, we can synthesize EMs E_{φ_1} and E_{φ_2} for each of them, and then compose them in series, as illustrated in Figure 4. In this type of serial composition the output of E_{φ_1} is fed as input to E_{φ_2}. As a result we obtain a new EM, denoted $E_{\varphi_1} \Rrightarrow E_{\varphi_2}$. In this section we investigate whether $E_{\varphi_1} \Rrightarrow E_{\varphi_2}$ generally enforces $\varphi_1 \cap \varphi_2$. We are also interested to see whether the final output that we obtain using the serial composition approach is equal to the output we would obtain using the monolithic approach.

Let us now formally define serial composition of two EMs.

Definition 3 (Serial composition of enforcement monitors). *Let $E_{\varphi_1} : \Sigma^* \to \Sigma^*$ be the enforcer for a property $\varphi_1 \subseteq \Sigma^*$, and $E_{\varphi_2} : \Sigma^* \to \Sigma^*$ be the enforcer for a property $\varphi_2 \subseteq \Sigma^*$. Their serial composition is a new enforcer $E_{\varphi_1} \Rrightarrow E_{\varphi_2} : \Sigma^* \to \Sigma^*$ defined as follows: $\forall \sigma \in \Sigma^*, (E_{\varphi_1} \Rrightarrow E_{\varphi_2})(\sigma) = E_{\varphi_2}(E_{\varphi_1}(\sigma))$.*

Example 4 (Composing enforcers in series does not generally enforce both properties). Let the automaton in Figure 2a define property φ_1, and the automaton in Figure 2b define property φ_2. Table 1 shows the outputs $E_{RE1}(\sigma)$, $E_{RE2}(E_{RE1}(\sigma))$, $E_{RE2}(\sigma)$, and $E_{RE1}(E_{RE2}(\sigma))$, when the input sequence $\sigma = abac$ is processed incrementally. We notice that generally $E_{RE2}(E_{RE1}(\sigma)) \neq E_{RE1}(E_{RE2}(\sigma))$, which implies that

$E_{\varphi_1} \Rrightarrow E_{\varphi_2}$ and $E_{\varphi_2} \Rrightarrow E_{\varphi_1}$ generally differ. Also notice that $E_{RE1}(E_{RE2}(abac))$ is ab which does not satisfy $RE1 \cap RE2$, and $E_{RE2}(E_{RE1}(abac))$ is aba which also does not satisfy $RE1 \cap RE2$.

What the above example shows is that, independently of the order used, the serial approach is generally non-compositional. In particular, given regular properties φ_1, φ_2, and input sequence $\sigma \in \Sigma^*$, neither $E_{\varphi_1}(E_{\varphi_2}(\sigma))$ nor $E_{\varphi_2}(E_{\varphi_1}(\sigma))$ generally satisfy $\varphi_1 \cap \varphi_2$. This is despite the fact that $E_{\varphi_1}(\sigma)$ is guaranteed to satisfy φ_1 and $E_{\varphi_2}(\sigma)$ is guaranteed to satisfy φ_2. As we shall see later in Section 4, the serial approach is compositional for certain subclasses of regular properties.

3.3 Predictive Runtime Enforcement in Serial Composition (does not help)

Let us now consider the predictive RE mechanism in a serial composition setting. Predictive RE mechanism makes use of knowledge of the system allowing to output some events immediately instead of delaying or blocking them [13]. In predictive RE, instead of letting the input sequence σ range over Σ^*, we let it range over a given property $\psi \subseteq \Sigma^*$ where ψ captures a model or some knowledge that we have about the system.

When we consider serial composition of enforcers $E_{\varphi_1} \Rrightarrow E_{\varphi_2}$, we know that every input received by E_{φ_2} belongs to φ_1. This is because every input received by E_{φ_2} is an output generated by E_{φ_1}, and the latter is guaranteed to enforce φ_1. Thus, it makes sense to use the predictive method to generate the downstream enforcer, hoping that this will result in improved enforcement behavior. Let us now see whether this is indeed the case, i.e., whether taking the downstream enforcer to be $E_{\varphi_1 \triangleright \varphi_2}$ instead of E_{φ_2}, will be of any advantage.[5]

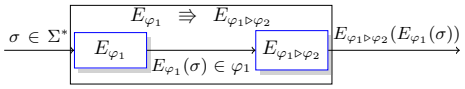


Fig. 5: Serial composition with prediction.

This alternative approach with prediction is illustrated in Figure 5. We consider serial composition where the downstream EM (i.e., the EM for φ_2) is the predictive EM $E_{\varphi_1 \triangleright \varphi_2}$. The input that $E_{\varphi_1 \triangleright \varphi_2}$ receives is $E_{\varphi_1}(\sigma)$, which is guaranteed to belong to φ_1.

One generally expects that in some cases, considering predictive RE for the second EM in serial composition allows to output some events earlier, and there may be some situations where $E_{\varphi_1 \triangleright \varphi_2}(E_{\varphi_1}(\sigma)) \neq E_{\varphi_2}(E_{\varphi_1}(\sigma))$ (in fact $E_{\varphi_1 \triangleright \varphi_2}(E_{\varphi_1}(\sigma))$ to be longer compared to $E_{\varphi_2}(E_{\varphi_1}(\sigma))$). Surprisingly, we found that this is not the case. For any given regular properties φ_1 and φ_2, we show that considering predictive enforcement for the second EM in serial composition has no advantage and the output for any input sequence will be equal to the output that we obtain using standard EM.

According to the definition of predictive enforcer (Definition 2), given properties $\psi, \varphi \subseteq \Sigma^*$, where ψ is the property defining the input and φ is the property we want to enforce, at runtime though the received input sequence σ_o does not satisfy the property φ, it is still immediately released as output if for every possible extension σ_{con} of σ_o such that $\sigma_o \cdot \sigma_{con} \in \psi$, there is a prefix $\sigma' \preccurlyeq \sigma_{con}$ such that $\sigma_o \cdot \sigma'$ satisfies φ.

As illustrated in Example 5, during some steps at runtime, the received input may not belong to the input property ψ (but we know that it will be eventually in the future

[5] Note that in order to compute $E_{\varphi_1 \triangleright \varphi_2}$ both φ_1 and φ_2 need to be known.

σ_o	$E_\psi(\sigma_o)$	$E_{\psi \triangleright \varphi}(E_\psi(\sigma_o))$	$E_\varphi(E_\psi(\sigma_o))$	$E_{\psi \triangleright \varphi}(\sigma_o)$
r	ϵ	ϵ	ϵ	r
ra	ϵ	ϵ	ϵ	ra
$rag \in \psi$	rga	rga	rga	rga
$ragr$	rga	rga	rga	$rgar$
$ragra$	rga	rga	rga	$rgara$
$ragrag \in \psi$	$ragrag$	$ragrag$	$ragrag$	$ragrag$

Table 2: Comparing predictive RE, and serial composition with/without prediction.

according to ψ). According to the definition of $\kappa_{\psi \triangleright \varphi}$, predicting future input sequences helps only at those moments when the received input σ_o does not belong to ψ, and it also does not belong to φ (the property we want to enforce), but if every extension of σ_o according to ψ allows to satisfy φ, then σ_o is output immediately since according to ψ we will receive some more events in the future that will certainly allow to satisfy φ.

Consequently, in the serial composition approach, where we consider the enforcer for the second property φ_2 as predictive enforcer and the input property for it to be φ_1, in serial composition $E_{\varphi_1} \Rrightarrow E_{\varphi_1 \triangleright \varphi_2}$, first notice that E_{φ_1} (standard non-predictive enforcer for φ_1) will only output sequences that belong to φ_1. When the observed input does not satisfy φ_1, then E_{φ_1} will block and wait for more events to be received. So, what the second enforcer $E_{\varphi_1 \triangleright \varphi_2}$ receives as input will always belong to φ_1. Thus, knowledge of φ_1 for the second enforcer in serial composition is not useful in any case.

