Equivalence-preserving corrective enforcement of security properties

Raphaël Khoury*

Department of Computer Science and Mathematics,
Université du Québec à Chicoutimi,
Quebec City, Canada
Email: raphael.khoury@uqac.ca
*Corresponding author

Nadia Tawbi

Department of Computer Science and Software Engineering,
Laval University,
Quebec City, Canada
Email: nadia.tawbi@ift.ulaval.ca

Abstract: Runtime monitoring is a widely used approach for the enforcement of security policies. It allows the safe execution of untrusted code by observing the execution and reacting if needed to prevent a violation of a user-defined security policy. Previous studies have determined that the set of security properties enforceable by monitors is greatly extended by giving the monitor some licence to transform its target execution. In this study, we present a new framework to model and study the behaviour of such monitors. In order to assure that the enforcement is meaningful, we bound the monitor's ability to transform the target execution 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 work in this field. Finally, we investigate how an a priori knowledge of the target program's behaviour would increase the monitor's enforcement power.

Keywords: dynamic analysis; monitoring; enforceable properties; security properties; corrective enforcement.

Reference to this paper should be made as follows: Khoury, R. and Tawbi, N. (2015) 'Equivalence-preserving corrective enforcement of security properties', *Int. J. Information and Computer Security*, Vol. 7, Nos. 2/3/4, pp.113–139.

Biographical notes: Raphaël Khoury obtained his PhD degree from Laval University and was a Post-Doctoral Fellow at the Defense Research and Development Canada (DRDC-RDDC) in Valcartier, Canada. He is currently a Professor at the Université du Québec à Chicoutimi. He is the recipient of a FQRNT and NSERC scholarships.

Nadia Tawbi obtained her PhD degree from the Université Pierre et Marie Curie in 1991. She worked at the Bull Research Center, in France from 1998 to 1996. Since 1996, she is a Professor of Computer Science at Laval University in Québec City, Canada. She has several publications on parallelising compilers, formal verification and language-based security.

This paper is a revised and expanded version of a paper entitled 'Using equivalence relations for corrective enforcement of security policies' presented at the 5th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security, MMM-ACNS 2010, St. Petersburg, Russia, 8–10 September 2010.

1 Introduction

The proliferation of mobile code and distributed software went hand in hand with a proliferation of security risks. This has led to a growing awareness on the part of the user to the need to protect their systems when running untrusted code or code of unknown origin. A widely used solution is runtime monitoring, in which a runtime monitor observes the execution of the target program and reacts to prevent a possible violation of a security policy. 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 specific execution. Furthermore, prior research has established that monitors are capable of enforcing a wide range of security policies.

The monitor is modelled as an automaton, termed an edit automaton (Bauer et al., 2002), which takes the program's execution as input, and outputs an alternate execution, usually by truncating the input if it is invalid. In this paper, we qualify an execution sequence as valid if it satisfies the security policy under observation and invalid otherwise. 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. One of the main results of these enquiries is that the set of security policies that can be enforced by monitors is greatly enhanced if the monitor is given some latitude to alter the input sequence. Indeed, some properties, such as general availability, can only be enforced if this is the case. However, this ability must be constrained, otherwise, the monitor can enforce any property, but not in a useful or desirable manner.

The most commonly used policy enforcement framework, effective$_{\cong}$ enforcement (Schneider, 2000; Bauer et al., 2002), is defined in terms of an equivalence relation $\cong$. It only requires that the monitor's output be equivalent to the input execution if the latter is valid. Otherwise, the monitor can output any valid sequence.

The only equivalence relation which has been studied in this framework is syntactic equality. This result from the lack of any constraints limiting exactly which equivalence relations can be used in practice. This point is concisely explained by Ligatti et al. (2005):

> "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 an arbitrarily chosen sequence. This strictly meets the definition of enforcement but does not provide a meaningful enforcement of the desired policy.

This problem is compounded by the fact that the definition of effective$_{\cong}$ enforcement does not place any restriction on the output of the monitor when the observed sequence does not respect the property, as long as this output is itself valid. This means that once the monitor determines that a sequence is irremediably invalid, it can ignore the target program's behaviour and any valid sequence, even if this output is completely unrelated to the observed execution. Once again, such a behaviour would fit the definition of enforcement, but would not provide a useful enforcement of the desired policy, where one would prefer that an invalid sequence be corrected in a more systematic or predictable manner.

This point was also raised in the literature, Bielova et al. (2009) write:

> "What distinguishes an enforcement mechanism is not what happens when traces are good, because nothing should happen! The interesting part is how precisely bad traces are converted into good ones".

For example, consider a system managing online purchases, and a security policy forbidding a user from browsing certain merchandise without prepaying. A monitor could abort the execution as soon as this is attempted. But the property would also be enforced by replacing the input sequence with any sequence of actions respecting the policy, even if it contains purchases unrequested by any users, or by outputting nothing, depriving legitimate users of the ability to use the system.

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 propose to use equivalence relations to model the essential behaviours of the execution being monitored, which must be preserved throughout any transformation performed on it by the monitor. This has the effect of binding the monitor's behaviour in its treatment of both valid and invalid sequence, since in both cases, the output must remain equivalent to the input. This allows us to address both of the two issues raised above.

Intuitively, this enforcement parading models a behaviour that is closer to that which would be encountered in practice, in which the actions taken by the monitor are constrained by a limitation that certain element present in the original sequence be preserved. In this paper, we 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 Ligatti et al. (2005) for effective enforcement can be considered as special cases of our more general framework.

The remainder of this paper is organised 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 which conditions equivalence relations can be used to transform sequences and ensure that they respect 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 behaviour increases the monitor's enforcement power. In Section 8, we discuss some limitations of our framework. Concluding remarks and avenues for future work are laid out in Section 9.

2 Related work

2.1 Identifying the set of monitorable properties

A central question of the study of monitors has been to identify the set of properties enforceable by monitors operating under various constraints. As we will show in this section, this question is actually related to a more fundamental question: what behaviour is expected of a monitor which is said to enforce a property.

Schneider, in his seminal work (Schneider, 2000), addresses these questions. He argues that to enforce a property, a monitor should respect the following two principles:

1 *soundness:* all output must respect the desired property

2 *transparency:* the semantics of executions which already respect the property must be preserved.

The various enforcement paradigms that have been suggested in the literature only differ on how the transparency requirement is implemented. Schneider (2000) uses the most restrictive notion of transparency possible: every action of a valid sequence must be output by the monitor in lockstep with the output of the program. This is termed *precise enforcement*.

