Enforceable Security Policies Revisited*

David Basin[1], Vincent Jugé[2], Felix Klaedtke[1], and Eugen Zălinescu[1]

[1] Institute of Information Security, ETH Zurich, Switzerland
[2] MINES ParisTech, France

Abstract. We revisit Schneider's work on policy enforcement by execution monitoring. We overcome limitations of Schneider's setting by distinguishing between system actions that are controllable by an enforcement mechanism and those actions that are only observable, that is, the enforcement mechanism cannot prevent their execution. For this refined setting, we give necessary and sufficient conditions on when a security policy is enforceable. To state these conditions, we generalize the standard notion of safety properties. Our classification of system actions also allows one, for example, to reason about the enforceability of policies that involve timing constraints. Furthermore, for different specification languages, we investigate the decision problem of whether a given policy is enforceable. We provide complexity results and show how to synthesize an enforcement mechanism from an enforceable policy.

1 Introduction

Security policies come in all shapes and sizes, ranging from simple access-control policies to complex data-usage policies governed by laws and regulations. Given their diversity and their omnipresence in regulating processes and data usage in modern IT systems, it is important to have a firm understanding of what kinds of policies can be enforced and to have general tools for their enforcement.

Most conventional enforcement mechanisms are based on some form of execution monitoring. Schneider [29] began the investigation of which kinds of security policies can be enforced this way. In Schneider's setting, an execution monitor runs in parallel with the target system and observes the system's actions just before they are carried out. In case an action leads to a policy violation, the enforcement mechanism terminates the system. Schneider's results on the enforceability of security policies has spurred various research, both practical and theoretical, on developing and analyzing runtime enforcement mechanisms. For instance, Erlingsson and Schneider [12, 13] implement and evaluate enforcement mechanisms based on monitoring. Ligatti and others [24–26] propose more powerful models for enforcement, which can not only terminate a system but also insert and suppress system actions, and they analyze the classes of properties that can be described by such models.

In this paper, we refine Schneider's setting, thereby overcoming several limitations. To explain the limitations, we first summarize Schneider's findings. Schneider [29] shows that only those security policies that can be described by a safety

* This work was partly supported by Google Inc.

P. Degano and J.D. Guttman (Eds.): POST 2012, LNCS 7215, pp. 309–328, 2012.

property [1, 23, 27] on traces are enforceable by execution monitoring. Roughly speaking, (1) inspecting the sequence of system actions is sufficient to determine whether it is policy compliant and (2) nothing bad ever happens on a prefix of a satisfying trace.[1] History-based access-control policies, for example, fall into this class of properties. Furthermore, Schneider defines so-called security automata that recognize the class of safety properties and that "can serve as the basis for an enforcement mechanism" [29, Page 40]. However, Schneider's conditions for enforceability are necessary but not sufficient. In fact, there are safety properties that are not enforceable. This is already pointed out by Schneider [29, Page 41].

We provide a formalization of enforceability for mechanisms similar to Schneider's [29], i.e., monitors that observe system actions and that terminate systems in case of a policy violation. A key aspect of our formalization is that we distinguish between actions that are only observable and those that are also controllable: An enforcement mechanism cannot terminate the system when observing an only-observable action. In contrast, it can prevent the execution of a controllable action by terminating the system. An example of an observable but not controllable action is a clock tick, since one cannot prevent the progression of time. With this classification of system actions, we can derive that, e.g., availability policies with hard deadlines, which require that requests are processed within a given time limit, are not enforceable although they are safety properties. Another example is administrative actions like assigning roles or permissions to users. Such actions change the system state and can be observed but not controlled by most (sub)systems and enforcement mechanisms. However, a subsystem might permit or deny other actions, which it controls, based on the system's current state. Therefore the enforceability of a policy for the subsystem usually depends on this distinction.

In contrast to Schneider, we give also sufficient conditions for the existence of an enforcement mechanism in our setting with respect to a given trace property. This requires that we first generalize the standard notion of safety [1] to account for the distinction between observable and controllable actions. Our necessary and sufficient conditions provide a precise characterization of enforceability that we use for exploring the realizability of enforcement mechanisms for security policies. For different specification languages, we present decidability results for the decision problem that asks whether a given security policy is enforceable. In case of decidability, we also show how to synthesize an enforcement mechanism for the given policy. In particular, we prove that the decision problem is undecidable for context-free languages and PSPACE-complete for regular languages. Moreover, we extend our decidability result by giving a solution to the realizability problem where policies are specified in a temporal logic with metric constraints.