Example 5. Consider two properties ψ and φ defined by the automata in Figure 6, where the set of actions $\Sigma = \{r, a, g\}$. Let us consider input sequence $\sigma = ragrag \in \psi$. Table 2 illustrates the output at each step when the input word σ is processed incrementally. At each step, the observation of the input is a prefix of σ (denoted as σ_o). In the third column ($E_{\psi \triangleright \varphi}(E_\psi(\sigma_o))$), we consider serial composition of the enforcer for ψ and $E_{\psi \triangleright \varphi}$ (thus, $E_\psi(\sigma_o)$ is fed as input to $E_{\psi \triangleright \varphi}$). In the fifth column ($E_{\psi \triangleright \varphi}(\sigma_o)$), ψ is considered as a model of the system, and the observed input σ_o is fed as input to $E_{\psi \triangleright \varphi}$.

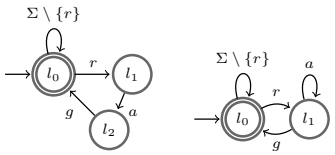


Fig. 6: $\mathcal{A}_\psi$ (left), $\mathcal{A}_\varphi$ (right).

We can clearly notice that from where the predictive enforcer $E_{\psi \triangleright \varphi}$ receives the input also matters. If $E_{\psi \triangleright \varphi}$ receives the input event after event from some system (event emitter), where ψ is a model of the system, then prediction indeed helps to release events earlier as illustrated in this example (column $E_{\psi \triangleright \varphi}(\sigma_o)$ in Table 2) and other examples in [13]. If $E_{\psi \triangleright \varphi}$ receives the input from a standard enforcer for ψ (column $E_{\psi \triangleright \varphi}(E_\psi(\sigma_o))$ in Table 2) which is the case in serial composition, knowledge of ψ for the second enforcer is not useful. In this particular example, we can also notice that $E_{\psi \triangleright \varphi}(E_\psi(\sigma_o)) = E_\varphi(E_\psi(\sigma_o))$.

Theorem 1 (Serial composition with prediction does not help). *For any two regular properties $\varphi_1 \subseteq \Sigma^*$ and $\varphi_2 \subseteq \Sigma^*$, $\forall \sigma \in \Sigma^*, E_{\varphi_1 \triangleright \varphi_2}(E_{\varphi_1}(\sigma)) = E_{\varphi_2}(E_{\varphi_1}(\sigma))$.*

We prove that given any two regular properties φ_1 and φ_2, in the serial composition approach, considering predictive enforcement for the second enforcer (when we do not

have a model of the system and when the first enforcer is a standard non-predictive one) has no advantage, and for any input, the output will be equal to the output we obtain using standard non-predictive enforcer for φ_2.

We thus conclude that if we do not have a model or knowledge of the system, and if the first enforcer in non-predictive, using predictive RE for second enforcer (though we know that the input words it receives belongs to φ_1) is not useful. When we have a property describing possible input sequences that the enforcement mechanism receives from a system, and if first enforcer is a predictive enforcer, then using predictive RE for the second enforcer in serial composition is useful.

3.4 Parallel Composition of Enforcement Monitors

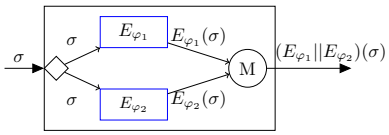


Fig. 7: Parallel composition.

We now introduce the parallel composition of two EMs E_{φ_1} and E_{φ_2}, denoted $E_{\varphi_1}||E_{\varphi_2}$, and check whether it enforces $\varphi_1 \cap \varphi_2$. As illustrated in Figure 7, in the parallel composition approach, the input sequence σ is fed into both E_{φ_1} and E_{φ_2} simultaneously. The enforcement monitors E_{φ_1} and E_{φ_2} run in parallel. Then their outputs $E_{\varphi_1}(\sigma)$ and $E_{\varphi_2}(\sigma)$ are somehow *merged* (block denoted M in the figure) in order to obtain the output of the composite enforcer $E_{\varphi_1}||E_{\varphi_2}$.

In our case, the merge block outputs the maximal common prefix of its inputs. Formally, we define the *merge* of two words $\sigma_1, \sigma_2 \in \Sigma^*$ as $merge(\sigma_1, \sigma_2) = max_{\preccurlyeq}\{\sigma \in \Sigma^* | \sigma \preccurlyeq \sigma_1 \wedge \sigma \preccurlyeq \sigma_2\}$.

Definition 4 (Parallel composition of enforcement monitors). *Let $E_{\varphi_1} : \Sigma^* \to \Sigma^*$ be the enforcer for a property $\varphi_1 \subseteq \Sigma^*$, and $E_{\varphi_2} : \Sigma^* \to \Sigma^*$ be the enforcer for a property $\varphi_2 \subseteq \Sigma^*$. Their parallel composition is the enforcer $E_{\varphi_1}||E_{\varphi_2} : \Sigma^* \to \Sigma^*$ defined as follows: $\forall \sigma \in \Sigma^*, (E_{\varphi_1}||E_{\varphi_2})(\sigma) = merge(E_{\varphi_1}(\sigma), E_{\varphi_2}(\sigma))$.*

Example 6 (Composing enforcers in parallel does not generally enforce both properties). Consider two regular properties $RE1$ and $CS2$, defined by automata in Figures 2a and 2f. Table 3 illustrates the outputs of E_{RE1}, E_{CS2}, and $E_{RE1}||E_{CS2}$ when the input sequence abc is processed incrementally. Notice that $(E_{RE1}||E_{CS2})(abc)$ is ab which does not satisfy $RE1 \cap CS2$.

For some regular properties φ_1, φ_2, for some input sequence $\sigma \in \Sigma^*$, $(E_{\varphi_1}||E_{\varphi_2})(\sigma)$ may not satisfy $\varphi_1 \cap \varphi_2$. Thus, when we want to enforce two regular properties φ_1 and φ_2, we cannot use parallel composition since the final output may not belong to $\varphi_1 \cap \varphi_2$ as illustrated in our example. As we shall see later in Section 4, parallel composition is compositional for certain subclasses of regular properties.

Remark 3 (Merge). Our $merge$ function is independent of the properties and also of the enforcers. This is intentional, as we want to achieve maximal modularity. An alternative merge operation is one which receives letters from the two enforcers one at a time, and outputs a letter if and only if both enforcers choose to output it; otherwise it buffers a letter until it it released by both enforcers. This alternative merge would be compositional for all regular properties.

4 Compositionality for Subclasses of Regular Properties

In Section 3, we saw that the serial and parallel approaches are not compositional for all regular properties. In this section we consider subclasses of regular properties, and in particular, safety or co-safety properties, and investigate compositionality for these subclasses.

σ	$E_{RE1}(\sigma)$	$E_{CS2}(\sigma)$	$(E_{RE1}\|\|E_{CS2})(\sigma)$
a	ϵ	ϵ	ϵ
ab	ab	ϵ	ϵ
abc	ab	abc	$ab \notin RE1 \cap CS2$

Table 3: Counterexample to compositionality of the parallel approach.

Safety and co-safety properties. Given two properties φ_1 and φ_2 such that one is a safety property and the other is a co-safety property, for some input words, the output of serial and parallel composition of EMs E_{φ_1} and E_{φ_2} may not satisfy $\varphi_1 \cap \varphi_2$. Moreover, the output obtained using serial and parallel composition approaches may not be equal to the output obtained using the monolithic approach. For serial composition, the order of composition of enforcers also matters. We illustrate this via Example 7.