He further focuses on a specific class of monitors, which observe the execution of a target program with no knowledge of its possible future behaviour and with no ability to affect it, except by aborting the execution. Under these conditions, he found that a monitor could precisely enforce the security policies that are identified in the literature as *safety* properties, and are informally characterised by prohibiting a certain bad thing from occurring in a given execution.

Following this result, several implementations of formal monitors are limited to the enforcement of safety properties. Yet, Schneider's study also suggested that the set of properties enforceable by monitors could be extended under certain conditions. Conversely, he also points out that other constraints, such as computability constraints, which are unavoidable, restrict further the behaviour of monitors.

Building on this insight, Bauer et al. (2002) and Ligatti et al. (2004) examined the way the set of policies enforceable by monitors could be extended if the monitor had some knowledge of its target's possible behaviour or if its ability to alter that behaviour 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 behaviour

3 according to how much latitude the monitor is given to transform its targets execution.

Consequently, they provide a rich taxonomy of classes of security policies, associated with the appropriate model needed to enforce them.

They find that the notion of precise enforcement is so rigid that if it used, the added power of some monitors to transform the executions of their targets (by inserting or suppressing program actions) does not result in an increase in the set of enforceable properties.

As an alternative, they suggest the notion of effective$_{\cong}$ enforcement. A monitor effectively$_{\cong}$ enforces a property if any execution respecting the property is replaced by an equivalent execution, w.r.t. some equivalence relation $\cong$. No restriction is imposed on the output of the monitor if its input is invalid, other than that the output sequence must be valid. Operating under this enforcement paradigm, a monitor capable of more powerful responses when faced with a violation of the security property is able to enforce a wider range of security properties. They also conclude that the range of properties enforceable in most monitoring frameworks can be extended by relying upon static analysis to provide the monitor with a model of its target's possible behaviour.

Subsequently, Ligatti et al. (2005) delineate the set of properties that are effectively$_{\cong}$ enforceable for a specific equivalence relation, syntactic equality. Intuitively, this monitor operates by suppressing potentially malicious sequences until it determines the execution to be valid, at which point the suppressed sequence can safely be output. In Falcone et al. (2008a), it is observed that a monitor behaving in this manner always outputs the longest valid prefix of its input, if the latter is invalid.

Ligatti et al. define the set of infinite renewal properties (*renewal*) in order to characterise the set of properties enforceable in this manner. 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. Every property over finite sequence is in renewal. Falcone et al. (2008a, 2008b, 2009a, 2008b) show that the class of renewal properties proposed by Ligatti et al. is equivalent to the union of five of the six classes of the safety-progress classification of properties (Chang et al., 1991), namely safety, guarantee, obligation, response and persistence, with the class of reactivity properties containing properties which are not effectively$_{=}$ enforceable in this manner.

Bielova et al. (2009) define several sub-classes to the effective$_{=}$ enforcement paradigm, based on the possible reactions of the monitor to potential violations of the security policy. Indeed, while the definition of effective$_{\cong}$ enforcement does not place any restriction on the behaviour of the monitor when its input is invalid, Bielova observes that such restrictions are actually central to the implementations of real monitors.

Fong (2004) first considered the question of monitors operating with memory constraints. He studied monitors which only record the shallow history (i.e., the unordered set of security relevant events performed by the target program) and found that despite this limitation, monitors could still enforce a number of interesting real-life properties. Subsequently, Talhi et al. (2008) considered monitors with a bounded memory, and found a close connection between the set of properties they can enforce and a class of languages termed locally testable languages (or local properties) (Beauquier and Pin, 1991). In Kupferman et al. (2006), it is observed that such properties are particularly well suited to be enforced by monitoring because a monitor enforcing such a property needs only to keep a record of a bounded number of computational cycles.

Furthermore, if a monitor enforces several such properties, this record can be shared, so that the memory overhead of the monitor is independent of the number of locally testable properties being enforced.

A final enquiry in the set properties enforceable by monitors with memory constraints was done by Beauquier et al. (2009), who examine the set of properties enforceable by an edit automaton with a finite, but unbounded, set of states.

Both Schneider and Ligatti et al. have stated that the computability constraints may cause a property to be unenforceable by a given security mechanism even if it lies inside the set of properties enforceable by this mechanism. Kim et al. (2002) give a more precise characterisation of this restriction. They observe that for a monitor to be able to enforce a property, it needs to be able to identify any invalid sequence upon the inspection of a finite prefix of this sequence. It follows that only properties for which this is possible are enforceable by monitors. As observed in Viswanathan (2000), this result can also be given as stating that the set of enforceable properties is the class of co-recursively enumerable properties (coRE).

Hamlen et al. (2006) tighten this bound further by showing that even properties that meet this criteria may not be monitorable, if the property is stated in such a way that the violation occurs before the monitor can detect it, or if the monitor is unable to correct the violation once it has detected it. They show that a better characterisation to the set of properties enforceable by monitors is the intersection between the class coRE and the class of properties enforceable by program rewriters. This result agrees with an intuition that monitors can be inlined in the program they aim to protect, so that any property enforceable by monitor can also be enforced by program rewriting.

Bauer et al. (2002, 2010) study the set of properties that are monitorable if the monitor is tasked only with detecting and reporting the occurrence of a violation of the security property. The properties enforceable in this context includes safety properties, co-safety properties (Kupferman et al., 2001), and some other properties which are neither safety nor co-safety. The authors further show how to construct a monitor that is both minimal (w.r.t. its size) and optimal, in that it detects a good or bad prefix as early as possible. This result, and its implications, are expounded in more detail in Bauer (2010). Basin et al. (2013) tighten previous results by considering a situation where monitor might not have the capability to interrupt every program action before it takes place.

In this paper, we suggest an alternative definition of enforcement which incorporates more fine-grained restrictions on the treatment of invalid sequences. Previous work tackling the problem addressed in this paper includes Bielova et al. (2009), Bielova and Massacci (2011) and Khoury and Tawbi (2012a).

Bielova et al. (2009) and Bielova and Massacci (2011) suggest that syntactic, rather than semantic constraints be imposed on monitors possible transformations. We believe that a semantic-based grouping of similar executions would give the user greater flexibility, as in cases where divergent executions can be thought of as similar, or conversely, where two syntactically close executions are semantically different.

In a previous paper (Khoury and Tawbi, 2012a), we proposed a solution based on organising the possible sequences along a preorder while forbidding the monitor from replacing an invalid input with a sequence that lies lower on this preorder. The new solution developed in this paper facilitates the process of stating the constraints to great advantage, and would therefore constitute suitable basis to address the emerging goal of monitor certification (Sridhar and Hamlen, 2011).

A comprehensive survey of studies related to the enforcement power of monitors is presented in Khoury and Tawbi (2012b).

