Using Equivalence Relations for Corrective Enforcement of Security Policies

Raphael Khoury and Nadia Tawbi

Department of Computer Science and Software Engineering, Laval University, 1065, ave de la Mdecine, Qubec (QC), Canada G1V 0A6.
raphael.khoury.1@ulaval.ca, Nadia.Tawbi@ift.ulaval.ca

Abstract. In this study, we present a new framework of runtime enforcement of security policies. Building on previous studies, we examine the enforcement power of monitors capable of transforming their target's execution. We bound this ability by a restriction stating that any transformation must preserve equivalence between the monitor's input and output. We proceed by giving examples of meaningful equivalence relations and identify the security policies that are enforceable with their use. We also relate our work to previous findings in this field. Finally, we investigate how an a priori knowledge of the target program's behavior would increase the monitor's enforcement power.

Keywords: Monitoring, Security Policy Enforcement, Program Transformation, inlined reference monitors

1 Introduction

In light of the increasing complexity and interconnectivity of modern software, there is a growing realization that formal security frameworks are needed to ensure code safety. Because they have solid theoretical underpinnings, such framework can provide assurances that the desired security policy will be enforced regardless of the target program's output. One such formal security framework, which has gained wide acceptance in recent years is runtime monitoring. This approach to code safety seeks to allow an untrusted code to run safely by observing its execution and reacting if need be to prevent a potential violation of a user-supplied security policy.

Several studies have focused on establishing the set of security policies that are enforceable by monitors operating under various constraints. This is necessary to best select the appropriate enforcement mechanism given the desired security policy and enforcement context. In this study, we take this framework one step further and examine the enforcement power of monitors capable of transforming their input. However, the monitor's ability to do so must be constrained by a requirement to maintain an equivalence between input and output. This intuitively corresponds to an enforcement paradigm, closer to one that would be encountered in practice, in which the actions taken by the monitor are constrained by a limitation that certain behaviors present in the original sequence be preserved.

The question of identifying the set of security policies (termed properties) enforceable by monitors capable of transforming their input was raised several times in the literature [14, 4, 13, 10]. While these studies observe that this ability considerably extends the monitor's enforcement power, they do not to provide a more specific characterization of the set of enforceable properties w.r.t equivalence relations other than syntactic equality. This results from the lack of a framework constraining the ability of a monitor to transform its input. This point is concisely explained by Ligatti et. al. in [13]. "*A major difficulty with semantic equivalence is its generality: for any reasonable property $\hat{\mathcal{P}}$ there exists a sufficiently helpful equivalence relation that enables a security automaton to enforce $\hat{\mathcal{P}}$*".

Indeed, the authors go on to note that if all valid sequences can be thought of as being equivalent to one another, any security policy can be enforced simply by always outputting the same sequence. This strictly meets the definition of enforcement but does not provide a meaningful enforcement of the desired policy.

In this paper, we suggest a framework to study the enforcement power of monitors. The key insight behind our work is to state certain criteria which must be met for an equivalence relation to be useful in monitoring. We then give two examples of such equivalence relations, and show which security properties are enforceable with their use.

The contributions of this paper are as follows: First, we develop a framework of enforcement, termed corrective$_{\cong}$ enforcement to reason about the enforcement power of monitors bounded to produce an output which is semantically equivalent to their input with respect to some equivalence relation $\cong$. We suggest two possible examples of such relations and give the set of enforceable security policies as well as examples of real policies for each. Finally, we show that the set of enforceable properties defined in [13] for effective enforcement can be considered as special cases of our more general framework.

The remainder of this paper is organized as follows. Section 2 presents a review of related work. In Section 3, we define some concepts and notations that are used throughout the paper. In Section 4, we show under what conditions equivalence relations can be used to transform sequences and ensure the respect of the security policy. The set of security policies which can be enforced in this manner is examined in Section 5. In Section 6, we give two examples of possible equivalence relations and show that they can serve as the basis for the enforcement of meaningful security properties. In section 7, we investigate how an a priori knowledge of the target program's behavior would increase the monitor's enforcement power. Concluding remarks and avenues for future work are laid out in Section 8.

2 Related work

Schneider, in his seminal work [14], was the first to investigate the question of which security policies could be enforced by monitors. He focused on specific classes of monitors, which observe the execution of a target program with no knowledge of its possible future behavior and with no ability to affect it, except by aborting the execution. Under these conditions, he found that a monitor could enforce the precise security policies that are identified in the literature as *safety* properties, and are informally characterized by prohibiting a certain bad thing from occurring in a given execution.

Schneider's study also suggested that the set of properties enforceable by monitors could be extended under certain conditions. Building on this insight, Ligatti, Bauer and Walker [4, 12] examined the way the set of policies enforceable by monitors would be extended if the monitor had some knowledge of its target's possible behavior or if its ability to alter that behavior were increased. The authors modified the above definition of a monitor along three axes, namely (1) the means at the disposal of the monitor in order to respond to a possible violation of the security policy; (2) whether the monitor has access to information about the program's possible behavior; and (3) how strictly the monitor is required to enforce the security policy. Consequently, they were able to provide a rich taxonomy of classes of security policies, associated with the appropriate model needed to enforce them. Several of these models are strictly more powerful than the security automata developed by Schneider and are used in practice.

Evolving along this line of inquiry, Ligatti et al. [13] gave a more precise definition of the set of properties enforceable by the most powerful monitors, while Fong [9] and Talhi et al. [16] expounded on the capabilities of monitors operating under memory constraints. Hamlen et al. [10] , on the other hand, showed that in-lined monitors (whose operation is injected into the target program's code, rather than working in parallel) can also enforce more properties than those modeled by a security automaton. In [3], a method is given to enforce both safety and *co-safety* properties by monitoring. The set of properties enforceable by monitors aided by static analysis of the program is examined in [6, 7]. In [5], Bielova et al. delineate the set of properties enforceable by a monitor limited to suppressing a finite subsequence of the execution before either outputting or deleting them.

3 Preliminaries

Let us briefly start with some preliminary definitions.

Executions are modeled as sequences of atomic actions taken from a finite or countably infinite set of actions Σ. The empty sequence is noted ϵ, the set of all finite length sequences is noted Σ^*, that of all infinite length sequences is noted Σ^ω, and the set of all possible sequences is noted $\Sigma^\infty = \Sigma^\omega \cup \Sigma^*$. Likewise, for a set of sequences $\mathcal{S}$, $\mathcal{S}^*$ denote the finite iterations of sequences of $\mathcal{S}$ and $\mathcal{S}^\omega$ that of infinite iterations, and $\mathcal{S}^\infty = \mathcal{S}^\omega \cup \mathcal{S}^*$. Let $\tau \in \Sigma^*$ and $\sigma \in \Sigma^\infty$ be two sequences of actions. We write $\tau;\sigma$ for the concatenation of τ and σ. We say that τ is a prefix of σ noted $\tau \preceq \sigma$,or equivalently $\sigma \succeq \tau$ *iff* there exists a sequence σ' such that $\tau;\sigma' = \sigma$. We write $\tau \prec \sigma$ (resp. $\sigma \succ \tau$) for $\tau \preceq \sigma \wedge \tau \neq \sigma$ (resp. $\sigma \succeq \tau \wedge \tau \neq \sigma$). Finally, let $\tau, \sigma \in \Sigma^\infty$, τ is said to be a suffix of σ iff there exists a $\sigma' \in \Sigma^*$ s.t. $\sigma = \sigma';\tau$.