σ	$E_{S1}(\sigma)$	$E_{CS2}(\sigma)$	$E_{S1\cap CS2}(\sigma)$	$E_{CS2}(E_{S1}(\sigma))$	$E_{S1}(E_{CS2}(\sigma))$	$(E_{S1}\|\|E_{CS2})(\sigma)$
ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ
a	a	ϵ	ϵ	ϵ	ϵ	ϵ
ab	ab	ϵ	ϵ	ϵ	ϵ	ϵ
abc	ab	abc	ϵ	ϵ	ab	ab

Table 4: Composing enforcers of a safety and a co-safety property.

Example 7 (Composing enforcers of a safety and a co-safety property does not generally enforce both properties). Consider the safety automaton in Figure 2c and the co-safety automaton in Figure 2f defining properties $S1$ and $CS2$ respectively. Table 4 presents $E_{S1}(\sigma)$, $E_{CS2}(\sigma)$, $E_{S1\cap CS2}(\sigma)$, $E_{CS2}(E_{S1}(\sigma))$, $E_{S1}(E_{CS2}(\sigma))$ and $(E_{S1}||E_{CS2})(\sigma)$ when the input sequence $\sigma = abc$ is processed incrementally. We can notice that in the last step, upon receiving abc, $E_{S1}(E_{CS2}(\sigma))$ and $(E_{S1}||E_{CS2})(\sigma)$ is ab which does not satisfy $CS2$ and thus also does not satisfy $S1 \cap CS2$.

Composing in series enforcers of a safety property and a regular property. In Section 3 we already saw that for two regular properties serial composition may not work. Via Example 7 we also saw that when one property is a safety property and the other property is a co-safety property, then serial composition approach may not work. However, in Example 7, we can also notice that at every step $E_{CS2}(E_{S1}(\sigma))$ satisfies $S1 \cap CS2$.

In fact, we show that given two properties φ_1 and φ_2, if one of them is identified to be a *safety* property (say φ_1), then serial composition approach works by fixing the order of composition. Property φ_1 should be considered as the first (upstream) property in serial composition. The second (downstream) property φ_2 can be any regular property.

Since φ_1 is a safety property, for any input word σ, all the prefixes of $E_{\varphi_1}(\sigma)$ satisfies φ_1. Thus when $E_{\varphi_1}(\sigma)$ is fed as input to E_{φ_2} (where φ_2 is any regular property), its output will be the maximal prefix of $E_{\varphi_1}(\sigma)$ that satisfies φ_2, which will be the maximal prefix of σ satisfying both properties φ_1 and φ_2.

Theorem 2 (Serial composition of a safety and a regular property). *Given a safety property $\varphi_1 \subseteq \Sigma^*$, and a regular property $\varphi_2 \subseteq \Sigma^*$, $\forall \sigma \in \Sigma^*$, $E_{\varphi_2}(E_{\varphi_1}(\sigma)) = E_{\varphi_1 \cap \varphi_2}(\sigma)$.*

Let us now see whether serial or parallel composition works when both properties φ_1 and φ_2 are safety (or both co-safety).

Serial composition (safety properties). When both φ_1 and φ_2 are safety properties, it is straightforward from Theorem 2 that serial composition approach works and the order of composition of enforcers also does not matter. To understand further, consider $E_{\varphi_1} \Rrightarrow E_{\varphi_2}$. For any word σ, $E_{\varphi_1}(\sigma)$ is the maximal prefix of σ satisfying φ_1. Since φ_1 is prefix-closed, any prefix of $E_{\varphi_1}(\sigma)$ satisfies φ_1. $E_{\varphi_2}(E_{\varphi_1}(\sigma))$ will be the maximal prefix of $E_{\varphi_1}(\sigma)$ satisfying φ_2, which also satisfies φ_1. Similarly, we can also easily notice that $E_{\varphi_2} \Rrightarrow E_{\varphi_1}$ will also satisfy both φ_1 and φ_2 since both are safety properties.

Corollary 1. *Given any two safety properties φ_1 and φ_2, $\forall \sigma \in \Sigma^*$, $E_{\varphi_1}(E_{\varphi_2}(\sigma)) = E_{\varphi_2}(E_{\varphi_1}(\sigma)) = E_{\varphi_1 \cap \varphi_2}(\sigma)$.*

Corollary 1 is a direct consequence of Theorem 2. When both φ_1 and φ_2 are safety properties, for any input word, the output we obtain by composing enforcers E_{φ_1} and E_{φ_2} in series will be equal to the output obtained using the monolithic approach and thus satisfies $\varphi_1 \cap \varphi_2$ (if it is $\neq \epsilon$).

Parallel composition (safety properties). When both φ_1 and φ_2 are safety properties, then parallel composition also works. Since both φ_1 and φ_2 are prefix-closed, any prefix of $E_{\varphi_1}(\sigma)$ satisfies φ_1, and any prefix of $E_{\varphi_2}(\sigma)$ satisfies φ_2. $(E_{\varphi_1}||E_{\varphi_2})(\sigma)$ is the maximal common prefix of both these words which also certainly satisfies both φ_1 and φ_2 .

Theorem 3 (Parallel composition of safety properties). *Given any two safety properties φ_1 and φ_2, $\forall \sigma \in \Sigma^*$, $(E_{\varphi_1}||E_{\varphi_2})(\sigma) = E_{\varphi_1 \cap \varphi_2}(\sigma)$.*

Theorem 3 shows that when both φ_1 and φ_2 are safety properties, for any input word, the output we obtain by composing enforcers E_{φ_1} and E_{φ_2} in parallel will be equal to the output obtained using the monolithic approach and thus satisfies $\varphi_1 \cap \varphi_2$ (if it is $\neq \epsilon$).

σ	$E_{S1}(\sigma)$	$E_{S2}(\sigma)$	$E_{S1 \cap S2}(\sigma)$	$E_{S2}(E_{S1}(\sigma))$	$E_{S1}(E_{S2}(\sigma))$	$(E_{S2}\|\|E_{S1})(\sigma)$
a	a	a	a	a	a	a
aa	aa	aa	aa	aa	aa	aa
aaa	aaa	aa	aa	aa	aa	aa
$aaab$	$aaab$	aa	aa	aa	aa	aa

Table 5: Composing enforcers of two safety properties.

Example 8 (Composing enforcers of two safety properties). Let us consider properties $S1$ and $S2$ defined by automata in Figures 2c and 2d. Table 5 illustrates the output of different methods for enforcing $S1 \cap S2$ when the input sequence $aaab$ is processed incrementally. We can notice that at every step, all the methods result in the same output which satisfies the property $S1 \cap S2$.

Co-safety properties. When both φ_1 and φ_2 are co-safety properties, then both serial and parallel composition work. Regarding serial composition, consider $E_{\varphi_1} \Rrightarrow E_{\varphi_2}$. For any input word $\sigma \in \Sigma^*$, since φ_1 is extension-closed, if $\sigma \notin \varphi_1$, then $E_{\varphi_1}(\sigma) = \epsilon$ and thus $E_{\varphi_2}(E_{\varphi_1}(\sigma)) = \epsilon$ (irrespective of whether $\sigma \in \varphi_2$ or not). If $\sigma \in \varphi_1$, then $E_{\varphi_1}(\sigma) = \sigma$. In this case, if $\sigma \in \varphi_2$, then $E_{\varphi_2}(E_{\varphi_1}(\sigma)) = \sigma$ which is the maximal prefix of σ satisfying both φ_1 and φ_2. But if $\sigma \notin \varphi_2$, then since φ_2 is extension-closed, there is no prefix of $E_{\varphi_1}(\sigma)$ (which is σ) that satisfies φ_2, and the output will be ϵ.