2.2 Other study in monitoring

Alongside the question of determining the set of properties enforceable by monitors, other studies of this enforcement mechanisms have contributed our understanding of how to construct such formal systems.

The question of how to construct a monitor from an automaton representing a security property, has been addressed several times in the literature. A constructive proof that this is possible for all properties over finite sequences is given in Bauer et al. (2002). An alternative algorithm is given in Bielova et al. (2009). Falcone et al. (2008a, 2008b, 2009a, 2008b) give algorithms to construct a monitor from a property for each of the five classes of the safety-progress classification of properties for which such a construction is possible.

In d'Amorim and Roşu (2005), an algorithm is given to extract a monitor which detects the minimal bad prefix of a property stated as a Büchi automaton. This method does not monitor the liveness component of a property.

While the model of monitors presented throughout this section is the most widely used in the literature, other models have been proposed. Zhu et al. (2006) suggest modelling monitors by way of a *stream automata* while another alternative model, the *mandatory results automata* is suggested by Ligatti and Reddy (2010). Both of these models differ from the edit automaton model in that they distinguish between the action set of the target and that of the of system it interacts with. These models made it easier to study the interaction between the target program, the monitor and the system. In this study, while we will continue to focus on the edit automaton, we believe the results we present can easily be applied to the automata models introduced in these papers. Jun et al. (2008) models monitors as Mealy (1955) machines and study their expressive power.

Other monitoring frameworks are more specific with regard to the systems or the properties they can be used to enforce. The monitoring of security protocols is discussed in Bauer and Jürjens (2008). That of information flow policies is discussed in numerous papers including Chudnov and Naumann (2010) and Le Guernic (2007). Sen at al. (2004) propose a decentralised monitor which monitors safety properties in distributed programs. The optimisation of monitors is further discussed in Yan and Fong (2009). The in-lining of monitors in concurrent programs is discussed in Langa et al. (2006). An algebraic method to in-line a safety property into a program is given in Langar and Mejri (2005) and Langar et al. (2007). In this approach, both the property and the program are stated using process algebra. The instrumented program is shown to be equivalent to the original one using a notion of equivalence based on bisimulation. The monitoring of networks is discussed in Mechri et al. (2007).

3 Preliminaries

Let us briefly start with some preliminary definitions.

Executions are modelled 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 $\tau;\sigma' = \sigma$. = Σ. We write $\tau \prec \sigma$ (resp. $\sigma \succ \tau$) for $\tau \preceq \sigma \wedge \tau \neq \sigma$ (resp. $\sigma \succ \tau \wedge \tau \neq \sigma$). 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. Overloading 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 written 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 inclusively, 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 (Syropoulos, 2001) is a generalisation 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 $\mathcal{A} = \langle A, f \rangle$ yields a new multiset $\mathcal{B} = \langle A, g \rangle$ where $g(a) = max(f(a) - 1, 0)$.

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. $\sigma \in \mathcal{P} \Leftrightarrow \hat{\mathcal{P}}(\sigma)$. 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 (Lamport, 1977), 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

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

Alternatively, a *liveness* property (Alpern and Schneider, 1985) 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

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

Any property can be stated as the conjunction of a safety property and a liveness property (Alpern and Schneider, 1987). Another relevant set of properties is that of infinite

renewal properties (*renewal*), defined in Ligatti et al. (2005) to characterise 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

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

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

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

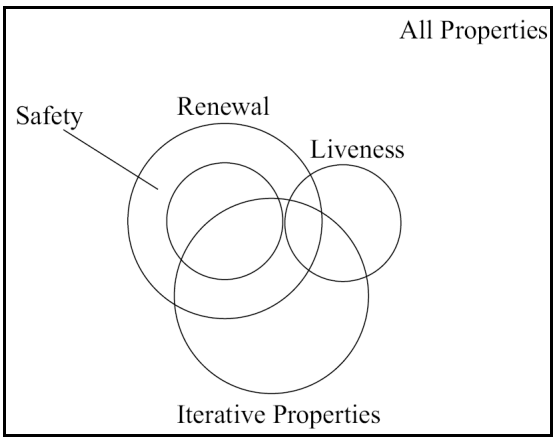
Note that the definition of renewal imposes no restrictions on the finite sequences in $\hat{\mathcal{P}}$. For infinite sequences, the set of renewal properties includes all safety properties, some liveness properties and some properties which are neither safety nor liveness.

Finally, we formalise the set of *transactional* properties, suggested in Ligatti et al. (2005), 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 behaviour of systems which repeatedly interacts with clients using a well defined protocol, such as a system managing the allocation of resource or the accesses to a database. Let Σ be an action set and $\mathcal{T} \subseteq \Sigma^*$ be a subset of finite transactions, $\hat{\mathcal{P}}_{\mathcal{T}}$ is a transactional property over set $\mathcal{T}$ iff

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

This definition is subtly different, and indeed forms a subset, to that of iterative properties defined in Bielova et al. (2009). 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. Figure 1 shows how the various classes of properties mentioned in this section relate to one another.

Figure 1 Iterative properties



4 Monitoring with equivalence relations

As discussed earlier, this paper seeks to solve two issues caused by limitations of existing security policy enforcement paradigms. First, it is difficult to state meaningful equivalence relations which can be used in practice. Second, these enforcement paradigms place no restrictions on the behaviour of the monitor when the input is invalid, other than that the monitor must output a valid 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 behaviour 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 behaviours should be corrected or removed. It is precisely these corrective behaviours that we seek to model using our equivalence relations. The enforcement paradigm thus ensures that the output is always valid, and that all valid behaviour intended by the user in the input, is present in the monitor's output.

The idea of using equivalence relations to transform execution sequences was first suggested in Hamlen et al. (2006), in context of program rewriters. The only restriction placed on the possible equivalence relations is that they must be *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:

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

A similar restriction, *indistinguishability* is suggested in Ligatti et al. (2005).

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 (2004), 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. The set $\mathcal{I}$ can be any abstraction of the program's behaviour. Fong focuses on the shallow (unordered) history of the execution and makes use of this abstraction to reduce the overhead of the monitor. In the following sections, we will suggest other abstractions and show how they can be used 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:

$$\forall \sigma, \sigma' \in \Sigma^* : \mathcal{F}(\sigma) = \mathcal{F}(\sigma') \Rightarrow \hat{\mathcal{P}}(\sigma) \Leftrightarrow \hat{\mathcal{P}}(\sigma') \qquad (1)$$

Since we wish to use the abstraction to restrict the possible behaviour of the monitor, a natural framework would be to restrict the monitor to transformations for which any transformation preserves the same value of abstraction. While this approach could work for in a limited number of cases, it is in general too constraining. Instead, we propose that the equivalence relation group together a number of abstraction values in each equivalence class. In order to avoid the problem discussed above, whereas any property can be enforceable over meaningless equivalence relations, we impose restrictions on the how the equivalence relations can be stated.