Summarizing, we see our contributions as follows. We overcome limitations of Schneider's setting on policy enforcement based on execution monitoring [29]. First, we distinguish between controllable and observable system actions when monitoring executions. Second, we give conditions for policy enforcement based

[1] Note that a trace property must also be a decidable set to be enforceable, as remarked later by Viswanathan [32] and Hamlen et al. [18].

on execution monitoring that are necessary and also sufficient. These two refinements of Schneider's work allow us to reason about the enforceability of policies that, for instance, involve timing constraints. We also provide results on the decidability of the decision problem of whether a policy is enforceable with respect to different specification languages.

We proceed as follows. In Section 2, we define our notion of enforceability. In Section 3, we relate it to a generalized notion of safety. In Section 4, we analyze the realizability problem for different specification languages. In Sections 5 and 6, we discuss related work and draw conclusions.

2 Enforceability

In this section, we first describe abstractly how enforcement mechanisms monitor systems and prevent policy violations. Afterwards, we define our notion of enforceability.

2.1 Policy Enforcement Based on Execution Monitoring

We take an abstract view of systems and their behaviors similar to Schneider [29] and others [24–26], where executions are finite or infinite sequences over an alphabet Σ. We assume that a system execution generates such a sequence incrementally, starting from the empty sequence ε. In the following, we also call these sequences *traces*. Possible interpretations of the elements in Σ are system actions, system states, or state-action pairs. Their actual meaning is irrelevant for us. However, what is important is that each of these elements is finitely represented and visible to a system observer, and that policies are described in terms of these elements. For convenience, we call the elements in Σ *actions*. Furthermore, we assume that the actions are classified as being either *controllable* actions $C \subseteq \Sigma$ or only *observable* actions $O \subseteq \Sigma$, with $O = \Sigma \setminus C$.

Our abstract system architecture for equipping a system S with an enforcement mechanism E is as follows. Before S executes an action $a \in \Sigma$, E intercepts it and checks whether a's execution violates the given policy P. If the execution of a leads to a policy violation and a is controllable, E terminates S. Otherwise, E does not intervene and S successfully executes a. Note that if the execution of a leads to a policy violation but a is only observable, E detects the violation but cannot prevent it. Hence, in this interaction between S and E, we extend Schneider's setting [29] by distinguishing between controllable and observable actions.

We conclude the description of this system architecture with the following remarks. First, in process algebras like CSP and CCS, S and E are modeled by processes over the action set Σ, and their interaction is the synchronous composition of processes. See, for example, [6], where it is assumed that all actions are controllable. The composed system deadlocks in case of policy violation. Since we distinguish between controllable and observable actions, the process modeling E must always be able to engage in actions in O. Second, instead of assuming that system actions are solely generated by the system S, the enforcement mechanism E can generate observable actions, which are internal and invisible to S.

For instance, the enforcement mechanism can have its own internal clock, which generates clock ticks. Third, instead of action interception and system termination, we could require that S sends a query to E whether executing an action $a \in C$ is authorized. E sends then a permit-or-deny message back to S who proceeds according to E's answer: in case of permit, S executes the action and in case of deny, S continues with an alternative action for which S might need to send a request to E prior to executing it. When executing an action in O, S notifies E of its execution. With this kind of interaction, E's function is similar to a policy decision point (PDP) in standard access-control architectures like XACML.

As pointed out by Schneider [29], a necessary condition for enforcing a policy by execution monitoring is that policy compliance is determined by the observed trace. We therefore require that a policy P is a *property of traces*, i.e., $P \subseteq \Sigma^* \cup \Sigma^\omega$, where Σ^* is the set of finite sequences over Σ and Σ^ω is the set of infinite sequences over Σ. We also write Σ^∞ for $\Sigma^* \cup \Sigma^\omega$. Since systems might not terminate—in fact, they often should not terminate—we also consider infinite traces, which describe system behaviors in the limit.

Another necessary condition for enforceability is that the decision of whether the enforcement mechanism E terminates the system S cannot depend on possible future actions [29]. This point is reflected in how and when E checks policy compliance in its interaction with S: E's decision depends on whether τa is in P, where a is the intercepted action and τ is the trace of the previously executed actions.