We denote by $pref(\sigma)$ (resp. $suf(\sigma)$) the set of all prefixes (resp. suffixes) of σ. Let $A \subseteq \Sigma^\infty$ be a subset of sequences. Abusing the notation, we let $pref(A)$ (resp. $suf(A)$)stands for $\bigcup_{\sigma \in A} pref(\sigma)$ (resp. $\bigcup_{\sigma \in A} suf(\sigma)$). The i^{th} action in a sequence σ is given as σ_i, σ_1 denotes the first action σ, $\sigma[i,j]$ denotes the sequence occurring between the i^{th} and j^{th} actions of σ, and $\sigma[i,..]$ denotes the remainder of the sequence, starting from action σ_i. The length of a sequence $\tau \in \Sigma^*$ is given as $|\tau|$.

A multiset, or bag [15] is a generalization of a set in which each element may occur multiple times. A multiset $\mathcal{A}$ can be formally defined as a pair $\langle A, f\rangle$ where A is a set and $f : A \rightarrow \mathbb{N}$ is a function indicating the number of occurrences of each element of A in $\mathcal{A}$. Note that $a \notin A \Leftrightarrow f(a) = 0$. Thus, by using this insight, to define basic operations on multisets one can consider a universal set A and different functions of type $A \rightarrow \mathbb{N}$ associated with it to form different multisets.

Given two multisets $\mathcal{A} = \langle A, f\rangle$ and $\mathcal{B} = \langle A, g\rangle$,the multiset union $\mathcal{A} \cup \mathcal{B} = \langle A, h\rangle$ where $\forall a \in A : h(a) = f(a) + g(a)$. Furthermore, $\mathcal{A} \subseteq \mathcal{B} \Leftrightarrow \forall a \in A : f(a) \leq g(a)$. The removal of an element $a \in A$ from multiset $\mathcal{A}$ is done by updating function f so that $f(a) = max(f(a) - 1, 0)$.

Finally, a security policy $P \subseteq \Sigma^\infty$ is a set of allowed executions. A policy P is a property iff there exists a decidable predicate $\hat{\mathcal{P}}$ over the executions of Σ^∞ s.t. $\sigma \in P \Leftrightarrow \hat{\mathcal{P}}(\sigma)$. In other words, a property is a policy for which the membership of any sequence can be determined by examining only the sequence itself. Such a sequence is said to be valid or to respect the property. Since all policies enforceable by monitors are properties, we use $\hat{\mathcal{P}}$ to refer to policies and their characteristic predicate interchangeably. Properties for which the empty sequence ϵ is a member are said to be *reasonable*.

A number of classes of properties have been defined in the literature and are of special interest in the study of monitoring. First are safety properties [11], which proscribe that certain "bad things" occur during the execution. Let Σ be a set of actions and $\hat{\mathcal{P}}$ be a property, $\hat{\mathcal{P}}$ is a safety property iff

$$\forall \sigma \in \Sigma^\infty : \neg\hat{\mathcal{P}}(\sigma) \Rightarrow \exists \sigma' \preceq \sigma : \forall \tau \succeq \sigma' : \neg\hat{\mathcal{P}}(\tau) \qquad \text{(safety)}$$

Alternatively, a *liveness* property [1] is a property prescribing that a certain "good thing" must occur in any valid execution. Formally, for an action set Σ^∞ and a property $\hat{\mathcal{P}}$, $\hat{\mathcal{P}}$ is a liveness property iff

$$\forall \sigma \in \Sigma^* : \exists \tau \in \Sigma^\infty : \tau \succeq \sigma \wedge \hat{\mathcal{P}}(\tau) \qquad \text{(liveness)}$$

Any property can be stated as the conjunction of a safety property and a liveness property [2]. Another relevant set of properties is that of infinite renewal properties (*renewal*), defined in [13] to characterize the set properties enforceable by edit-automata monitors using syntactic equality as the equivalence relation. A property is member of this set if every infinite valid sequence has infinitely many valid prefixes, while every invalid infinite sequence has only finitely many such prefixes. Formally, for an action set Σ and a property $\hat{\mathcal{P}}$, $\hat{\mathcal{P}}$ is a renewal property iff it meets the following two equivalent conditions

$$\forall \sigma \in \Sigma^\omega : \hat{\mathcal{P}}(\sigma) \Leftrightarrow \{\sigma' \preceq \sigma | \hat{\mathcal{P}}(\sigma')\} \textit{is an infinite set} \qquad (\text{renewal}_1)$$

$$\forall \sigma \in \Sigma^\omega : \hat{\mathcal{P}}(\sigma) \Leftrightarrow (\forall \sigma' \preceq \sigma : \exists \tau \preceq \sigma : \sigma' \preceq \tau \wedge \hat{\mathcal{P}}(\tau)) \qquad (\text{renewal}_2)$$

Note that the definition of renewal imposes no restrictions on the finite sequences in $\hat{\mathcal{P}}$. For infinite sequences, the set of renewal properties includes all safety properties, some liveness properties and some properties which are neither safety nor liveness.

Finally, we formalize the the set of *transactional* properties, suggested in [13], which will be of use in section 6.1. A transactional property is one in which any valid sequence consists of a concatenation of valid finite transactions. Such properties can model, for example, the behavior of systems which repeatedly interacts with clients using a well defined protocol, such as a system managing the allocation of resource or the access to a database. Let Σ be an action set and $T \subseteq \Sigma^*$ be a subset of finite transactions, $\hat{\mathcal{P}}_T$ is a transactional property over set T iff

$$\forall \sigma \in \Sigma^\infty : \hat{\mathcal{P}}_T(\sigma) \Leftrightarrow \sigma \in T^\infty \qquad (\text{transactional})$$

This definition is subtly different, and indeed forms a subset, to that of iterative properties defined in [5]. transactional properties also form a subset to the set of renewal properties, and include some but not all safety properties, liveness properties as well as properties which are neither.

4 Monitoring with Equivalence Relations

The idea of using equivalence relations to transform execution sequences was first suggested in [10]. The equivalence relations are restricted to those that are *consistent* with the security policy under consideration. Let $\hat{\mathcal{P}}$ be a security policy, the consistency criterion for an equivalence relation $\cong$ is given as:

$$\forall \sigma, \sigma' \in \Sigma^\infty : \sigma \cong \sigma' \Rightarrow \hat{\mathcal{P}}(\sigma) \Leftrightarrow \hat{\mathcal{P}}(\sigma'). \qquad (\text{consistency})$$

Yet, upon closer examination, this criterion seems too restrictive for our purposes. If any two equivalent sequences always meet this criterion, an invalid prefix can never be made valid by replacing it with another equivalent one. It is thus impossible to “correct” an invalid prefix and output it.

It is still necessary to impose some restrictions on equivalence relations and their relation to properties. Otherwise, as discussed above, any property would be enforceable, but not always in a meaningful manner.

In this paper, we suggest the following alternative framework.