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. $\tau' \sqsubseteq \tau$. 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.

$$\forall \sigma, \sigma', \sigma'' \in \Sigma^\infty : \sigma \sqsubseteq \sigma' \sqsubseteq \sigma'' \wedge \sigma \cong \sigma'' \Rightarrow \sigma \cong \sigma' \tag{2}$$

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:

$$\forall \sigma, \sigma' \in \Sigma^* : \sigma \cong \sigma' \Rightarrow \exists \tau \in \Sigma^* : \tau = (\sigma \sqcap \sigma') \wedge \tau \cong \sigma \tag{3}$$

where

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

The intuition behind the above two restrictions, is that, if an equivalence restriction meets these two criteria, a monitor looking for a 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.

Some examples can help illustrate this idea.

- *Access control:* A simple example is that of access control policies, safety properties that limit which resources each process may access. Enforcement of this property could employ a number of strategies including, aborting a process, suppressing illicit access but allowing the remainder of the execution to continue, reordering request to make a sequence compliant with a policy, or adding certain actions (such as log entries or access requests) to the sequence. In any case, a meaningful enforcement of this property should ensure that all valid access requests present in the input sequence are preserved, and that the monitor does not add any additional unrequested resource accesses. A natural abstraction function to enforce this property would be a function $\mathcal{F}_{ac}$: $\Sigma^* \rightarrow \wp(\Sigma)$ returning the set of valid resource accesses present in a

given sequence. Two sequences are similar, w.r.t. this abstraction function, if they share the same valid resource accesses, irrespective of any invalid access requests present in either sequence. Imposing that the monitor must replace any invalid sequence with a valid similar one gives the monitor just enough latitude to output a valid sequence, without allowing it to transform unnecessarily.

- *Mutual exclusion*: As a second example, consider mutual exclusion, a policy forbidding two processes from simultaneously executing a critical section. Each action of the input is executed by either one of the two processes running simultaneously. In this case, the abstraction function is $\mathcal{F}_{me}$: $\Sigma^* \rightarrow (\Sigma^\infty \times \Sigma^\infty)$ whose image is a pair of sequences, each containing the subsequences of actions performed by one of the processes. A monitor enforcing this property by reordering the input sequence to ensure sequential access to the critical section returns an output sequence with the same abstraction as the original input sequence.
- *Fair resource allocation:* If the monitor is operating in a context in which the users are not believed to be malicious, (as is the case, for instance, of a monitor allocating resources between trusted users), then the most important restriction on its behaviour would be that it never worsen an accepted situation. Consider the case of a monitor optimising the allocation of CPU time between several processes. A suitable abstraction function would be $\mathcal{F}_{ra}$: $\Sigma^* \rightarrow \mathbb{N}$, which indicates the longest waiting period of any process before it executes some action. Binding the monitor's transformation to this abstraction function captures a restriction on the monitor's behaviour to the effect that any transformation performed on the input sequences cannot increase the idle time of any of the monitored processes.

As these examples illustrate, we believe it is much easier to bind the behaviour of the monitor using abstraction functions than equivalence relations, as previous work on this topic suggested. In fact, we have found that the needed abstraction function usually flows immediately from the chosen property. These examples also illustrate the flexibility of the approach and the diversity of possible abstractions functions that can be used. These examples also motivate the need for corrective enforcement, as in each case, a monitor could effectively$_=$ enforce the desired property by replacing an invalid execution by an unrelated valid sequence, but the resulting enforcement would not be useful. For instance, a monitor that effectively enforce the access control policy and observes an illegal resource access attempt must suppress it, but it could additionally choose to suppress any other valid resource accesses present in the execution and can even add unrequested legal resource accesses.

We define the abstraction function over finite sequences only, since any modification performed by the monitor on the target sequence always occurs in finite time. Since the notion of equivalence must be extended to infinite sequences, we impose that two infinite sequences are equivalent iff they have infinitely many pair-wise equivalent prefixes.

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

$$\forall \sigma, \sigma' \in \Sigma^\omega : \sigma \cong \sigma' \Leftrightarrow \forall \tau \prec \sigma : \exists \upsilon \succeq \tau : \exists \tau' \prec \sigma' : \upsilon \cong \tau' \tag{4}$$

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:

$$\forall \tau, \tau' \in \Sigma^* : \tau \cong \tau' \Rightarrow \tau;\sigma \cong \tau';\sigma \tag{5}$$

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, $\sqsubseteq$ organises 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. We believe the approach presented in this paper could easily be adapted to the more elaborate model of monitoring presented in Ligatti and Reddy (2010).

The edit automaton (Bauer et al., 2002; Ligatti et al., 2005) is the most general model of a monitor. It captures the behaviour of a monitor capable of inserting or suppressing any action, as well as halting the execution in progress.

Definition 5.1: 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
- δ: $(Q \times \Sigma) \rightarrow (Q \times \Sigma^\infty)$ 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. As discussed above, a mechanism effectively enforces a security property iff it always outputs a valid sequence, and the output is equivalent to the input when the latter is valid.

Definition 5.2 (from Bauer et al., 2002): 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 (Bauer et al., 2002). Effective enforcement is only one paradigm of enforcement which has been suggested. Other enforcement paradigms include precise enforcement (Bauer et al., 2002), all-or-nothing delayed enforcement (Bielova et al., 2009) or conservative enforcement (Bauer et al., 2002). Yet, these enforcement paradigms are also limited to syntactic equality.

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 behaviour 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 behaviours should be corrected or removed. It is precisely these corrective behaviours that we seek to model using our equivalence relations. The enforcement paradigm thus ensures that the output is always valid, and that all valid behaviour intended by the user in the input, is present in the monitor's output.

Definition 5.3: 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. We write enforceable$_{\cong}$ for the set of properties which are correctively$_{\cong}$ enforceable.

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

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

2 $\hat{\mathcal{P}}$ is reasonable

3 there exists a computable function $\gamma : \Sigma^{\infty} \to \hat{\mathcal{P}}' : \forall \sigma \in \Sigma^{\infty} : \gamma(\sigma) \cong \sigma$

4 $\forall \sigma' \preceq \sigma : \gamma(\sigma') \preceq \gamma(\sigma)$.

Proof 1: ($\Rightarrow$ direction) By construction of the following automaton. $\mathcal{A}$ = ⟨Σ, Q, q_0, δ⟩ where

- $Q = \Sigma^*$, the sequence of actions seen so far
- $q0 = \epsilon$
- the transition function δ is given as $\delta(\sigma, a) = (\sigma; a, \sigma')$, where $\exists a', \tau : \sigma = a'; \tau$ 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.