Regarding parallel composition, for any input word σ if $\sigma \notin \varphi_1$ and $\sigma \notin \varphi_2$, then $E_{\varphi_1}(\sigma)$ and $E_{\varphi_2}(\sigma)$ will be ϵ (since the properties are extension-closed) and thus $(E_{\varphi_1}||E_{\varphi_2})(\sigma)$ will also be ϵ. If σ satisfies only one property among φ_1 and φ_2, then the output of one of the enforcers will be ϵ, and thus the final output (which is the maximal common prefix of both the outputs) will be ϵ. Finally, if σ satisfies both φ_1 and φ_2 then $E_{\varphi_1}(\sigma)$ and $E_{\varphi_2}(\sigma)$ will be σ, the final output will thus be σ in this case.

σ	$E_{CS1}(\sigma)$	$E_{CS2}(\sigma)$	$E_{CS1 \cap CS2}(\sigma)$	$E_{CS2}(E_{CS1}(\sigma))$	$E_{CS1}(E_{CS2}(\sigma))$	$(E_{CS1}\|\|E_{CS2})(\sigma)$
a	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ
ab	ab	ϵ	ϵ	ϵ	ϵ	ϵ
abc	abc	abc	abc	abc	abc	abc
$abca$	$abca$	$abca$	$abca$	$abca$	$abca$	$abca$

Table 6: Composing enforcers of two co-safety properties.

Example 9 (Composing enforcers of two co-safety properties). Consider two co-safety properties $CS1$ and $CS2$ defined by automata in Figures 2e, 2f. From Table 6, when the input sequence abc is processed incrementally, we can notice that at every step, the output of all the three methods (monolithic, serial and parallel composition) are equal and belongs to $CS1 \cap CS2$ (if $\neq \epsilon$).

Theorem 4 (Serial and parallel composition of co-safety properties). *Given any two co-safety properties φ_1 and φ_2, $\forall \sigma \in \Sigma^*$,*

1. $E_{\varphi_1}(E_{\varphi_2}(\sigma)) = E_{\varphi_2}(E_{\varphi_1}(\sigma)) = E_{\varphi_1 \cap \varphi_2}(\sigma)$,
2. $(E_{\varphi_1}||E_{\varphi_2})(\sigma) = E_{\varphi_1 \cap \varphi_2}(\sigma)$.

Theorem 4 shows that when both φ_1 and φ_2 are co-safety properties, for any input word, the output we obtain by composing enforcers E_{φ_1} and E_{φ_2} in series (or parallel) will be equal to the output obtained using the monolithic approach and thus satisfies $\varphi_1 \cap \varphi_2$ (if it is $\neq \epsilon$).

Monolithic, serial and parallel composition approaches. From Theorems 2, 3 and 4 (also illustrated by our examples), we conclude that for safety (or co-safety) properties, all the three approaches are equivalent with respect to the input-output behavior. For any input word σ, we obtain the same output using any of these three approaches, which is the maximal prefix of σ that satisfies all the properties (if the output is $\neq \epsilon$).

Corollary 2. *Given two safety (co-safety) properties φ_1 and φ_2, for any input sequence $\sigma \in \Sigma^*$ the output of the enforcer that we obtain for enforcing $\varphi_1 \cap \varphi_2$ using any of*

the methods (composing properties and synthesising a single EM, serial composition of enforcers, and parallel composition of enforcers) will be equal.

$$\forall \sigma \in \Sigma^*, E_{\varphi_1 \cap \varphi_2}(\sigma) = E_{\varphi_2}(E_{\varphi_1}(\sigma)) = E_{\varphi_2}(E_{\varphi_1}(\sigma)) = (E_{\varphi_1} || E_{\varphi_2})(\sigma).$$

5 Related Work

Compositionality is essential for the design, analysis and verification of large and complex systems, and has been extensively studied in various settings such as test generation [7], model-checking [8,3] and reactive synthesis [9].

In the area of runtime monitoring, [10] deals with composition of monitors for runtime verification. Composition of two monitors raises an alarm whenever one of them does so, and the authors discuss about different ways to organize monitors (based on how information flows between them). In [4] authors integrate runtime verification in to the BIP (Behavior, Interaction and Priority) framework, which is a component-based framework that allows to build complex systems by coordinating the behavior of atomic components described as labeled transition systems (LTS). However, it does consider sub-properties, and composition of monitors, and deals with integrating verification monitor for a property in to the BIP framework.

Runtime enforcement was initiated by security automata proposed by Schneider [14] that focus on safety properties, and blocks the execution when an illegal sequence of actions (not compliant with the property) is recognized. Several enforcement models have been proposed later which also allow a monitor to correct the input sequence by suppressing and (or) inserting events [11]. Bloem et al. [2] presented a framework to synthesize enforcement monitors for reactive systems, called *shields*, from a set of safety properties. In all these approaches, system is considered as a black-box. Recently, predictive RE framework presented in [13] makes use of a-priori knowledge of the system for providing better quality-of-service. All these approaches focus mainly on synthesis of an EM for a given property but do not handle compositionality of EMs. If enforcing a set of properties is considered, is is done using the monolithic approach.

The framework in [5] deals with producing a monitor from a property defined as a Street automaton. Enforcers in [5] are finite state machines with auxiliary memory, and can be composed by a product-automaton type of construction. This resembles our parallel composition of enforcers, but with a different merge operation which is less modular as product of automata requires the "internals" (e.g., state-space) of both enforcers to be known. Also, serial composition is not discussed in [5].

Polymer [1] is a programming language supporting definition and composition of runtime security policies for Java applications. Policies in Polymer specify runtime constraints on un-trusted Java programs. Polymer allows composition of smaller sub-policy modules.

6 Conclusion

When we want to enforce multiple properties on a system, an obvious solution is to use a monolithic approach, where the properties are first combined into one single property φ, and a single enforcer is synthesized for φ. The drawback of this approach is that it is not modular. In this paper, we study the compositionality of runtime enforcement,

with the goal of developing modular approaches which address scalability, reuse, and security concerns.

On the negative side, we showed that enforcement of regular properties is generally non-compositional, w.r.t. both serial and parallel composition. On the positive side, we identified special cases (subclasses of regular properties, such as safety or co-safety properties) for which enforcement is compositional. We also showed that using the predictive RE method to compute the downstream enforcer in a serial composition setting does not improve quality-of-service. This is a surprising result, since predictive RE generally results in enforcers which anticipate their input to reduce delays in their output [13].

In addition to the benefits listed above, the compositional approach presented in this paper allows for easier fault *localization* compared to the monolithic approach (e.g., identifying which sub-property is causing the problem, when it is impossible to correct a given input stream). We plan to investigate such localization in future work. Future work also includes studying compositionality in the context of runtime enforcement for real-time systems. (Monolithic) RE for real-time systems has been studied in [12,6]. We intend to study serial and parallel composition schemes in these timed settings. Implementing our framework and evaluating it on case studies is also part of future work.