Following previous work in monitoring by Fong [9], we use an abstraction function $\mathcal{F} : \Sigma^* \rightarrow \mathcal{I}$, to capture the property of the input sequence which the monitor must preserve throughout its manipulation. While Fong made use of abstractions to reduce the overhead of the monitor, we use them as the basis for our equivalence relations. Such an abstraction can capture any property of relevance. This may be, for example, the presence of certain subwords or factors or any other semantic property of interest. We expect the property to be consistent with this abstraction rather than with the equivalence relation itself. Formally:

$$\mathcal{F}(\sigma) = \mathcal{F}(\sigma') \Rightarrow \hat{\mathcal{P}}(\sigma) \Leftrightarrow \hat{\mathcal{P}}(\sigma') \tag{4.1}$$

Furthermore, we restrict ourselves to equivalence relations which group together sequences for which the abstraction is similar. To this end, we let $\leq$ stand for some partial order over the values of $\mathcal{I}$. We define $\sqsubseteq$ as the partial order defined as $\forall \sigma, \sigma' \in \Sigma^* : \sigma \sqsubseteq \sigma' \Leftrightarrow \mathcal{F}(\sigma) \leq \mathcal{F}(\sigma')$. We equivalently write $\sigma' \sqsupseteq \sigma$ and $\sigma \sqsubseteq \sigma'$.

The transformation performed by the monitor on a given sequence τ produces a new sequence τ' s.t. $\tau' \sqsubseteq \tau$. To ease the monitor's task in finding such a suitable replacement, we impose the following two constraints on the equivalence relations used in monitoring.

First, if two sequences are equivalent, any intermediary sequence over $\sqsubseteq$ is also equivalent to them.

$$\sigma \sqsubseteq \sigma' \sqsubseteq \sigma'' \wedge \sigma \cong \sigma'' \Rightarrow \sigma \cong \sigma' \tag{4.2}$$

Second, two sequences cannot be equivalent if they do not share a common greatest lower bound.Conversely, the greatest lower bound of two equivalent sequences is also equivalent to them. These last two criteria are stated together as:

$$\forall \sigma, \sigma' \in \Sigma^* : \sigma \cong \sigma' \Rightarrow \exists \tau \in \Sigma^* : \tau = (\sigma \sqcap \sigma') \wedge \tau \cong \sigma \tag{4.3}$$

where $(\sigma \sqcap \sigma') = \tau$ s.t. $\tau \sqsubseteq \sigma \wedge \tau \sqsubseteq \sigma' \wedge \neg\exists \tau' \sqsupseteq \tau : \tau' \sqsubseteq \sigma \wedge \tau' \sqsubseteq \sigma'$

The intuition behind the above two restrictions, is that, if an equivalence restriction meets these two criteria, a monitor looking for an valid sequence equivalent to an invalid input simply has to iteratively perform a certain transformation until such a sequence is found or until every equivalent sequence has been examined.

We define our equivalence relations over finite sequences. Two infinite sequences are equivalent, iff they have infinitely many valid equivalent prefixes.

Let $\cong$ be an equivalence relation over the sequences of Σ^*

$$\forall \sigma, \sigma' \in \Sigma^\omega : \sigma \cong \sigma' \Leftrightarrow \forall \tau \prec \sigma : \exists \upsilon \succeq \tau : \exists \tau' \prec \sigma' : \upsilon \cong \tau' \tag{4.4}$$

It is easy to see that an equivalence between infinite sequence not meeting this criterion would be of no use to a monitor, which is bound to transform its input in finite time.

Finally, we impose the following closure restriction:

$$\tau \cong \tau' \Rightarrow \tau; \sigma \cong \tau'; \sigma \tag{4.5}$$

This may, at first sight, seem like an extremely restrictive condition to be imposed but in fact every meaningful relation that we examined has this property.

Furthermore, no security property can be enforced using an equivalence relation lacking this property. Consider for example what would happen if a monitor is presented with an invalid prefix τ of a longer input sequence for which there exists a valid equivalent sequence τ'. It would be natural for the monitor to transform τ into τ'. Yet it would also be possible that the full original sequence $\sigma \succ \tau$ be actually valid, but that there exists no equivalent sequence for which τ' is a prefix.

In fact, $\sqsubseteq$ organizes the sequences according to some semantic framework, using values given by an abstraction function $\mathcal{F}$, $\hat{\mathcal{P}}$ establishes that only certain values of $\mathcal{F}$ are valid or that a certain threshold must be reached, while $\cong$ groups the sequences if their abstractions are equivalent. In section 6, we give examples that show how the framework described in this section can be used to model desirable security properties of programs and meaningful equivalence relations between their executions.

5 Corrective Enforcement

In this section, we present the automata-based model used to study the enforcement mechanism, and give a more formal definition of our notion of enforcement.

The edit automaton [4, 13] is the most general model of a monitor. It captures the behavior of a monitor capable of inserting or suppressing any action, as well as halting the execution in progress.

Definition 1. *An edit automaton is a tuple $\langle \Sigma, Q, q_0, \delta \rangle$ where*[1]*:*

- *Σ is a finite or countably infinite set of actions;*
- *Q is a finite or countably infinite set of states;*
- *$q_0 \in Q$ is the initial state;*
- *$\delta : (Q \times \Sigma) \rightarrow (Q \times \Sigma^\infty)$ is the transition function, which, given the current state and input action, specifies the automaton's output and successor state. At any step, the automaton may accept the action and output it intact, suppress it and move on to the next action, outputting nothing, or output some other sequence in Σ^∞. If at a given state the transition for a given action is undefined, the automaton aborts.*

Let $\mathcal{A}$ be an edit automaton, we let $\mathcal{A}(\sigma)$ be the output of $\mathcal{A}$ when its input is σ.

Most studies on this topic have focused on effective enforcement. A mechanism effectively enforces a security property iff it respects the two following principles, from [4]:

1. *Soundness* : All output must respect the desired property.

[1] This definition, taken from [16], is equivalent to the one given in [4].

2. $Transparency$: The semantics of executions which already respect the property must be preserved. This naturally requires the use of an equivalence relation, stating when one sequence can be substituted for another.

Definition 2. *Let $\mathcal{A}$ be an edit automaton. $\mathcal{A}$ effectively$_{\cong}$ enforces the property $\hat{\mathcal{P}}$ iff $\forall \sigma \in \Sigma^{\infty}$*

1. $\hat{\mathcal{P}}(\mathcal{A}(\sigma))$ (i.e. $\mathcal{A}(\sigma)$ is valid)
2. $\hat{\mathcal{P}}(\sigma) \Rightarrow A(\sigma) \cong \sigma$

In the literature, the only equivalence relation $\cong$ for which the set of effectively$_{\cong}$ enforceable properties has been formally studied is syntactic equality[4]. Yet, effective enforcement is only one paradigm of enforcement which has been suggested. Other enforcement paradigms include precise enforcement[4], all-or-nothing delayed enforcement[5] or conservative enforcement[4].