Let $\Sigma \in \Sigma^{\infty}$ be the input sequence, let $\Sigma[..i]$ be the prefix of the input processed at step i and let $q_i = (\sigma[..i], \sigma_i')$ be the state of $\mathcal{A}$ reached after having processed $\Sigma[..i]$. At each step i of the execution, the automaton maintains the following invariants $INV(i) = \sigma_i' \in \hat{\mathcal{P}} \wedge \sigma_i' \sim \sigma[..i]$.

The invariant holds initially, as by condition 2 of Theorem 1, ϵ is valid and similar to itself. Let us assume INV(i) holds and let a be the next input action. By the definition of the automaton's transition function, $\sigma_i' + 1 = \gamma(\sigma_i'; a)$ and by the definition of γ, $\gamma(\sigma_i'; a)$ is both valid and similar to $\Sigma[..i + 1]$.

For finite sequences the invariant is sufficient for the proof. For infinite sequence, there are two cases to consider. The monitor will either output infinitely many valid prefixes, or else at some point of the execution it will discard the input and output an infinite sequence that is similar to every possible continuation of the input. In the former case, condition 6 imposes that the output is in Renewal. By the definition of renewal, a sequence with infinitely many valid prefixes is necessarily valid, while equation (5) ensures that the output is similar to the input. In the latter cases, the definition of γ ensures the output is both valid and similar to the input.

($\Leftarrow$ 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 $\mathcal{A}(\Sigma) \sim \Sigma$.

That the image of γ, ($\hat{\mathcal{P}}'$) is a subset of $\hat{\mathcal{P}}$ follows trivially from the assumptions $\forall \sigma \in \Sigma^{\infty} : \hat{\mathcal{P}}(\mathcal{A}(\sigma))$. 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. Furthermore, since γ is applied recursively to every prefix of the input, it is thus unavoidable that $\forall \sigma' \preceq \sigma : \gamma(\sigma') \preceq \gamma(\sigma)$. If condition 5 did not hold, it would mean that for some input Σ, we have that $\gamma(\Sigma)$ is an infinite sequence and $\Sigma; \tau \nsim \gamma(\Sigma)$. Thus, whenever $\Sigma; \tau$ is the input sequence, the monitor will return an output that is not similar to the input, and the property is not correctively$_{\sim_{\mathcal{F}}}$ enforced. Finally, if condition 6 did not hold, it would mean that the monitor could output an invalid infinite sequence with infinitely many valid prefixes. Since such an output would be invalid, the property would not be enforced.

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 $\cong_1 < \cong_2 \Rightarrow \text{enforceable}_{\cong_1} \subset \text{enforceable}_{\cong_2}$.

Proof 2: It is easy to see that any property which is correctively$_{\cong_1}$ enforceable is also correctively$_{\cong_2}$ enforceable, since every pair of sequences 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 $[\sigma]_{\cong_2} \setminus [\sigma]_{\cong_1}$ 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 the next section to set an upper bound to the set of enforceable properties with specific equivalence relations.

Lemma 1: 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 its 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 we will consider is factor equivalence, which models the class of transactional properties introduced in Section 3. 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 behaviour 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. Transactional properties form a subset to the class of renewal properties which can be effectively$_{=}$ enforced (Ligatti et al., 2005), by outputting the longest valid prefix. Bielova et al. (2009) propose an alternate enforcement paradigm, which allows all valid transactions to be output. Corrective$_{\sqsubseteq}$ enforcement can be seen as a generalisation of their work.

Let the abstraction function $\mathcal{F}$ be $valid_{\mathcal{T}}(\sigma)$, which stand for the multiset of factors of from the sequence σ which are present in $\mathcal{T}$. The partial order $\sqsubseteq$ used to correctively enforce this property is thus given as $\forall \sigma, \sigma' \in \Sigma^{\infty} : \sigma \sqsubseteq \sigma' \Leftrightarrow \text{valid}_{\mathcal{T}}(\sigma) \subseteq \text{valid}_{\mathcal{T}}(\sigma')$. This partial order captures the intuition that any valid transaction present in the original sequence must also be present in the monitor's output.

For example, let $\mathcal{T}$ = {`open`, `close`, `log`} be a set of atomic actions and let T = {`open`; `log`; `close`} be the set containing the only allowed transaction. If the input sequence is given as Σ = `log`; `open`; `log`; `close`; `log`; `open`; `close`; `open`; `log`; `close`, then valid$_{\mathcal{T}}$(Σ) is the multiset containing two instances of the factor `open`; `log`; `close`.

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, possibly with a different ordering. Two sequences τ and τ' are factor equivalent, written $\tau \cong_{\mathcal{T}} \tau'$ iff they share the exact same value of $\mathcal{F}$, and thus, the exact same multiset of valid transactions. Note that this equivalence relation does not meet the consistency criterion suggested by Hamlen et al. (2006) since a valid sequence τ could be equivalent to an invalid sequence τ' if they both contain the same valid transactions, but τ' additionally contains at least one invalid transaction. This equivalence relation does, however, meet the restrictions 2, 3, 4 and 5 stated in Section 4.

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 Bielova et al. (2009) 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.

Let $\mathcal{T} \subseteq \Sigma^*$ be a set of factors and let $\hat{\mathcal{P}}_{\mathcal{T}}$ be 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 $\mathcal{T}$:

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

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 on which to base such a decision. Conversely, if the set $\mathcal{T}$ is ambiguous, there may be sequences which cannot be stated as a concatenation of valid transactions, and thus are not in the property, but for which every action contained in the sequence is part of a valid transaction, and thus cannot be removed without also removing a valid transaction. It follows that the property cannot be enforced in such a case.

Theorem 3: A transactional property $\hat{\mathcal{P}}_{\mathcal{T}}$ correctively$_{\cong_{\mathcal{T}}}$ enforceable if it is transactional, reasonable, and $\mathcal{T}$ is unambiguous.

Proof 3: From Theorem 1, the property $\hat{\mathcal{P}}$ is correctively$_{\cong}$ enforceable iff there exists a function $\gamma : \Sigma^{\infty} \to \hat{\mathcal{P}}'$, 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 $\gamma : \Sigma^{\infty} \to \hat{\mathcal{P}}_{\mathcal{T}}$. 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 $\forall\sigma,\sigma' \in \Sigma^* : \sigma' \preceq \sigma : \gamma(\sigma') \preceq \gamma(\sigma)$. Transactional properties are reasonable by definition.

We have only to refer to Lemma 1 in order to state a precise upper bound to the set of enforceable properties.