Additionally, although implicit in Schneider's work [29], there are *soundness* and *transparency* requirements for an enforcement mechanism [11, 18, 24, 25]. Soundness means that the enforcement mechanism must prevent system executions that are not policy compliant. Transparency means that the enforcement mechanism must not terminate system executions that are policy compliant. These requirements clearly restrict the class of trace properties that can be enforced by the interaction described above between S and E.

2.2 Formalization

Checking whether the execution of an action is policy compliant is at the core of any enforcement mechanism. The maximum information available to check is the intercepted action a together with the already executed trace τ. Our formalization of enforceability requires the existence of a Turing machine that carries out these checks. In particular, for every check, the Turing machine must terminate, either accepting or rejecting the input τa. Accepting the input means that executing a is policy compliant whereas rejecting τa means that a's execution leads to a policy violation. We do not formalize the interaction between the enforcement mechanism and the system and how actions are intercepted.

Prior to formalizing enforceability, we first introduce the following definitions. For a sequence $\sigma \in \Sigma^\infty$, we denote the set of its prefixes by $\mathrm{pre}(\sigma)$ and the set of its finite prefixes by $\mathrm{pre}_*(\sigma)$, i.e., $\mathrm{pre}_*(\sigma) := \mathrm{pre}(\sigma) \cap \Sigma^*$. The *truncation* of $L \subseteq \Sigma^*$ is $\mathrm{trunc}(L) := \{\sigma \in \Sigma^* \mid \mathrm{pre}(\sigma) \subseteq L\}$ and its *limit closure* is $\mathrm{cl}(L) := L \cup \{\sigma \in \Sigma^\omega \mid \mathrm{pre}_*(\sigma) \subseteq L\}$. Note that $\mathrm{trunc}(L)$ is the largest subset

of L that is prefix-closed and $\mathrm{cl}(L)$ contains, in addition to the sequences in L, the infinite sequences whose finite prefixes are all elements of L. Furthermore, for $L \subseteq \Sigma^*$ and $K \subseteq \Sigma^\infty$, we define $L \cdot K := \{\sigma\tau \in \Sigma^\infty \mid \sigma \in L \text{ and } \tau \in K\}$. For generality, we formalize enforceability relative to a *trace universe* U, which is a nonempty prefix-closed subset of Σ^∞.

Definition 1. *Let Σ be a set of actions. The property of traces $P \subseteq \Sigma^\infty$ is* enforceable *in the trace universe $U \subseteq \Sigma^\infty$ with the observable actions in $O \subseteq \Sigma$,* (U, O)-enforceable *for short, if there is a deterministic Turing machine $\mathcal{M}$ with the following properties, where $A \subseteq \Sigma^*$ is the set of inputs accepted by $\mathcal{M}$:*

(i) $\mathcal{M}$ *halts on the inputs in* $(\mathrm{trunc}(A) \cdot \Sigma) \cap U$.
(ii) $\mathcal{M}$ *accepts the inputs in* $(\mathrm{trunc}(A) \cdot O) \cap U$.
(iii) $\mathrm{cl}(\mathrm{trunc}(A)) \cap U = P \cap U$.
(iv) $\varepsilon \in A$.

Intuitively, with property (i) we ensure that whenever the enforcement mechanism E checks whether τa is policy compliant by using the Turing machine $\mathcal{M}$ (when intercepting the action $a \in \Sigma$), then E obtains an answer from $\mathcal{M}$. Note that we require that the trace τ produced so far by the system S is in $\mathrm{trunc}(A)$ and not in A, since if there is a prefix of τ that is not accepted by $\mathcal{M}$, then E would have terminated S earlier. Furthermore, we are only interested in traces in the universe U. Property (ii) states that $A \supseteq (\mathrm{trunc}(A) \cdot O) \cap U$ and we guarantee with it that a finite trace τa with $a \in O$ is policy compliant provided that $\tau a \in U$ and τ is policy compliant. Property (iii) relates the policy P with the inputs accepted by $\mathcal{M}$. Note that $\mathrm{cl}(\mathrm{trunc}(A)) \cap U \subseteq P \cap U$ formalizes the soundness requirement for an enforcement mechanism and $\mathrm{cl}(\mathrm{trunc}(A)) \cap U \supseteq P \cap U$ formalizes the transparency requirement. With property (iv) we ensure that the system S is initially policy compliant.