In this study, we introduce a new paradigm of security property enforcement, termed corrective$_{\cong}$ enforcement. An enforcement mechanism correctively$_{\cong}$ enforces the desired property if every output sequence is both valid and equivalent to the input sequence. This captures the intuition that the monitor is both required to output a valid sequence, and forbidden from altering the semantics of the input sequence. Indeed, it is not always reasonable to accept, as do preceding studies of monitor's enforcement power, that the monitor is allowed to replace an invalid execution with any valid sequence, even ϵ. A more intuitive model of the desired behavior of a monitor would rather require that only minimal alterations be made to an invalid sequence, for instance by releasing a resource or adding an entry in a log. Those parts of the input sequence which are valid, should be preserved in the output, while invalid behaviors should be corrected or removed. It is precisely these corrective behaviors that we seek to model using our equivalence relations. The enforcement paradigm thus ensures that the output is always valid, and that all valid behavior intended by the user in the input, is present in the monitor's output.

Definition 3. *Let $\mathcal{A}$ be an edit automaton. $\mathcal{A}$ correctively$_{\cong}$ enforces the property $\hat{\mathcal{P}}$ iff $\forall \sigma \in \Sigma^{\infty}$*

1. $\hat{\mathcal{P}}(\mathcal{A}(\sigma))$
2. $\mathcal{A}(\sigma) \cong \sigma$

A monitor can correctively$_{\cong}$ enforce a property iff for every possible sequence there exists an equivalent valid sequence which is either finite or has infinitely many valid prefixes, and the transformation into this sequence is computable.

Theorem 1 *A property $\hat{\mathcal{P}}$ is correctively$_{\cong}$ enforceable iff*

1. $\exists \hat{\mathcal{P}}' : \hat{\mathcal{P}}' \subseteq \hat{\mathcal{P}} \wedge \hat{\mathcal{P}}' \subseteq Renewal$

2. $\hat{\mathcal{P}}$ *is reasonable*
3. There exists a computable function $\gamma : \Sigma^\infty \rightarrow \hat{\mathcal{P}}' : \forall \sigma \in \Sigma^\infty : \gamma(\sigma) \cong \sigma$*.*
4. $\forall \sigma' \preceq \sigma : \gamma(\sigma') \preceq \gamma(\sigma)$

Proof. ($\Rightarrow$ direction) By construction of the following automaton.
$\mathcal{A} = \langle \Sigma, Q, q_0, \delta \rangle$ where

- $Q = \Sigma^*$, the sequence of actions seen so far.
- $q_0 = \epsilon$
- The transition function δ is given as $\delta(\sigma, a) = (\sigma; a, \sigma')$, where $\sigma = a'; \tau$ and $\gamma(\sigma; a) = \gamma(\sigma); \sigma'$

  Note that from condition 3 of theorem 1 we have that $\gamma(\sigma; a)$ is always defined, and from condition 4 that it will take the recursive form described above.

The automaton maintains the following invariants INV(q):
At state $q = \sigma$, $\gamma(\sigma)$ has been output so far, this output is valid and equivalent to σ.
The invariant holds initially, as by definition, ϵ is valid and equivalent to itself. An induction can then show that the invariant is preserved by the transition relation.

($\Leftarrow$ direction) Let $\gamma(\sigma)$ be whatever the automaton outputs on input σ. By definition, γ is a computable function. Furthermore, we have that $\hat{\mathcal{P}}(\sigma)$ and $(\sigma) \cong \sigma$.

We need to show that the image of γ is a property $\hat{\mathcal{P}}'$ included in $\hat{\mathcal{P}}$ and in renewal. That the image of γ is a subset of $\hat{\mathcal{P}}$ follows trivially from the assumptions $\forall \sigma \in \Sigma^\infty : \hat{\mathcal{P}}(\mathcal{A}(\sigma))$. Furthermore, were the output not in renewal, it would include valid sequences with only finitely many valid prefixes. Yet, since the automaton's transition function is restricted to outputing finite valid sequences by the requirement that the finite input be equivalent to the output and equation 4.4 , this is impossible. It follows that the image of γ is a subset of $\hat{\mathcal{P}}$ and renewal. It is also easy to see that $\hat{\mathcal{P}}(\epsilon)$, since if it were not the case, a violation would occur even in the absence of any input action. Finally, since γ is applied recursively to every prefix of the input, it is thus unavoidable that $\forall \sigma' \preceq \sigma : \gamma(\sigma') \preceq \gamma(\sigma)$. □

An equivalence relation $\cong$ over a given set Σ^* can be seen as a set of pairs (x, y), with $x, y \in \Sigma^*$. This allows equivalence relations over the same sets to be compared. Relation $\cong_1$ is a refinement of relation $\cong_2$, noted $\cong_1 < \cong_2$ if the set of pairs in $\cong_1$ is a strict subset of those in $\cong_2$.

Theorem 2 *Let* $\cong_1$*,* $\cong_2$ *be two equivalence relations and let enforceable*$_{\cong}$ *stand for the set of properties which are correctively*$_{\cong}$ *enforceable, then we have* $\cong_1 < \cong_2 \Rightarrow enforceable_{\cong_1} \subset enforceable_{\cong_2}$*.*

Proof. It is easy to see that any property which is correctively$_{\cong_1}$ enforceable is also correctively$_{\cong_2}$ enforceable, since every pair of sequence that are equivalent w.r.t. $\cong_1$ are also equivalent w.r.t. $\cong_2$. The property can thus be correctively$_{\cong_2}$ enforced using the same transformation function γ as was used in its correctively$_{\cong_1}$ enforcement.

Let $[\sigma]_{\cong}$ stand for the set of sequences equivalent to σ with respect to relation $\cong$. By assumption, there is a σ s.t. $[\sigma]_{\cong_1} \subset [\sigma]_{\cong_2}$. Let $\hat{\mathcal{P}}$ be the property defined s.t. $\neg\hat{\mathcal{P}}(\tau) \Leftrightarrow \tau \in [\sigma]_{\cong_1}$. This property is not correctively$_{\cong_1}$ enforceable as there exists no valid equivalent sequences which the monitor can output when its input is σ. The property can be correctively$_{\cong_2}$ enforced by outputting a sequence in $[\sigma]_{\cong_2} \backslash [\sigma]_{\cong_1}$ when the input is σ. □

It follows from this theorem that the coarser the equivalence relation used by the monitor is, the greater the set of enforceable$_{\cong}$ properties.

The following lemma is used in setting an upper bound to the set of enforceable properties.

Lemma 3 *Let $\cong$ be an equivalence relation and $\hat{\mathcal{P}}$ be some correctively$_{\cong}$ enforceable property. Then, for all $\hat{\mathcal{P}}$' s.t. $\hat{\mathcal{P}} \subseteq \hat{\mathcal{P}}'$ we have that $\hat{\mathcal{P}}$' is correctively$_{\cong}$ enforceable.*

The monitor has only to simulate it's enforcement of $\hat{\mathcal{P}}$ in order to correctively$_{\cong}$ enforce $\hat{\mathcal{P}}$'.

6 Equivalence Relations

In this section, we consider two examples of the equivalence relation $\cong$, and examine the set of properties enforceable by each.