Theorem 4: 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 4: ($\Rightarrow$ direction) Follows directly from Theorem 3 and Lemma 1. ($\Leftarrow$ 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 Σ_1, Σ_2, $\Sigma_3 \in \mathcal{T}$ and invalid sequences $\tau, \tau' \notin \mathcal{T}$ s.t. τ ; Σ_3; $\tau' = \sigma_1;\sigma_2$ and Σ_1, Σ_2, Σ_3 are valid transactions. Let υ is an invalid transaction (possibly τ or τ') and let the infinite sequence Σ_1; υ; Σ_2; υ; Σ_1; υ... be the input sequence. There cannot be a valid equivalent sequence since any concatenation Σ_1; Σ_2 also contains the transaction Σ_3. Such a property is thus unenforceable. There cannot be a valid equivalent sequence since any concatenation Σ_1; Σ_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 (2005) result, 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[2], can be stated as a special case of our framework. This proof also serves as a proof that if the enforcement method described in Ligatti et al. (2005) is used, the monitor always outputs the longest valid prefix of a invalid sequence. This was stated but not proven in Falcone et al. (2008a).

First, we need to align our definitions of enforcement. Using effective enforcement, Ligatti et al. 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. However, in Falcone et al. (2008a), it is stated that the semantics of their monitor do impose that the

output remain a prefix of the input in all cases and that the longest valid prefix always be output. This characterisation can be translated in our formalism by instantiating $\cong$ to $\cong_{\preceq} \stackrel{def}{=\!=} \forall \sigma, \sigma' \in \Sigma^* : \sigma \cong_{\preceq} \sigma' \Leftrightarrow pref(\sigma) \cap \hat{\mathcal{P}} = pref(\sigma') \cap \hat{\mathcal{P}}$. 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. This equivalence relation can be obtained by using the identity function as the abstraction function $\mathcal{F}$ and $\preceq$ as the basis of the partial order between sequences[3]. Once again, this equivalence relation satisfies the restrictions imposed in Section 4.

Theorem 5: A property $\hat{\mathcal{P}}$ is effectively= enforceable iff it is correctively$_{\cong_{\preceq}}$ enforceable.

Proof 5: ($\iint$ direction) From Ligatti et al. (2005), we have that a property is effectively$_{=}$ enforceable iff

1 $\hat{\mathcal{P}}(\sigma) \Rightarrow \mathcal{A}(\sigma) = \sigma$

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 Ligatti et al. (2005) 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 $\sigma \cong_{\preceq} \mathcal{A}(\sigma)$.

Theorem 6: A property $\hat{\mathcal{P}}$ is correctively$_{\cong_{\preceq}}$ enforceable iff it is in renewal, reasonable and computable.

Proof 6: Immediate from Theorem 5 and Theorem 3 of Ligatti et al. (2005). As discussed in Ligatti et al. (2005), 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 behaviour of the target system is known to consist only of finite executions, then every sequence is in renewal.

The proof of Theorem 5 implies that a monitor which effectively$_{=}$ enforces a property $\hat{\mathcal{P}}$ by suppressing a potentially invalid sequence, and outputting it when it determines this sequence to be valid will always output the longest valid prefix of an invalid sequence. This result is stated but not proved in Falcone et al. (2008a).

Lemma 2: A monitor that effectively$_{=}$ enforces a property $\hat{\mathcal{P}}$ in the manner described in Ligatti et al. (2005) always outputs the longest valid prefix of an invalid input sequence.

Proof 7: Immediate from the proof of Theorem 5.

7 Non-uniform 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 behaviour. This question was first raised in Schneider (2000). Bauer et al. (2002) 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 non-uniform context, in which the set of possible executions is a subset of Σ^{∞}. They further show that in some cases, the set of properties enforceable in a non-uniform context is greater than that which is enforceable in an uniform context. Later Chabot et al. (2009) 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 7.1: 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 7: 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 :

$$(\exists \gamma \in \mathcal{S} \to \hat{\mathcal{P}}' : \ (\forall \sigma \in \mathcal{S} : \gamma(\sigma) \cong \sigma)$$
$$\wedge (\forall \sigma, \sigma' \in \mathcal{S} : \sigma' \preceq \sigma \Rightarrow \gamma(\sigma') \preceq \gamma(\sigma)) \wedge \gamma \text{ is computable}).$$

Proof 8: The proof follows exactly as that of Theorem 1.

Lemma 3: 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 monotonically 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. $\mathcal{S} \subseteq \mathcal{S}' \wedge \mathcal{S}' \setminus \mathcal{S} \neq \{\epsilon\}$, there exists an equivalence relation $\cong$ for which enforceable$^{\mathcal{S}'}_{\cong} \subset$ enforceable$^{\mathcal{S}}_{\cong}$.

Theorem 8: Let $\mathcal{S} \subset \mathcal{S}' \subseteq \Sigma^{\infty} \wedge \mathcal{S}' \setminus \mathcal{S} \neq \{\epsilon\}$. There exists an equivalence relation $\cong$ s.t. enforceable$^{\mathcal{S}'}_{\cong} \subset$ enforceable$^{\mathcal{S}}_{\cong}$.

Proof 9: Let $\cong$ be defined s.t. $\exists \sigma \in \mathcal{S}' \setminus \mathcal{S} : [\sigma] \cap \mathcal{S} \neq$ Û. 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$^{\mathcal{S}'}_{\cong}$ since there exists sequences in $\mathcal{S}'$ with no valid equivalent. The property is trivially enforceable $\subset^{\mathcal{S}}_{\cong}$.

A final question of relevance on the topic of non-uniform enforcement is whether there exists some equivalence relations $\cong$ for which every reduction of the size of $\mathcal{S}$ monotonically increases the set of properties that are correctively$^{\mathcal{S}}_{\cong}$ enforceable. In other words, if there exists some $\cong$ for which $\mathcal{S} \subset \mathcal{S}' \Rightarrow$ enforceable$^{\mathcal{S}'}_{\cong} \subset$ enforceable$^{\mathcal{S}}_{\cong}$. 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 9: $(\sigma \cong \sigma \Leftrightarrow \sigma = \sigma') \Leftrightarrow \forall \mathcal{S}, \mathcal{S}' \subseteq \Sigma^{\infty}$: $\mathcal{S} \subseteq \mathcal{S}' \wedge \mathcal{S}' \setminus \mathcal{S} \neq \{\epsilon\}$: enforceable$^{\mathcal{S}'}_{\cong}$ $\subset$ enforceable$^{\mathcal{S}}_{\cong}$.