We illustrate Definition 1 by determining whether the following two policies are enforceable.

Example 2. The policy P_1 requires that whenever there is a *fail* action then there must not be a *login* action for at least 3 time units. The policy P_2 requires that every occurrence of a *request* action must be followed by a *deliver* action within 3 time units provided the system does not stop in the meanwhile. We give their trace sets below. We assume, for the ease of exposition, that actions do not happen simultaneously and whenever time progresses by one time unit, the system sends a *tick* action to the enforcement mechanism. However, more than one action can be executed in a single time unit.

Let Σ be the action set $\{tick, fail, login, request, deliver\}$. The trace universe $U \subseteq \Sigma^\infty$ consists of all infinite traces containing infinitely many *tick* actions and their finite prefixes. This models that time does not stop. We define P_1 as the complement with respect to U of the limit closure of

$$\big\{a_1 \dots a_n \in \Sigma^* \;\big|\; \text{there are } i, j \in \{1, \dots, n\} \text{ with } i < j \text{ such that } a_i = fail, \; a_j = login, \text{ and } a_{i+1} \dots a_{j-1} \text{ contains 3 or fewer } tick \text{ actions}\big\}$$

and P_2 as the complement with respect to U of the limit closure of

$$\{a_1 \dots a_n \in \Sigma^* \mid \text{there are } i, j \in \{1, \dots, n\} \text{ with } i < j \text{ such that } a_i = \mathit{request} \text{ and } a_{i+1} \dots a_j \text{ contains no } \mathit{deliver} \text{ action and more than 3 } \mathit{tick}\text{s}\}.$$

A *tick* action is only observable by an enforcement mechanism since the enforcement mechanism cannot prevent the progression of time. It is also reasonable to assume that *fail* actions are only observable since otherwise an enforcement mechanism could prevent the failure from happening in the first place. Hence we define $O := \{\mathit{tick}, \mathit{fail}\}$.

It is straightforward to define a Turing machine $\mathcal{M}$ as required in Definition 1, showing that P_1 is (U, O)-enforceable. Intuitively, whenever the enforcement mechanism observes a *fail* action, it prevents all *login* actions until it has observed sufficiently many *tick* actions. This requires that *login* actions are controllable, whereas the actions *tick* and *fail* need only be observed by the enforcement mechanism.

The set of traces P_2 is not (U, O)-enforceable. The reason is that whenever an enforcement mechanism observes a *request* action, it cannot terminate the system in time to prevent a policy violation when no *deliver* action occurs within the given time bound. This is because the enforcement mechanism cannot prevent the progression of time. More precisely, assume that there exists a Turing machine $\mathcal{M}$ required in Definition 1, which must accept the trace $\mathit{request}\ \mathit{tick}^3 \in P_2 \cap U$. But then, by condition (ii) of Definition 1, it also must accept the trace $\mathit{request}\ \mathit{tick}^4 \notin P_2 \cap U$.

Natural questions that arise from Definition 1 are (1) for which class of trace properties does such a Turing machine $\mathcal{M}$ exist, (2) for which specification languages can we decide whether such a Turing machine $\mathcal{M}$ exists, and (3) when a policy is enforceable, can we synthesize from its given description an enforcement mechanism? We investigate these questions in the next two sections.

3 Relation between Enforceability and Safety

In this section, we characterize the class of trace properties that are enforceable with respect to Definition 1. To provide this characterization, we first generalize the standard notions of safety properties [1, 19].

3.1 Generalizing Safety

According to Lamport [23], a safety property intuitively states that nothing bad ever happens. A widely accepted formalization of this intuition, from Alpern and Schneider [1], is as follows: the set $P \subseteq \Sigma^\omega$ is ω-*safety* if

$$\forall \sigma \in \Sigma^\omega.\ \sigma \notin P \rightarrow \exists i \in \mathbb{N}.\ \forall \tau \in \Sigma^\omega.\ \sigma^{<i}\tau \notin P,$$

where $\sigma^{<i}$ denotes the prefix of σ of length i. In particular, $\sigma^{<0}$ is the empty sequence ε. Alpern and Schneider's definition takes only infinite sequences into

account. Their definition, however, straightforwardly generalizes to finite and infinite sequences: the set $P \subseteq \Sigma^\infty$ is ∞-*safety* if

$$\forall \sigma \in \Sigma^\infty.\ \sigma \notin P \rightarrow \exists i \in \mathbb{N}.\ \forall \tau \in \Sigma^\infty.\ \sigma^{<i}\tau \notin P\,,$$

where $\sigma^{<i} = \sigma$ if σ is finite and $i \in \mathbb{N}$ is greater than or equal to σ's length.