6.1 Factor equivalence

The first equivalence relation under consideration is factor equivalence. A word $\tau \in \Sigma^*$ is a factor of a word $\omega \in \Sigma^\infty$ if $\omega = \upsilon;\tau;\upsilon'$, with $\upsilon \in \Sigma^*$ and $\upsilon' \in \Sigma^\infty$. Two sequences τ, τ' are factor equivalent, w.r.t. a given set of valid factors $\mathcal{T} \subseteq \Sigma^*$ if they both contain the same multiset of factors from $\mathcal{T}$. We use a multiset rather than simply comparing the set of factors from $\mathcal{T}$ occurring in each sequence so as to be able to distinguish between sequences containing a different number of occurrences of the same subset of factors. This captures the intuition that if certain valid transactions are present in the input sequence, they must still be present in the output sequence, regardless of any other transformation made to ensure compliance with the security property. In this context, the desired behavior of the system can be defined by a multiset of valid transactions. A valid run of this system consists of a finite or infinite sequence of well-formed transactions, while an invalid sequence is a sequence containing malformed or incomplete transactions. One may reasonably consider all sequences exhibiting the same multiset of valid transactions to be equivalent to each other.

Before moving on, let us introduce some definitions, needed to parse the sequence as a concatenation of valid and invalid transactions, with respect to a set of transactions $\mathcal{T}$. Recall that the abstraction function is only applied to finite sequences and that infinite input are treated locally, in concordance with equation 4.4. Let $index_\sigma(\tau)$ be the function returning a list of intervals $l = ([i_1, j_1], ...[i_n, j_n])|\forall[i, j] \in l : \sigma[i, j] = \tau$, ordered such that $\forall[i_n, j_n], [i_m, j_m] \in l : n < m \Rightarrow i_n < i_m \wedge j_n < j_m$. Abusing the notation we write $index_\sigma(S)$ for $S \subseteq \Sigma^*$ for the similarly ordered list of intervals $l = ([i_1, j_1], ...[i_n, j_n])|\forall[i, j] \in l : \exists\tau \in S : \sigma[i, j] = \tau$. If two intervals $[i_n, j_n]$ and $[i_m, j_m]$ overlap,[2], only the interval $[min(i_n, i_m), max(j_n, j_m)]$ is added to l. Let $count_\sigma(\Sigma^*, l)$ where σ is a sequence and l is an ordered list of intervals be a function which returns a multiset $(\Sigma^*, (\rho : \Sigma^* \rightarrow \mathbb{N}))$ s.t. $\forall\tau \in \Sigma^* : \rho(\tau) = x$ iff there exists x intervals $[i, j]$in l s.t. $\sigma[i, j] = \tau$. In effect, $count_\sigma(\Sigma^*, l)$ returns the multiset of the factors present in the interval l over sequence σ. Let l be an ordered list of intervals, $\bar{l}_k$ is the ordered list of intervals excluded from l, and occuring before the position $k + 1$ in the sequence. Formally, if $l = ([i_1..j_1], ...[i_n, j_n])$, $\bar{l}_k = ([1, i_1 - 1], [j_1 + 1, i_2 - 1], ...[j_n + 1, k])$, with empty intervals of the form $[i, j] s.t. j > i$ omitted.

Finally, we can now define the functions $valid_\mathcal{T}(\sigma)$ and $invalid_\mathcal{T}(\sigma)$ which compute multisets of valid and invalid sequences occurring in a given finite sequence σ. The multiset of valid factors is given as $valid_\mathcal{T}(\sigma) = count_\sigma(\Sigma^*, index(\mathcal{T}))$. Conversely, that of invalid factors is $invalid_\mathcal{T}(\sigma) = count_\sigma(\Sigma^*, \overline{index(\mathcal{T})}_{|\sigma|})$.

For example, let $\Sigma = \{\texttt{open}, \texttt{close}, \texttt{log}\}$ be a set of atomic actions and let $\mathcal{T} = \{\texttt{open}; \texttt{log}; \texttt{close}\}$ be the set containing the only allowed transaction. If the input sequence is given as $\sigma = \texttt{log}; \texttt{open}; \texttt{log}; \texttt{close}; \texttt{log}; \texttt{open}; \texttt{close}; \texttt{open}; \texttt{log}; \texttt{close}$, then $valid_\mathcal{T}(\sigma)$ is the multiset containing two instances of the factor `open`; `log`; `close` while $invalid_\mathcal{T}(\sigma)$ is the multiset containing a single instance of each of `log` and of `log`;`open`;`close`.

We can now instantiate the abstraction function $\mathcal{F}$ to the set of transactions occurring in a given sequence σ as $trans_\mathcal{T}(\sigma) = valid_\mathcal{T}(\sigma) \cup invalid_\mathcal{T}(\sigma)$. The partial order $\sqsubseteq$ is instantiated as $\sigma \sqsubseteq \sigma' \Leftrightarrow trans_\mathcal{T}(\sigma) \subseteq trans_\mathcal{T}(\sigma')$. The equivalence relation $\cong_\mathcal{T}$ is also given in terms of $\mathcal{T}$. Two sequences σ, σ' are equivalent iff $\sigma \cong_\mathcal{T} \sigma' \Leftrightarrow valid_\mathcal{T}(\sigma) = valid_\mathcal{T}(\sigma')$.

Intuitively, a sequence is smaller than another on the partial order if it has strictly fewer transactions, and two sequences are equivalent if they share the same valid transactions.

We now turn our attention to the set of properties that are correctively$_{\cong_\mathcal{T}}$ enforceable. Intuitively, a monitor can enforce this property by first suppressing the execution until it has seen a factor in $\mathcal{T}$, at which point the factor is output, while any invalid transaction is suppressed. This method of enforcement is analogous to the one described in [5] as delayed all-or-nothing enforcement. Any sequence output in this manner would preserve all its factors in $\mathcal{T}$, and thus be equivalent to the input sequence, but is composed of a concatenation of factors from $\mathcal{T}$, and hence is valid.

[2] meaning that $i_n < j_n < i_m < j_m$

Let $\mathcal{T} \subseteq \Sigma^*$ be a set of factors and let $\hat{\mathcal{P}}_{\mathcal{T}}$ a transactional property as defined in section 3. Note first that all properties enforceable by this approach are in renewal, as they are formed by a concatenation of valid finite sequences. Also, the property necessarily must be reasonable, (i.e. $\hat{\mathcal{P}}(\epsilon)$) as the monitor will not output anything if the input sequence does not contain any factors in $\mathcal{T}$. Finally, for the property $\hat{\mathcal{P}}_{\mathcal{T}}$ to be correctively$_{\cong_{\mathcal{T}}}$ enforceable in the manner described above, the following restriction, termed unambiguity must be imposed on $\mathcal{T}$:

$$\forall \sigma, \sigma' \in \mathcal{T} : \forall \tau \in pref(\sigma) : \forall \tau' \in suf(\sigma') : \tau \neq \epsilon \wedge \tau' \neq \epsilon \Rightarrow \tau; \tau' \notin \mathcal{T} \tag{unambiguity}$$

To understand why this restriction is necessary, consider what would happen in its absence: it would be possible for the monitor to receive as input a sequence which can be parsed either as the concatenation of some valid transactions, or as a different valid transaction bracketed with invalid factors. That is, let $\sigma_1; \sigma_2 = \tau_1; \sigma_3; \tau_2$ be the monitor's input, with $\sigma_1, \sigma_2, \sigma_3 \in \mathcal{T}$ and $\tau_1, \tau_2 \notin \mathcal{T}$. If the monitor interprets the sequence as a concatenation of the valid transactions σ_1 and σ_2, then it has to preserve both factors in its output. However, if it parses the sequence as $\tau_1; \sigma_3; \tau_2$, then it must output only the equivalence sequence σ_3. Since the two sequences are syntactically identical, the monitor has no information of which to base such a decision.