References

1. Bauer, L., Ligatti, J., Walker, D.: Composing expressive runtime security policies. ACM Trans. Softw. Eng. Methodol. 18(3) (2009)
2. Bloem, R., Könighofer, B., Könighofer, R., Wang, C.: Shield synthesis: Runtime enforcement for reactive systems. In: TACAS. LNCS, vol. 9035. Springer (2015)
3. Clarke, E., Long, D., McMillan, K.: Compositional model checking. In: Logic in Computer Science, 1989. LICS '89, Proceedings., Fourth Annual Symposium on. pp. 353–362 (1989)
4. Falcone, Y., Jaber, M., Nguyen, T.H., Bozga, M., Bensalem, S.: Runtime Verification of Component-based Systems in the BIP Framework with Formally-proved Sound and Complete Instrumentation. Softw. Syst. Model. 14(1), 173–199 (Feb 2015)
5. Falcone, Y., Mounier, L., Fernandez, J.C., Richier, J.L.: Runtime enforcement monitors: composition, synthesis, and enforcement abilities. FMSD 38(3), 223–262 (2011)
6. Falcone, Y., Jéron, T., Marchand, H., Pinisetty, S.: Runtime enforcement of regular timed properties by suppressing and delaying events. Science of Computer Programming (Submitted for review) (2014)
7. Godefroid, P.: Compositional dynamic test generation. In: Proceedings of the 34th Annual ACM SIGPLAN-SIGACT. pp. 47–54. POPL, ACM, New York, NY, USA (2007)
8. Grumberg, O., Long, D.E.: Model checking and modular verification. ACM Trans. Program. Lang. Syst. 16(3), 843–871 (May 1994)
9. Kugler, H., Segall, I.: Compositional synthesis of reactive systems from live sequence chart specifications. In: TACAS, York, UK, March. Proceedings. pp. 77–91 (2009)
10. Levy, J., Saïdi, H., Uribe, T.E.: Combining monitors for runtime system verification. Electronic Notes in Theoretical Computer Science 70(4), 112 – 127 (2002), runtime Verification
11. Ligatti, J., Bauer, L., Walker, D.: Run-time enforcement of nonsafety policies. ACM Trans. Inf. Syst. Secur. 12(3), 19:1–19:41 (Jan 2009)
12. Pinisetty, S., Falcone, Y., Jéron, T., Marchand, H., Rollet, A., Nguena Timo, O.: Runtime enforcement of timed properties revisited. FMSD 45(3), 381–422 (2014)
13. Pinisetty, S., Preoteasa, V., Tripakis, S., Jéron, T., Falcone, Y., Marchand, H.: Predictive runtime enforcement. In: Symposium on Applied Computing (SAC-SVT). ACM (2016)
14. Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. 3(1), 30–50 (2000)

A Proofs

In Section 2.2 we informally discussed about the soundness, transparency and monotonicity constraints that an enforcement function for a given property φ should satisfy. Enforcement function definition (Definition 1) satisfies these constraints. In Theorem 5, we recall formal definitions of these constraints. More details and proofs are available in [13].

Theorem 5 (Soundness, transparency, and monotonicity). *Given a property φ, the enforcement function E_φ as per definition 1 is a enforcer satisfying the following soundness, transparency and monotonicity constraints.*

Soundness

$$\forall \sigma \in \Sigma^* : E_\varphi(\sigma) \neq \epsilon \implies E_\varphi(\sigma) \in \varphi \tag{Snd}$$

Transparency

$$\forall \sigma \in \Sigma^* : E_\varphi(\sigma) \preccurlyeq \sigma \tag{Tr1}$$

$$\forall \sigma \in \Sigma^* : \sigma \in \varphi \implies E_\varphi(\sigma) = \sigma \tag{Tr2}$$

Monotonicity

$$\forall \sigma, \sigma' \in \Sigma^* : \sigma \preccurlyeq \sigma' \implies E_\varphi(\sigma) \preccurlyeq E_\varphi(\sigma') \tag{Mo}$$

Proof (of Theorem 5). This result is proved in [13].

In the predictive setting, soundness is restricted to input words that belong to the input property ψ. As discussed briefly in Section 2.2, a predictive enforcer should satisfy another additional constraint called urgency.

Theorem 6 (Soundness, transparency, monotonicity and urgency (Predictive enforcer)). *Given two properties ψ, and φ, the predictive enforcement function $E_{\psi \triangleright \varphi}$ as per definition 2 is a predictive enforcer satisfying constraints (***SndP***), (***Tr1P***), (***Tr2P***), (***Ur***) and (***MoP***).*

Soundness

$$\forall \sigma \in \psi : E_{\psi \triangleright \varphi}(\sigma) \neq \epsilon \implies E_{\psi \triangleright \varphi}(\sigma) \in \varphi \tag{SndP}$$

Transparency

$$\forall \sigma \in \Sigma^* : E_{\psi \triangleright \varphi}(\sigma) \preccurlyeq \sigma \tag{Tr1P}$$

$$\forall \sigma \in \Sigma^* : \sigma \in \varphi \implies E_{\psi \triangleright \varphi}(\sigma) = \sigma \tag{Tr2P}$$

Urgency

$$\begin{aligned} \forall \sigma \in \Sigma^* : (\forall \sigma_{\text{con}} \in \Sigma^* : \sigma \cdot \sigma_{\text{con}} \in \psi \implies \\ \exists \sigma' \in \Sigma^* : \sigma' \preccurlyeq \sigma_{\text{con}} \wedge \sigma \cdot \sigma' \in \varphi) \\ \implies E_{\psi \triangleright \varphi}(\sigma) = \sigma \end{aligned} \tag{Ur}$$

Monotonicity

$$\forall \sigma, \sigma' \in \Sigma^* : \sigma \preccurlyeq \sigma' \implies E_{\psi \triangleright \varphi}(\sigma) \preccurlyeq E_{\psi \triangleright \varphi}(\sigma') \tag{MoP}$$

Proof (of Theorem 5). This result is proved in [13].

We introduce some lemmas used later is proving some of the theorems. Lemma 1 states that for both predictive and non-predictive enforcement functions, for any input sequence σ, the concatenation of the two output words $\sigma_s \cdot \sigma_c$ of the function store will be equal to the input word σ.

Lemma 1. *For all $\sigma, \sigma_s, \sigma_c \in \Sigma^*$, we have*
1. $\text{store}_{\varphi}(\sigma) = (\sigma_s, \sigma_c) \implies \sigma = \sigma_s \cdot \sigma_c$
2. $\text{store}_{\psi \triangleright \varphi}(\sigma) = (\sigma_s, \sigma_c) \implies \sigma = \sigma_s \cdot \sigma_c$

Proof (of Lemma 1). Proof of this lemma is straightforward by using induction on the length of the input word.

Proof (of Theorem 1). We shall prove that given any two regular properties φ_1 and φ_2,

$$\forall \sigma \in \Sigma^*, E_{\varphi_1 \triangleright \varphi_2}(E_{\varphi_1}(\sigma)) = E_{\varphi_2}(E_{\varphi_1}(\sigma)).$$

We prove using induction on the length of the input word σ.

Induction basis. If $\sigma = \epsilon$, from Definitions 1 and 2, $E_{\varphi_1 \triangleright \varphi_2}(E_{\varphi_1}(\sigma)) = E_{\varphi_2}(E_{\varphi_1}(\sigma)) = \epsilon$.