Proof 10:

($\Rightarrow$ direction)

Let $\hat{\mathcal{P}}$ be defined such that $\hat{\mathcal{P}}(\sigma) \Leftrightarrow (\sigma \notin \mathcal{S}' \setminus \mathcal{S})$. This property cannot be correctively$_{\cong}^{\mathcal{S}'}$ enforceable since any sequence in $\mathcal{S}' \setminus \mathcal{S}$ 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 $\mathcal{S}'\{\sigma \cong \sigma'\}$ and $\mathcal{S} = \{\sigma\}$. 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
- $\sigma, \sigma' \notin \hat{\mathcal{P}} \wedge \exists \tau \cong \sigma : \hat{\mathcal{P}}(\tau)$: 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 Limitations

Like all monitors, the ones described in this paper are limited by memory and computational constraints, which we have not taken into account in our analysis. The results given in Theorems 4 and 6 should thus be seen as upper bounds to the set of properties that are enforceable using the enforcement paradigm proposed here.

The main limitation of the approach is the difficulty of stating meaningful equivalence relations. This problem is especially acute when the monitor inserts actions into the input stream, rather then suppressing them or truncating the execution. When this

is the case, an equivalence relation often implies that several distinct valid sequences, which are possible transformations of an invalid sequence, must be equivalent. Likewise, it requires that several invalid sequences be considered equivalent if a single valid sequence is a valid alternative to both.

For example, consider the possible equivalence relation of a monitor enforcing the transactional property of Section 6.1, but by correcting invalid transactions, rather than simply suppressing them. Whenever this monitor is presented with an invalid transaction τ, it can complete it by adding whichever actions are necessary to transform this invalid transaction into a valid one. For this to be permissible in a corrective$_{\cong}$ enforcement framework, the completed valid transaction must be thought of as equivalent to its incomplete factor. The equivalence relation given in 6.1 is thus no longer adequate. If an invalid sequence τ can be extended into two possible valid transactions, then both of these transactions must be thought of as equivalent. Furthermore, any other invalid sequence which can be corrected by transforming it into one of these valid sequences must in turn be thought of as equivalent to τ.

A possible solution to this problem is to replace equivalence relations with partial orders, thus organising executions according to a relation which is reflexive, transitive, but not symmetric. Actually, instead of grouping together sequences whose abstraction is similar, a partial order organises them in a gradual manner. In the example of transactional properties, a sequence with more valid transactions would be higher on the partial order than one with less such transactions. The transparency requirement would then be stated by imposing that the output always be higher on the partial order than the input. Not only would this solve the problem highlighted above but it would allow us to objectively describe one valid execution as better than another, which in turn could lead to comparing execution monitors along these lines. We are currently elaborating such an enforcement framework along these lines.

9 Conclusions and future work

In this paper, we propose a framework to analyse the security properties enforceable by monitors capable of transforming their input. By imposing constraints on the enforcement mechanism requiring that some behaviours existing in the input sequence must still be present in the output, we are able to model the desired behaviour 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 behaviour 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 behaviour 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 behaviour that the monitor must preserve is stated in a temporal logic.

Furthermore, as described above, we are also experimenting with an alternate enforcement paradigm that uses partial orders rather than equivalence relations. We believe this will provide a more flexible way to model the corrective behaviour of the monitor. Finally, we are especially interested in the class of transactional properties, since this class includes a number of interesting real-life properties. We are currently experimenting with various ways to enforce the properties in this class. In addition to the suppressing illicit transactions, as was described in this paper, invalid transactions could be corrected or replaced.

References

Alpern, B. and Schneider, F.B. (1985) 'Defining liveness', *Information Processing Letters*, October, Vol. 21, No. 4, pp.181–185.

Alpern, B. and Schneider, F.B. (1987) 'Recognizing safety and liveness', *Distributed Computing*, Vol. 2, No. 3, pp.117–126.

Basin, D., Jugé, V., Klaedtke, F. and Zalinescu, E. (2013) 'Enforceable security policies revisited', *ACM Transactions on Information and System Security*, Vol. 16, No. 1, pp.3:1–3:26.

Bauer, A. (2010) *Monitorability of ω-Regular Languages*, Computing Research Repository (CoRR) abs/1006.3638, Association for Computing Machinery (ACM), June.

Bauer, A. and Jürjens, J. (2010) 'Runtime verification of cryptographic protocols', *Computers & Security*, Vol. 29, No. 3, pp.315–330.

Bauer, A. and Jürjens, J. (2008) 'Security protocols, properties, and their monitoring', in *Proceedings of the Fourth International Workshop on Software Engineering for Secure Systems (SESS)*.

Bauer, A., Leucker, M. and Schallhart, C. (2006) 'Monitoring of real-time properties', in *Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, Vol. 4337 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, December, Springer-Verlag.

Bauer, A., Leucker, M. and Schallhart, C. (2010) 'Runtime verification for LTL and TLTL', *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 20, No. 4, pp.14:1–14:64.

Bauer, L., Ligatti, J. and Walker, D. (2002) 'More enforceable security policies', in *Foundations of Computer Security*, Copenhagen, Denmark, July.

Beauquier, D. and Pin, J-E. (1991) 'Languages and scanners', *Theoretical Computer Science*, Vol. 84, No. 1, pp.3–21.

Beauquier, D., Cohen, J. and Lanotte, R. (2009) 'Security policies enforcement using finite edit automata', *Electr. Notes Theor. Comput. Sci.*, Vol. 229, No. 3, pp.19–35.

Bielova, N. and Massacci, F. (2011) 'Do you really mean what you actually enforced?', *International Journal of Information Security*, Vol. 10, No. 4 , pp.239–254.

Bielova, N., Massacci, F. and Micheletti, A. (2009) 'Towards practical enforcement theories', in *Identity and Privacy in the Internet Age*, *14th Nordic Conference on Secure IT Systems*, *NordSec*, Oslo, Norway, 14–16 October, pp.239–254.

Chabot, H., Khoury, R. and Tawbi, N. (2009) 'Generating in-line monitors for Rabin automata', in *Proceedings of the 14th Nordic Conference on Secure IT Systems*, *NordSec*, Vol. 5838 of *LNCS*, Springer, October, pp.287–301.

Chang, E., Manna, Z. and Pnueli, A. (1991) 'The safety-progress classification', in F.L. Bauer, W. Brauer and H. Schwichtenberg (Eds.): *Logic and Algebra of Specifications*, pp.143–202, Springer-Verlag.

Chudnov, A. and Naumann, D.A. (2010) 'Information flow monitor inlining', *IEEE Computer Security Foundations*, July.

d'Amorim, M. and Roşu, G. (2005) 'Efficient monitoring of omega-languages', in *Computer Aided Verification, 17th International Conference, CAV*, Springer, Edinburgh, Scotland, UK, July 6–10, Vol. 3576 of *Lecture Notes in Computer Science*, pp.364–378.