Theorem 4 *A property $\hat{\mathcal{P}}_{\mathcal{T}}$ correctively$_{\cong_{\mathcal{T}}}$ enforceable if it is transactional, reasonable, and $\mathcal{T}$ is unambiguous.*

Proof. From theorem 1, the property $\hat{\mathcal{P}}$ is correctively$_{\cong}$ enforceable iff there exists a function $\gamma : \Sigma^\infty \rightarrow \hat{\mathcal{P}}'$, where $\hat{\mathcal{P}}$' is a subset of $\hat{\mathcal{P}}$ and is in renewal,$\hat{\mathcal{P}}$ is reasonable, and $\forall \sigma, \sigma' \preceq \sigma \in \Sigma^* : \gamma(\sigma') \preceq \gamma(\sigma) \wedge \gamma(\sigma) \cong \sigma$. We prove this theorem by exhibiting such a function.

Let $\gamma : \Sigma^\infty \rightarrow \hat{\mathcal{P}}_{\mathcal{T}}$. We define γ recursively as follows. $\forall \sigma \in \Sigma^\infty$.

$$\gamma(\sigma) = \begin{cases} \tau; \gamma(\sigma') & \text{if there exists a } \tau \text{ in } \mathcal{T} : \sigma = \tau; \sigma' \\ \gamma(\sigma[2..]) & \text{otherwise} \end{cases}$$

It is easy to show that the image of this function is $\hat{\mathcal{P}}_{\mathcal{T}}$, as all transactions not in $\mathcal{T}$ are deleted. The image is also in renewal as $\mathcal{T}$ is formed by a concatenation of finite sequences, and any infinite valid sequence not in renewal would necessarily have finitely many valid prefixes. Conversely, the output of γ is $\cong_{\mathcal{T}}$ equivalent to its input since only factors not in $\mathcal{T}$ are removed. From the fact that γ is applied to the input iteratively we have $\forall \sigma, \sigma' \in \Sigma^\infty : \sigma' \preceq \sigma : \gamma(\sigma') \preceq \gamma(\sigma)$. Transactional properties are reasonable by definition.

We have only to refer to lemma 3 in order to state a precise upper bound to the set of enforceable properties.

Theorem 5 *A property $\hat{\mathcal{P}}$ is correctively$_{\cong_{\mathcal{T}}}$ enforceable iff $\hat{\mathcal{P}}_{\mathcal{T}} \subseteq \hat{\mathcal{P}}$ and $\mathcal{T}$ is unambiguous.*

Proof. ($\Rightarrow$ direction) Follows directly from theorem 4 and lemma 3.
($\Leftarrow$ direction) We show that every sequence in $\hat{\mathcal{P}}_{\mathcal{T}}$ must be present in any correctively$_{\cong_{\mathcal{T}}}$ enforceable property by contradiction. Let $\sigma \in \hat{\mathcal{P}}_{\mathcal{T}}$ be an input sequence such that $\neg\hat{\mathcal{P}}(\sigma)$. The monitor may not enforce the property by removing or adding a transaction in $\mathcal{T}$ to σ, as the output would no longer be equivalent to the input. To understand why the monitor would be incapable of outputing a valid sequence containing exactly the same transactions as σ, even if one such sequence exists consider the following. Let $\sigma' \cong_{\mathcal{T}} \sigma \wedge \hat{\mathcal{P}}(\sigma')$. Let τ be the longest common prefix of σ and σ'. For it to be possible that σ' be both $\in \hat{\mathcal{P}}_{\mathcal{T}}$ and equivalent to σ, there must be at least two sequences $\tau', \tau'' \in \mathcal{T}$ which are appended to τ in a different order to produce σ and σ'. Without loss of generality, let τ' occur first in σ and second in σ'. Let the input sequence be the invalid sequence $\tau;\tau';(\upsilon)^*$ where υ is some invalid transaction. After having output τ, the monitor may not output τ' as this would result in an invalid sequence if the following valid transactions in the input occur in the same order as they do in σ. Yet if it does not output τ', the output sequence is not $\cong_{\mathcal{T}}$ equivalent to the input.

Likewise, let $\mathcal{T}$ not be unambiguous. By assumption there exists valid sequences $\sigma_1, \sigma_2, \sigma_3 \in \mathcal{T}$ and invalid sequences $\tau, \tau' \notin \mathcal{T}$ s.t. $\tau;\sigma_3;\tau' = \sigma_1;\sigma_2$ and $\sigma_1, \sigma_2, \sigma_3$ are valid transactions. Let the infinite sequence $\sigma_1;\upsilon;\sigma_2;\upsilon;\sigma_1;\upsilon...$ be the input sequence, where υ is an invalid transaction (possibly τ or τ'). There cannot be a valid equivalent sequence since any concatenation $\sigma_1;\sigma_2$ also contains the transaction σ_3. Such a property is thus unenforceable.

6.2 Prefix Equivalence

In this section, we show that Ligatti et al.'s result from [13], namely that the set of properties effectively$_=$ enforceable by an edit automaton corresponds to the set of reasonable renewal properties with a computability restriction added[3], can be stated as a special case of our framework.

First, we need to align our definitions of enforcement. Using effective enforcement, they only require that the monitor's output be equivalent to its input when the latter is valid, and while placing no such restriction on the output otherwise. The semantics of their monitor however, do impose that the output remain a prefix of the input in all cases, and indeed, that the longest valid prefix always be output (see [8]). This characterization can be translated in our formalism by instantiating $\cong$ to $\cong_{\preceq} \overset{def}{=} \forall \sigma, \sigma' \in \Sigma^* : \sigma \cong_{\preceq} \sigma' \Leftrightarrow pref(\sigma) \cap \hat{\mathcal{P}} = pref(\sigma') \cap \hat{\mathcal{P}}$. Using this relation, two sequences are equivalent, w.r.t. a given property $\hat{\mathcal{P}}$ iff they have the same set of valid prefixes.

[3] Actually, the authors identified a corner case in which a property not in the set described above. This occurs when the monitor reaches a point where only one valid continuation is possible. The input can then be ignored and this single continuation is output. We have neglected to discuss this case here as it adds comparatively little to the range of enforceable properties.

Theorem 6 *A property $\hat{\mathcal{P}}$ is effectively$_=$ enforceable iff it is correctively$_{\cong_\preceq}$ enforceable.*

Proof. ($\Rightarrow$ direction) From [13], we have that a property is effectively$_=$ enforceable iff 1)$\hat{\mathcal{P}}(\sigma) \Rightarrow \mathcal{A}(\sigma) = \sigma$ and 2) $\hat{\mathcal{P}}(\mathcal{A}(\sigma))$. Conditions 2 is present in identical form in the definition of corrective enforcement. For condition 1, we must consider two cases. If $\hat{\mathcal{P}}(\sigma)$, then it is trivial to show that $\sigma \cong_\preceq \mathcal{A}(\sigma)$, since both sequences are syntactically identical. Otherwise, the semantics of the enforcement mechanism described in [13] ensure that the longest valid prefix is output. It follows that $pref(\sigma) \cap \hat{\mathcal{P}} = pref(\mathcal{A}(\sigma)) \cap \hat{\mathcal{P}}$ and from the definition of $\cong_\preceq$, that $\sigma \cong_\preceq \mathcal{A}(\sigma)$.
($\Leftarrow$ direction) We must show that $\sigma \cong_\preceq \mathcal{A}(\sigma) \wedge \hat{\mathcal{P}}(\sigma) \Rightarrow \mathcal{A}(\sigma) = \sigma$. It is sufficient to observe that if a sequence σ is valid, there can exist no $\sigma' \prec \sigma : \sigma' \cong_\preceq \sigma$. Since the enforcement mechanism described above only outputs a sequence that is prefix or equal to its input we have that $\sigma \cong_\preceq \mathcal{A}(\sigma)$.