Induction step. Assume that for every $\sigma \in \Sigma^*$ of some length $n \in \mathbb{N}$, Theorem 1 holds. That is, $E_{\varphi_1 \triangleright \varphi_2}(E_{\varphi_1}(\sigma)) = E_{\varphi_2}(E_{\varphi_1}(\sigma))$. We now prove that for any $a \in \Sigma$, Theorem 1 holds for $\sigma \cdot a$. We have the following three possible cases:

- Case $\sigma \cdot a \notin \varphi_1$.
  In this case, from Definition 1 we have $E_{\varphi_1}(\sigma \cdot a) = \sigma_s = E_{\varphi_1}(\sigma)$. Thus we have $E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = E_{\varphi_2}(E_{\varphi_1}(\sigma))$ and $E_{\varphi_1 \triangleright \varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = E_{\varphi_1 \triangleright \varphi_2}(E_{\varphi_1}(\sigma))$. Using induction hypothesis, we can conclude that $E_{\varphi_1 \triangleright \varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a))$.
- Case $\sigma \cdot a \in \varphi_1$ and $\sigma \cdot a \notin \varphi_2$.
  Since $\sigma \cdot a \in \varphi_1$, from Definition 1, using Theorem 5 (**Tr2**) and Lemma 1, we have $E_{\varphi_1}(\sigma \cdot a) = \sigma \cdot a$.
  Let us first examine $E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = E_{\varphi_2}(\sigma \cdot a)$. From Definition 1, $\kappa_{\varphi_2}(\sigma \cdot a)$ evaluates to false in this case since $\sigma \cdot a \notin \varphi_2$. Thus, $E_{\varphi_2}(\sigma \cdot a) = E_{\varphi_2}(\sigma)$.
  Let us examine $E_{\varphi_1 \triangleright \varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = E_{\varphi_1 \triangleright \varphi_2}(\sigma \cdot a)$. From Definition 2, $\kappa_{\varphi_1 \triangleright \varphi_2}(\sigma \cdot a)$ evaluates to false since $\sigma \cdot a \in \varphi_1$, there is an extension ϵ such that $\sigma \cdot a \cdot \epsilon \in \varphi_1$, and there is no prefix σ' of ϵ such that $\sigma \cdot a \cdot \sigma' \in \varphi_2$. Thus, $E_{\varphi_1 \triangleright \varphi_2}(\sigma \cdot a) = E_{\varphi_1 \triangleright \varphi_2}(\sigma)$.
- Case $\sigma \cdot a \in \varphi_1$ and $\sigma \cdot a \in \varphi_2$.
  Since $\sigma \cdot a \in \varphi_1$, from Definition 1, using Lemma 1 and Theorem 5 (**Tr2**), we have $E_{\varphi_1}(\sigma \cdot a) = \sigma \cdot a$.
  Let us first examine $E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = E_{\varphi_2}(\sigma \cdot a)$. From Definition 1, $\kappa_{\varphi_2}(\sigma \cdot a)$ evaluates to true in this case since $\sigma \cdot a \in \varphi_2$. Using Lemma 1 and Theorem 5 (**Tr2**), we have $E_{\varphi_2}(\sigma \cdot a) = \sigma \cdot a$.
  Let us examine $E_{\varphi_1 \triangleright \varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = E_{\varphi_1 \triangleright \varphi_2}(\sigma \cdot a)$. From Definition 2, $\kappa_{\varphi_1 \triangleright \varphi_2}(\sigma \cdot a)$ evaluates to true since $\sigma \cdot a \in \varphi_2$, for every continuation σ_{con} such that $\sigma \cdot a \cdot \sigma_{con} \in \varphi_1$, ϵ is a prefix of σ_{con} such that $\sigma \cdot a \cdot \epsilon \in \varphi_2$. Using Lemma 1 and Theorem 6 (**Tr2P**), we have $E_{\varphi_1 \triangleright \varphi_2}(\sigma \cdot a) = \sigma \cdot a$.

Proof (of Theorem 2). We shall prove that given a safety property φ_1 and a regular property φ_2, $\forall \sigma \in \Sigma^*, E_{\varphi_2}(E_{\varphi_1}(\sigma)) = E_{\varphi_1 \cap \varphi_2}(\sigma)$. We prove using induction on the length of the input word σ.

Induction basis. If $\sigma = \epsilon$, from Definition 1, $E_{\varphi_1}(\sigma) = \epsilon$ and $E_{\varphi_2}(\sigma) = \epsilon$. Thus, $E_{\varphi_2}(E_{\varphi_1}(\sigma)) = E_{\varphi_1}(E_{\varphi_2}(\sigma)) = \epsilon$. We also have $E_{\varphi_1 \cap \varphi_2}(\sigma) = \epsilon$.

Induction step. Assume that for every $\sigma \in \Sigma^*$ of some length $n \in \mathbb{N}$, Theorem 2 holds. That is, $E_{\varphi_1}(E_{\varphi_2}(\sigma)) = E_{\varphi_2}(E_{\varphi_1}(\sigma)) = E_{\varphi_1 \cap \varphi_2}(\sigma)$. We now prove that for any $a \in \Sigma$, theorem holds for $\sigma \cdot a$. We have the following four possible cases:

– Case $\sigma \cdot a \in \varphi_1$ and $\sigma \cdot a \in \varphi_2$.
Let us first examine $E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a))$. Since $\sigma \cdot a \in \varphi_1$, from Definition 1, using Theorem 5 (**Tr2**) and Lemma 1, we have $E_{\varphi_1}(\sigma \cdot a) = \sigma \cdot a$. Thus, $E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = E_{\varphi_2}(\sigma \cdot a)$. Since $\sigma \cdot a \in \varphi_2$, we can derive that $E_{\varphi_2}(\sigma \cdot a) = \sigma \cdot a$.
Regarding $E_{\varphi_1 \cap \varphi_2}$, in this case, we know that $\sigma \cdot a$ belongs to both φ_1 and φ_2. from Definition 1, using Theorem 5 (**Tr2**) and Lemma 1, we have $E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a) = \sigma \cdot a$.
We thus have $E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a) = \sigma \cdot a$.
– Case $\sigma \cdot a \in \varphi_1$ and $\sigma \cdot a \notin \varphi_2$.
Let us first examine $E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a))$. Since $\sigma \cdot a \in \varphi_1$, similar to the previous case we have $E_{\varphi_1}(\sigma \cdot a) = \sigma \cdot a$, and thus $E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = E_{\varphi_2}(\sigma \cdot a)$. Since $\sigma \cdot a \notin \varphi_2$, $E_{\varphi_2}(\sigma \cdot a) = E_{\varphi_2}(\sigma)$. Since φ_1 is a safety property and $\sigma \cdot a \in \varphi_1$, using Theorem 5 (**Tr2**) we have $E_{\varphi_1}(\sigma) = \sigma$. So, $E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = E_{\varphi_2}(\sigma) = E_{\varphi_2}(E_{\varphi_1}(\sigma))$.
Regarding $E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a)$, since $\sigma \cdot a \notin \varphi_1 \cap \varphi_2$, we have $E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a) = E_{\varphi_1 \cap \varphi_2}(\sigma)$. Thus, using induction hypothesis we can conclude that $E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a)$.
– Case $\sigma \cdot a \notin \varphi_1$.
Let us first examine $E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a))$. Since $\sigma \cdot a \notin \varphi_1$, according to the definition of the enforcement function, $E_{\varphi_1}(\sigma \cdot a) = E_{\varphi_1}(\sigma)$. Thus, $E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = E_{\varphi_2}(E_{\varphi_1}(\sigma))$. Since $\sigma \cdot a \notin \varphi 1$, we have $\sigma \cdot a \notin \varphi_1 \cap \varphi_2$, and $E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a) = E_{\varphi_1 \cap \varphi_2}(\sigma)$.
Using induction hypothesis, we can thus conclude that $E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a)$.

Proof (of Theorem 3). We shall prove that given any two safety (prefix-closed) properties φ_1 and φ_2, then $\forall \sigma \in \Sigma^*, (E_{\varphi_1} || E_{\varphi_2})(\sigma) = E_{\varphi_1 \cap \varphi_2}(\sigma)$. We prove this using induction on the length of the input word σ.

Induction basis. If $\sigma = \epsilon$, from Definition 1, $E_{\varphi_1}(\epsilon) = \epsilon$ and $E_{\varphi_2}(\epsilon) = \epsilon$. Since $merge(\epsilon, \epsilon) = \epsilon$, we have $(E_{\varphi_1} || E_{\varphi_2})(\epsilon) = \epsilon$. We also have $E_{\varphi_1 \cap \varphi_2}(\sigma) = \epsilon$. For safety properties, Theorem 3 holds when $\sigma = \epsilon$.