Falcone, Y., Fernandez, J-C. and Mounier, L. (2008a) 'Synthesizing enforcement monitors wrt. the safety-progress classification of properties', in *Information Systems Security, 4th International Conference, ICISS*, Hyderabad, India, December 16–20, Vol. 5352 of *Lecture Notes in Computer Science*, pp.41–55.

Falcone, Y., Fernandez, J-C. and Mounier, L. (2008b) *Synthesizing Enforcement Monitors wrt. the Safety-progress Classification of Properties*, Technical Report TR-2008-7, Verimag Research Report.

Falcone, Y., Fernandez, J-C. and Mounier, L. (2009a) 'Enforcement monitoring wrt. the safety-progress classification of properties', in *SAC '09: Proceedings of the ACM Symposium on Applied Computing*, ACM, New York, NY, USA, pp.593–600.

Falcone, Y., Fernandez, J-C. and Mounier, L. (2009b) 'Runtime verification of safety-progress properties', in *Runtime Verification: 9th International Workshop (RV 2009)*, Grenoble, France, Selected Papers, Springer-Verlag, Berlin, Heidelberg, pp.40–59.

Fong, P. (2004) 'Access control by tracking shallow execution history', in *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, USA, May.

Hamlen, K.W., Morrisett, J.G. and Schneider, F.B. (2006) 'Computability classes for enforcement mechanisms', *ACM Transactions on Programming Languages and Systems*, Vol. 28, No. 1, pp.175–205.

Jun, P., Xingyuan, C., Bei, W., Xiangdong, D. and Yongliang, W. (2008) 'Policy monitoring and a finite state automata model', in *CSSE '08: Proceedings of the International Conference on Computer Science and Software Engineering*, IEEE Computer Society, Washington, DC, USA, pp.646–649.

Khoury, R. and Tawbi, N. (2012a) 'Corrective enforcement: a new paradigm of security policy enforcement by monitors', *ACM Transactions on Information and System Security*, Vol. 15, No. 2.

Khoury, R. and Tawbi, N. (2012b) 'Which security policies are enforceable by runtime monitors? A survey', *Computer Science Review*, Vol. 6, No. 1, pp.27–45.

Kim, M., Kannan, S., Lee, I., Sokolsky, O. and Viswanathan, M. (2002) 'Computational analysis of run-time monitoring – fundamentals of java-mac', *Electr. Notes Theor. Comput. Sci.*, Vol. 70, No. 4.

Kupferman, O. and Vardi, M.Y. (2001) 'Model checking of safety properties', *Normal Methods in System Design*, Vol. 19, No. 3, pp.291–314.

Kupferman, O., Lustig, Y. and Vardi, M.Y. (2006) 'On locally checkable properties', in *Proceedings of the 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning*, *Lecture Notes in Computer Science*, Springer-Verlag.

Lamport, L. (1977) 'Proving the correctness of multiprocess programs', *IEEE Transactions on Software Engineering*, Vol. 3, No. 2, pp.125–143.

Langar, M. and Mejri, M. (2005) Formal and efficient enforcement of security policies', in *Proceedings of the International Conference on Foundations of Computer Science (FCS)*, pp.143–149.

Langar, M., Mejri, M. and Adi, K. (2006) 'Formal monitor for concurrent programs', in *Workshop on Practice and Theory of IT Security*.

Langar, M., Mejri, M. and Adi, K. (2007) 'A formal approach for security policy enforcement in concurrent programs', in *Proceedings of the International Conference on Security & Management*, pp.165–171.

Le Guernic, G. (2007) 'Automaton-based confidentiality monitoring of concurrent programs', in *Proceedings of the 20th IEEE Computer Security Foundations Symposium (CSFS20)*, IEEE Computer Society, July 6–8, pp.218–232.

Ligatti, J. and Reddy, S. (2010) 'A theory of runtime enforcement, with results', in *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, September.

Ligatti, J., Bauer, L. and Walker, D. (2004) 'Edit automata: enforcement mechanisms for run-time security policies', *International Journal of Information Security*, Vol. 4, Nos. 1–2 , pp.2–16.

Ligatti, J., Bauer, L. and Walker, D. (2005) 'Enforcing non-safety security policies with program monitors', in *10th European Symposium on Research in Computer Security (ESORICS)*, Milan, September.

Mealy, G.H. (1955) 'A method for synthesizing sequential circuits', *Bell System Technical Journal*, Vol. 34, No. 5, pp.1045–1079.

Mechri, T., Langar, M., Mejri, M., Fujita, H. and Funyu, Y. (2007) 'Automatic enforcement of security in computer networks', in *New Trends in Software Methodologies*, *Tools and Techniques – Proceedings of the Sixth SoMeT*, pp.200–222.

Schneider, F.B. (2000) 'Enforceable security policies', *Information and System Security*, Vol. 3, No. 1 pp.30–50.

Sen, K., Vardhan, A., Agha, G. and Roşu, G. (2004) 'Efficient decentralized monitoring of safety in distributed systems', in *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, IEEE Computer Society Washington, DC, USA, pp.418–427.

Sridhar, M. and Hamlen, K.W. (2011) 'Flexible in-lined reference monitor certification: challenges and future directions', in *Proceedings of the 5th ACM Workshop on Programming Languages Meets Program Verification*, *PLPV '11*, ACM, New York, NY, USA, pp.55–60.

Syropoulos, A. (2001) 'Mathematics of multisets', in *Proceedings of the Workshop on Multiset Processing*, Springer-Verlag, pp.347–358.

Talhi, C., Tawbi, N. and Debbabi, M. (2008) 'Execution monitoring enforcement under memory-limitations constraints', *Information and Computation*, Vol. 206, No. 1, pp.158–184.

Viswanathan, M. (2000) *Foundations for the Run-Time Analysis of Software Systems*, PhD thesis, University of Pennsylvania.

Yan, F. and Fong, P.W.L. (2009) 'Efficient IRM enforcement of history-based access control policies', in *Proceedings of the ACM Symposium on Information*, *Computer and Communications Security*, ASIACCS, Sydney, Australia, pp.35–46.

Zhu, G., Tyagi, A. and Roop, P. (2006) *Stream Automata as Run-time Monitors for Open System Security Policies*, Department of Electrical and Computer Engineering, Iowa State University, Ames, Iowa, USA Tech Report 06-101.

Notes

1 This definition, taken from Talhi et al. (2008), is equivalent to the one given in Bauer et al. (2002).

2 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.

3 Other possibilities can be considered, which would more closely follow the specific property being enforced.