It would be intuitive to instantiate the partial order $\sqsubseteq$ to $\preceq$. Other possibilities can be considered, which would more closely follow the specific property being enforced.

Theorem 7 *A property $\hat{\mathcal{P}}$ is correctively$_{\cong_\preceq}$ enforceable iff it is in renewal, reasonable and computable.*

Proof. Immediate from theorem 6 and theorem 3 of [13].

As discussed in [13], this set includes a wide range of properties, including all safety properties, some liveness properties such as the "eventually audits" properties requiring that an action eventually be logged, and properties which are neither safety nor liveness such as the transactional properties described in section 4. Furthermore, if the behavior of the target system is known to consist only of finite executions, then every sequence is in renewal.

7 Nonuniform Enforcement

In this section, we investigate the possibility of extending the set of enforceable properties by giving the monitor some knowledge of the target program's possible behavior. This question was first raised in [14]. In [4], the authors distinguish between the uniform context, in which the monitor must consider that every sequence in Σ^∞ can occur during the target program's execution, from the nonuniform context, in which the set of possible executions is a subset of Σ^∞. They further show that in some case, the set of properties enforceable in a nonuniform context is greater than that which is enforceable in an uniform context. Later Chabot et. al. [7] showed that while this result did not apply to all runtime enforcement paradigms, it did apply to that of truncation-based monitor. Indeed, they show that in this monitoring context, a monitor operating with a subset of Σ^∞ is always more powerful than one which considers that every sequence can be output by its target.

Let $\mathcal{S}$ stand for the set of sequences which the monitor considers as possible executions of the target program. $\mathcal{S}$ is necessarily an over approximation, built from static analysis of the target. We write correctively$^{\mathcal{S}}_{\cong}$ enforceable, or just enforceable$^{\mathcal{S}}_{\cong}$, to denote the set of properties that are correctively$_{\cong}$ enforceable, when only sequences from $\mathcal{S} \subseteq \Sigma^{\infty}$ are possible executions of the target program. A property is correctively$^{\mathcal{S}}_{\cong}$ enforceable iff for every sequence in $\mathcal{S}$, the monitor can return a valid and equivalent sequence.

Definition 4. *Let $\mathcal{A}$ be an edit automaton and let $\mathcal{S} \subseteq \Sigma^{\infty}$ be a subset of executions. $\mathcal{A}$ correctively$^{\mathcal{S}}_{\cong}$ enforces the property $\hat{\mathcal{P}}$ iff $\forall \sigma \in \mathcal{S}$*

1. $\hat{\mathcal{P}}(\mathcal{A}(\sigma))$
2. $\mathcal{A}(\sigma) \cong \sigma$

Theorem 8 *A property $\hat{\mathcal{P}}$ is correctively$^{\mathcal{S}}_{\cong}$ enforceable iff*

1. *$\hat{\mathcal{P}}$ is Reasonable*
2. $\exists \hat{\mathcal{P}}' \subseteq \hat{\mathcal{P}} : \hat{\mathcal{P}}' \in$ *renewal* $: (\exists \gamma \in \mathcal{S} \rightarrow \hat{\mathcal{P}}' : (\forall \sigma \in \mathcal{S} : \gamma(\sigma) \cong \sigma) \wedge (\forall \sigma, \sigma' \in \mathcal{S} : \sigma' \preceq \sigma \Rightarrow \gamma(\sigma') \preceq \gamma(\sigma)) \wedge \gamma$ *is computable*$)$

Proof. The proof follows exactly as that of Theorem 1.

Lemma 9 *Let $\mathcal{S} \subseteq \Sigma^{\infty}$ and $\hat{\mathcal{P}}$ be a reasonable property $\hat{\mathcal{P}}$ is trivially correctively$^{\mathcal{S}}_{\cong}$ enforceable iff $\mathcal{S} \subseteq \hat{\mathcal{P}}$. If this is the case, the monitor can enforce the property by always returning the input sequence.*

It would be desirable if the set of enforceable properties increased monotonously each time a sequence was removed from $\mathcal{S}$. This means that any effort made to perform or refine a static analysis of the target program would payoff in the form of an increase in the set of enforceable properties. This is unfortunately not the case. As a counterexample, consider the equivalence relation defined as $\forall \sigma, \sigma' \in \Sigma^{\infty} : \sigma \cong \sigma'$. It is obvious that any satisfiable property can be trivially enforced in this context, simply by always outputting any valid sequence, which is necessarily equivalent to the input. No benefit can then be accrued by restricting $\mathcal{S}$.

There are, of course, some instances where constraining the set $\mathcal{S}$ does result in a increase in the set of correctively$^{\mathcal{S}}_{\cong}$ enforceable properties. This occurs when invalid sequences with no valid equivalent are removed from $\mathcal{S}$. Indeed, for any subsets, $\mathcal{S}, \mathcal{S}'$ of Σ^{∞} s.t. $\mathcal{S} \subseteq \mathcal{S}' \wedge \mathcal{S}' \backslash \mathcal{S} \neq \{\epsilon\}$, there exists an equivalence relation $\cong$ for which enforceable$^{\mathcal{S}'}_{\cong}$ enforceable$\subset^{\mathcal{S}}_{\cong}$.

Theorem 10 *Let $\mathcal{S} \subset \mathcal{S}' \subseteq \Sigma^{\infty} \wedge \mathcal{S}' \backslash \mathcal{S} \neq \{\epsilon\}$. There exists an equivalence relation $\cong$ s.t. enforceable$\subset^{\mathcal{S}'}_{\cong}$ enforceable$\subset^{\mathcal{S}}_{\cong}$.*

Proof. Let $\cong$ be defined s.t. $\exists \sigma \in \mathcal{S}' \backslash \mathcal{S} : [\sigma] \cap \mathcal{S} \neq \emptyset$. Let $\hat{\mathcal{P}}$ be the property defined as $\hat{\mathcal{P}}(\sigma) \Leftrightarrow (\sigma \notin \mathcal{S} \wedge \sigma \neq \epsilon)$. This property is not enforceable$_{\cong}^{\mathcal{S}'}$ since there exists sequences in $\mathcal{S}'$ with no valid equivalent. The property is trivially enforceable $\subset_{\cong}^{\mathcal{S}}$.