Induction step. Assume that for every $\sigma \in \Sigma^*$ of some length $n \in \mathbb{N}$, Theorem 3 holds. That is, $(E_{\varphi_1} || E_{\varphi_2})(\sigma) = E_{\varphi_1 \cap \varphi_2}(\sigma)$, and it is the maximal prefix of σ that belongs to $\varphi_1 \cap \varphi_2$. We now prove that for any $a \in \Sigma$, Theorem 3 holds for $\sigma \cdot a$. We have the following four possible cases:

– Case $\sigma \cdot a \in \varphi_1$ and $\sigma \cdot a \in \varphi_2$.
Let us first examine $E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a)$. Since $\sigma \cdot a \in \varphi_1$ and $\sigma \cdot a \in \varphi_2$, $\sigma \cdot a \in \varphi_1 \cap \varphi_2$. From Definition 1, Theorem 5 (**Tr2**) and Lemma 1, we can derive that $E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a) = \sigma \cdot a$.
Let us now examine $(E_{\varphi_1} || E_{\varphi_2})(\sigma \cdot a)$. Since $\sigma \cdot a \in \varphi_1$, from Definition 1 Theorem 5 (**Tr2**) and Lemma 1, we have $E_{\varphi_1}(\sigma \cdot a) = \sigma \cdot a$. Similarly, we have $E_{\varphi_2}(\sigma \cdot a) = \sigma \cdot a$. Since $merge(\sigma \cdot a, \sigma \cdot a) = \sigma \cdot a$, we have $(E_{\varphi_1} || E_{\varphi_2})(\sigma \cdot a) = \sigma \cdot a$.
Thus, Theorem 3 holds in this case.
– Case $\sigma \cdot a \in \varphi_1$ and $\sigma \cdot a \notin \varphi_2$.
Let us first examine $E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a)$. Since $\sigma \cdot a \notin \varphi_2$, $\sigma \cdot a \notin \varphi_1 \cap \varphi_2$. From Definition 1, we can derive that $E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a) = E_{\varphi_1 \cap \varphi_2}(\sigma)$.
Let us now examine $(E_{\varphi_1} || E_{\varphi_2})(\sigma \cdot a)$. Since $\sigma \cdot a \in \varphi_1$, from Definition 1 Theorem 5 (**Tr2**) and Lemma 1, we have $E_{\varphi_1}(\sigma \cdot a) = \sigma \cdot a$. Since, $\sigma \cdot a \notin \varphi_2$, we can derive that $E_{\varphi_2}(\sigma \cdot a) = E_{\varphi_2}(\sigma)$.
We also know that $E_{\varphi_2}(\sigma) \preccurlyeq \sigma$ (from **Tr1** of Theorem 5), thus $merge(\sigma \cdot a, E_{\varphi_2}(\sigma)) \preccurlyeq E_{\varphi_2}(\sigma)$. Since φ_1 is a safety property, and $\sigma \cdot a \in \varphi_1$, we have $\sigma \in \varphi_1$ and we can derive that $E_{\varphi_1}(\sigma) = \sigma$ (from **Tr2** of Theorem 5).

So, we have $(E_{\varphi_1}||E_{\varphi_2})(\sigma \cdot a) = merge(\sigma \cdot a, E_{\varphi_2}(\sigma)) = merge(\sigma, E_{\varphi_2}(\sigma)) = merge(E_{\varphi_1}(\sigma), E_{\varphi_2}(\sigma)) = (E_{\varphi_1}||E_{\varphi_2})(\sigma)$.
Using induction hypothesis, we have $(E_{\varphi_1}||E_{\varphi_2})(\sigma) = E_{\varphi_1 \cap \varphi_2}(\sigma) = E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a)$. Thus, the theorem holds in this case.

- Case $\sigma \cdot a \notin \varphi_1$ and $\sigma \cdot a \in \varphi_2$. Similar to the previous case.
- Case $\sigma \cdot a \notin \varphi_1$ and $\sigma \cdot a \notin \varphi_2$. Let us first examine $E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a)$. Since $\sigma \cdot a \notin \varphi_1$, and $\sigma \cdot a \notin \varphi_2$, we have $\sigma \cdot a \notin \varphi_1 \cap \varphi_2$. From the definition of the enforcement function, we can derive that $E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a) = E_{\varphi_1 \cap \varphi_2}(\sigma)$.
Let us now examine $(E_{\varphi_1}||E_{\varphi_2})(\sigma \cdot a)$. Since, $\sigma \cdot a \notin \varphi_1$, we can derive that $E_{\varphi_1}(\sigma \cdot a) = E_{\varphi_1}(\sigma)$. Similarly, we can also derive that $E_{\varphi_2}(\sigma \cdot a) = E_{\varphi_2}(\sigma)$. So, we have $(E_{\varphi_1}||E_{\varphi_2})(\sigma \cdot a) = merge(E_{\varphi_1}(\sigma), E_{\varphi_2}(\sigma)) = (E_{\varphi_1}||E_{\varphi_2})(\sigma)$.
Using induction hypothesis we can conclude that $(E_{\varphi_1}||E_{\varphi_2})(\sigma \cdot a) = E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a)$.

Proof (of Theorem 4 - item 1 (serial composition)). We shall prove that given any two co-safety properties φ_1 and φ_2, $\forall \sigma \in \Sigma^*, E_{\varphi_1}(E_{\varphi_2}(\sigma)) = E_{\varphi_2}(E_{\varphi_1}(\sigma)) = E_{\varphi_1 \cap \varphi_2}(\sigma)$. We prove this using induction on the length of the input word σ.

Induction basis. If $\sigma = \epsilon$, from Definition 1, $E_{\varphi_1}(\sigma) = \epsilon$ and $E_{\varphi_2}(\sigma) = \epsilon$. Thus, $E_{\varphi_2}(E_{\varphi_1}(\sigma)) = E_{\varphi_1}(E_{\varphi_2}(\sigma)) = \epsilon$. We also have $E_{\varphi_1 \cap \varphi_2}(\sigma) = \epsilon$. Item 1 of Theorem 4 trivially holds.

Induction step. Assume that item 1 of Theorem 4 holds for every $\sigma \in \Sigma^*$ of some length $n \in \mathbb{N}$. That is, $E_{\varphi_1}(E_{\varphi_2}(\sigma)) = E_{\varphi_2}(E_{\varphi_1}(\sigma)) = E_{\varphi_1 \cap \varphi_2}(\sigma)$, and it is the maximal prefix of σ that satisfies both φ_1 and φ_2. We now prove that for any $a \in \Sigma$, item 1 of Theorem 4 holds for $\sigma \cdot a$. We have the following four possible cases:

- Case $\sigma \cdot a \in \varphi_1$ and $\sigma \cdot a \in \varphi_2$.
Let us first examine $E_{\varphi_1}(E_{\varphi_2}(\sigma \cdot a))$. Since $\sigma \cdot a \in \varphi_2$, from Definition 1, using Theorem 5 (**Tr2**) and Lemma 1, $E_{\varphi_2}(\sigma \cdot a) = \sigma \cdot a$. Thus, $E_{\varphi_1}(E_{\varphi_2}(\sigma \cdot a)) = E_{\varphi_1}(\sigma \cdot a)$. Since $\sigma \cdot a \in \varphi_1$, again from Definition 1, using Theorem 5 (**Tr2**) and Lemma 1, we have $E_{\varphi_1}(\sigma \cdot a) = \sigma \cdot a$. We can similarly show that $E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = \sigma \cdot a$. Since $\sigma \cdot a \in \varphi_1 \cap \varphi_2$, from Definition 1, using Theorem 5 (**Tr2**) and Lemma 1, we also have $E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a) = \sigma \cdot a$.
We thus have $E_{\varphi_1}(E_{\varphi_2}(\sigma \cdot a)) = E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a) = \sigma \cdot a$.
- Case $\sigma \cdot a \in \varphi_1$ and $\sigma \cdot a \notin \varphi_2$.
Let us first examine $E_{\varphi_1}(E_{\varphi_2}(\sigma \cdot a))$. Since $\sigma \cdot a \notin \varphi_2$, according to Definition 1, $E_{\varphi_2}(\sigma \cdot a) = E_{\varphi_2}(\sigma)$. Thus, $E_{\varphi_1}(E_{\varphi_2}(\sigma \cdot a)) = E_{\varphi_1}(E_{\varphi_2}(\sigma))$. Note that since φ_2 is a co-safety property, and $\sigma \cdot a \notin \varphi_2$, we can in fact also show that $E_{\varphi_2}(\sigma) = \epsilon$, and thus $E_{\varphi_1}(E_{\varphi_2}(\sigma)) = \epsilon$.
Let us now examine $E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a))$. Since $\sigma \cdot a \in \varphi_1$, from Definition 1, Theorem 5 (**Tr2**) and Lemma 1, we have $E_{\varphi_1}(\sigma \cdot a) = \sigma \cdot a$. Thus we have $E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = E_{\varphi_2}(\sigma \cdot a)$. Since $\sigma \cdot a \notin \varphi_2$, from Definition 1, $E_{\varphi_2}(\sigma \cdot a) = E_{\varphi_2}(\sigma)$. Since φ_2 is a co-safety property, and $\sigma \cdot a \notin \varphi_2$, we can show that $E_{\varphi_2}(\sigma \cdot a) = \epsilon$. Thus $E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = \epsilon$.
We showed that $E_{\varphi_1}(E_{\varphi_2}(\sigma \cdot a)) = E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = \epsilon$. Regarding $E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a)$, we know that $\varphi_1 \cap \varphi_2$ (since both φ_1 and φ_2 are co-safety) is a co-safety property. Since $\sigma \cdot a \notin \varphi_1 \cap \varphi_2$, we can show that $E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a) = \epsilon$.
- Case $\sigma \cdot a \notin \varphi_1$ and $\sigma \cdot a \in \varphi_2$.
Similar to the previous case.
- Case $\sigma \cdot a \notin \varphi_1$ and $\sigma \cdot a \notin \varphi_2$.
Let us first examine $E_{\varphi_1}(E_{\varphi_2}(\sigma \cdot a))$. Since $\sigma \cdot a \notin \varphi_2$, according to the definition of the enforcement function, $E_{\varphi_2}(\sigma \cdot a) = E_{\varphi_2}(\sigma)$. Thus, $E_{\varphi_1}(E_{\varphi_2}(\sigma \cdot a)) =$

$E_{\varphi_1}(E_{\varphi_2}(\sigma))$. Similarly, we can show that $E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = E_{\varphi_2}(E_{\varphi_1}(\sigma))$. Since $\sigma \cdot a \notin \varphi_1 \cap \varphi_2$, we have $E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a) = E_{\varphi_1 \cap \varphi_2}(\sigma)$.
Using induction hypothesis, we can thus conclude that $E_{\varphi_1}(E_{\varphi_2}(\sigma \cdot a)) = E_{\varphi_2}(E_{\varphi_1}(\sigma \cdot a)) = E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a)$.

Proof (of Theorem 4 - item 2 (parallel composition)). We shall prove that given any two co-safety (extension-closed) properties φ_1 and φ_2, then $\forall \sigma \in \Sigma^*, (E_{\varphi_1} || E_{\varphi_2})(\sigma) = E_{\varphi_1 \cap \varphi_2}(\sigma)$. We prove this using induction on the length of the input word σ.

Induction basis. Similar to induction basis in Proof of Theorem 3.

Induction step. Assume that for every $\sigma \in \Sigma^*$ of some length $n \in \mathbb{N}$, item 2 of Theorem 4 holds. That is, $(E_{\varphi_1} || E_{\varphi_2})(\sigma) = E_{\varphi_1 \cap \varphi_2}(\sigma)$. We now prove that for any $a \in \Sigma$, item 2 of Theorem 4 holds for $\sigma \cdot a$. We have the following four possible cases:

- Case $\sigma \cdot a \in \varphi_1$ and $\sigma \cdot a \in \varphi_2$.
  Similar to the first case in Proof A, using Theorem 5 (**Tr2**) and Lemma 1, we can derive that $E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a) = (E_{\varphi_1} || E_{\varphi_2})(\sigma \cdot a) = \sigma \cdot a$.
- Case $\sigma \cdot a \in \varphi_1$ and $\sigma \cdot a \notin \varphi_2$.
  Let us first examine $E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a)$. Since $\sigma \cdot a \notin \varphi_2$, $\sigma \cdot a \notin \varphi_1 \cap \varphi_2$. From Definition 1, we can derive that $E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a) = E_{\varphi_1 \cap \varphi_2}(\sigma)$. Since φ_1 and φ_2 are co-safety properties, $\varphi_1 \cap \varphi_2$ is a co-safety property and since $\sigma \cdot a \notin \varphi_1 \cap \varphi_2$, $\forall \sigma' \preccurlyeq \sigma \cdot a, \sigma' \notin \varphi_1 \cap \varphi_2$. We can thus derive that $E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a) = \epsilon$.
  Let us now examine $(E_{\varphi_1} || E_{\varphi_2})(\sigma \cdot a)$. Since $\sigma \cdot a \in \varphi_1$, from Definition 1, Theorem 5 (**Tr2**) and Lemma 1, we can derive that $E_{\varphi_1}(\sigma \cdot a) = \sigma \cdot a$. Since, $\sigma \cdot a \notin \varphi_2$, we can derive that $E_{\varphi_2}(\sigma \cdot a) = E_{\varphi_2}(\sigma)$. Since φ_2 is a co-safety property and $\sigma \cdot a \notin \varphi_2, \forall \sigma' \preccurlyeq \sigma \cdot a$, we have $\sigma' \notin \varphi_2$ and we can thus derive that $E_{\varphi_2}(\sigma \cdot a) = \epsilon$. Thus $merge(\sigma \cdot a, \epsilon) = \epsilon$.
  Since $(E_{\varphi_1} || E_{\varphi_2})(\sigma \cdot a) = E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a) = \epsilon$, the theorem holds in this case.
- Case $\sigma \cdot a \notin \varphi_1$ and $\sigma \cdot a \in \varphi_2$. Similar to the previous case.
- Case $\sigma \cdot a \notin \varphi_1$ and $\sigma \cdot a \notin \varphi_2$. Let us first examine $E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a)$. Since $\sigma \cdot a \notin \varphi_1$, and $\sigma \cdot a \notin \varphi_2$, we have $\sigma \cdot a \notin \varphi_1 \cap \varphi_2$. From the definition of the enforcement function, we can derive that $E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a) = E_{\varphi_1 \cap \varphi_2}(\sigma)$.
  Let us now examine $(E_{\varphi_1} || E_{\varphi_2})(\sigma \cdot a)$. Since, $\sigma \cdot a \notin \varphi_1$ and φ_1 is a co-safety property, we can derive that $E_{\varphi_1}(\sigma \cdot a) = E_{\varphi_1}(\sigma)$. Similarly we can also derive that $E_{\varphi_2}(\sigma \cdot a) = E_{\varphi_2}(\sigma)$.
  Using induction hypothesis, we can conclude that $(E_{\varphi_1} || E_{\varphi_2})(\sigma \cdot a) = E_{\varphi_1 \cap \varphi_2}(\sigma \cdot a)$.