A final question of relevance on the topic of nonuniform enforcement is whether there exists some equivalence relations $\cong$ for which every reduction of the size of $\mathcal{S}$ monotonously increases the set of properties that are correctively$_{\cong}^{\mathcal{S}}$ enforceable. In other words, if there exists some $\cong$ for which $\mathcal{S} \subset \mathcal{S}' \Rightarrow$ enforceable$_{\cong}^{\mathcal{S}'} \subset$ enforceable$_{\cong}^{\mathcal{S}}$. Anyone operating under such an equivalence relation would have an added incentive to invest in static analysis of the target, as he would be guaranteed an increase in the set of enforceable properties. Unfortunately, it can be shown that this result holds only when $\cong$ is syntactic equality and at least one sequence different from ϵ is removed from the set of possible sequences.

Theorem 11 $(\sigma \cong \sigma \Leftrightarrow \sigma = \sigma') \Leftrightarrow \forall \mathcal{S}, \mathcal{S}' \subseteq \Sigma^{\infty} : \mathcal{S} \subset \mathcal{S}' \wedge \mathcal{S}' \backslash \mathcal{S} \neq \{\epsilon\}$: *enforceable*$_{\cong}^{\mathcal{S}'} \subset$ *enforceable*$_{\cong}^{\mathcal{S}}$

Proof. ($\Rightarrow$ direction)
Let $\hat{\mathcal{P}}$ be defined such that $\hat{\mathcal{P}}(\sigma) \Leftrightarrow (\sigma \notin \mathcal{S}' \backslash \mathcal{S})$. This property cannot be correctively$_{\cong}^{\mathcal{S}'}$ enforceable since any sequence in $\mathcal{S}' \backslash \mathcal{S}$ does not have a valid equivalent. The property is trivially correctively$_{\cong}^{\mathcal{S}}$ enforceable.
($\Leftarrow$ direction)
By contradiction, let $\cong$ be different than syntactic equality. This implies there exists $\sigma, \sigma' \in \mathcal{S}' : \sigma \cong \sigma' \wedge \sigma \neq \sigma'$. Further, let $\mathcal{S}' = \{\sigma, \sigma'\}$ and $S = \{\sigma\}$. We show that any property that is correctively$_{\cong}^{\mathcal{S}}$ enforceable is also correctively$_{\cong}^{\mathcal{S}'}$ enforceable. There are five cases to consider. :

- $\sigma, \sigma' \in \hat{\mathcal{P}}$: In this case, the property is always trivially enforceable.
- $\sigma \in \hat{\mathcal{P}} \wedge \sigma' \notin \hat{\mathcal{P}}$: Such a property would be both correctively$_{\cong}^{\mathcal{S}}$ enforceable and correctively$_{\cong}^{\mathcal{S}'}$ enforceable by automaton $\mathcal{A}$ for which $\mathcal{A}(\tau) = \sigma$ for all τ in the input set.
- $\sigma' \in \hat{\mathcal{P}} \wedge \sigma \notin \hat{\mathcal{P}}$: Such a property would be both correctively$_{\cong}^{\mathcal{S}}$ enforceable and correctively$_{\cong}^{\mathcal{S}'}$ enforceable by automaton $\mathcal{A}$ for which $\mathcal{A}(\tau) = \sigma'$ for all τ in the input set.
- $\sigma, \sigma' \notin \hat{\mathcal{P}} \wedge \exists \sigma'' \cong \sigma : \hat{\mathcal{P}}(\sigma'')$: Such a property would be both correctively$_{\cong}^{\mathcal{S}}$ enforceable and correctively$_{\cong}^{\mathcal{S}'}$ enforceable by automaton $\mathcal{A}$ for which $\mathcal{A}(\tau) = \sigma''$ for all τ in the input set.
- $\sigma, \sigma' \notin \hat{\mathcal{P}} \wedge \neg \exists \tau \cong \sigma : \hat{\mathcal{P}}(\tau)$: This property can neither be correctively$_{\cong}^{\mathcal{S}}$ enforceable nor can it be correctively$_{\cong}^{\mathcal{S}'}$ enforceable since there exists some sequences with no valid equivalent.

Finally, observe that since only reasonable sequences are enforceable, no possible gain can be accrued from removing only ϵ from the set of possible sequences.

8 Conclusion and Future Work

In this paper, we propose a framework to analyze the security properties enforceable by monitors capable of transforming their input. By imposing constraints on the enforcement mechanism to the effect that some behaviors existing in the input sequence must still be present in the output, we are able to model the desired behavior of real-life monitors in a more realistic and effective way. We also show that real life properties are enforceable in this paradigm, and give prefix equivalence and factor equivalence as possible examples of realistic equivalence relations which could be used in a monitoring context. The set of properties enforceable using these two equivalence relations is related to previous results in the field.

Future work will focus on other equivalence relations. Two meaningful equivalence relations which we are currently studying are subword equivalence and permutation equivalence. The first adequately models the behavior of a monitor that is allowed to insert actions into the program's execution, but may not subtract anything from it. The second models the behavior of a monitor which can reorder the actions performed by its target, but may not add or remove any of them. An even more general framework that could be envisioned would be one in which the behavior that the monitor must preserve is stated in a temporal logic.

References

1. B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, October 1985.
2. B. Alpern and F. B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2:17–126, 1987.
3. A. Bauer, M. Leucker, and C. Schallhart. Monitoring of real-time properties. In *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science, pages 260–272, 2006.
4. L. Bauer, J. Ligatti, and D. Walker. More enforceable security policies. In *proceedings of the Foundations of Computer Security Workshop*, July 2002.
5. N. Bielova, F. Massacci, and A. Micheletti. Towards practical enforcement theories. In *Proceedings of the 14th Nordic Conference on Secure IT Systems, NordSec 2009*, volume 5838 of *LNCS*, pages 239–254. Springer, October 2009.
6. H. Chabot. Sécurisation de code basée sur la combinaison d'analyse statique et dynamique, génération de moniteur partir d'un automate de Rabin. Master's thesis, Laval University, 2008.
7. H. Chabot, R. Khoury, and N. Tawbi. Generating in-line monitors for Rabin automata. In Audun Jøsang, Torleiv Maseng, and Svein J. Knapskog, editors, *Proceedings of the 14th Nordic Conference on Secure IT Systems, NordSec 2009*, volume 5838 of *LNCS*, pages 287–301. Springer, October 2009.
8. Y. Falcone, J.-C. Fernandez, and L. Mounier. Enforcement monitoring wrt. the safety-progress classification of properties.
9. P. Fong. Access control by tracking shallow execution history. In *proceedings of the 2004 IEEE Symposium on Security and Privacy.*

10. K. W. Hamlen, G. Morrisett, and F. B. Schneider. Computability classes for enforcement mechanisms. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 28(1):175–205, 2006.
11. L. Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, 3(2):125–143, 1977.
12. J. Ligatti, L. Bauer, and D. Walker. Edit automata: Enforcement mechanisms for run-time security policies. *International Journal of Information Security*, 2004.
13. J. Ligatti, L. Bauer, and D. Walker. Enforcing non-safety security policies with program monitors. In *proceedings of the 10th European Symposium on Research in Computer Security (ESORICS)*, September 2005.
14. F. B. Schneider. Enforceable security policies. *Information and System Security*, 3(1):30–50, 2000.
15. A. Syropoulos. Mathematics of multisets. In *Proceedings of the Workshop on Multiset Processing*, pages 347–358. Springer-Verlag, 2001.
16. C. Talhi, N. Tawbi, and M. Debbabi. Execution monitoring enforcement under memory-limitations constraints. *Information and Computation*, 206(1):158–184, 2008.