Note that ω-safety is not directly related to enforceability, since an enforcement mechanism monitors finite traces and ω-safety restricts the infinite traces in a set of traces. Moreover, note that ω-safety and ∞-safety differ even on sets of infinite sequences. For instance, the set $\{a\} \cdot \Sigma^\omega$ is ω-safety, since for every infinite sequence σ that does not start with a, no extension of $\sigma^{<1}$ is in $\{a\} \cdot \Sigma^\omega$. However, $\{a\} \cdot \Sigma^\omega$ is not ∞-safety, since we can extend the empty sequence, which is not in $\{a\} \cdot \Sigma^\omega$, by an infinite sequence τ that starts with the letter a. In general, whenever a policy $P \subseteq \Sigma^\infty$ is ∞-safety, the set $P \cap \Sigma^\omega$ of its infinite traces is ω-safety, whereas the converse is invalid.

In Definition 3 below, we give our generalized notion of safety, which is parametric in the universe U. The sets Σ^ω and Σ^∞ used in the definitions for ω-safety and ∞-safety are just two instances for U. This generalization is similar to Henzinger's [19] definitions of safety and liveness, which extends the classical safety-liveness classification for properties of untimed systems [1] to real-time settings. Furthermore, our definition is parametric in the set $O \subseteq \Sigma$. Intuitively, if a trace $\sigma \in U$ violates P, then this violation must be caused by a finite prefix of σ not ending with an element in O.

Definition 3. *Let $U \subseteq \Sigma^\infty$ and $O \subseteq \Sigma$. The set $P \subseteq \Sigma^\infty$ is* (U, O)-safety *if*

$$\forall \sigma \in U.\ \sigma \notin P \rightarrow \exists i \in \mathbb{N}.\ \sigma^{<i} \notin \Sigma^* \cdot O \wedge \forall \tau \in \Sigma^\infty.\ \sigma^{<i}\tau \notin P \cap U\,.$$

In the following examples, we illustrate this generalized notion of safety.

Example 4. Both the policies P_1 and P_2 from Example 2 are ∞-safety. If a trace τ violates P_1 then the violation can be pinpointed to a position where a *login* action is executed, i.e., there is some $i \geq 1$ with $\tau^{<i-1} \in P_1$, $\tau^{<i} \notin P_1$, and $\tau^{<i}$ ends with a *login* action. No matter how we extend $\tau^{<i}$, the extension still violates P_1. Analogously for P_2, policy violations are caused by *tick* actions instead of *login* actions.

However, P_1 is (U, O)-safety and P_2 is not (U, O)-safety, where U and O are as in Example 2. A violation of P_1 is caused by executing a *login* action, i.e., $\tau \in P_1$ and $\tau\, \mathit{login} \notin P_1$. We cannot extend such an execution so that the resulting extended trace is policy compliant. For P_2, a violation is caused by a *tick*. Here, the prefix excluding this *tick* action can be extended to a trace that is in P_2. Namely, we discharge the *request* action in the prefix by adding a *deliver* action.

Example 5. Recall the trace universe $U \subseteq \Sigma^\infty$ from Example 2, where $\Sigma = \{\mathit{tick}, \mathit{fail}, \mathit{login}, \mathit{request}, \mathit{deliver}\}$. It consists of the infinite traces with infinitely many *tick* actions and their finite prefixes. The trace property "always eventually a *tick* action," formalized as follows, is not safety:

$$\begin{aligned} P := {} & \{\varepsilon\} \cup \{a_0 \ldots a_n \in \Sigma^* \mid n \in \mathbb{N} \text{ and } a_n = \mathit{tick}\} \cup \\ & \{a_0 a_1 \cdots \in \Sigma^\omega \mid \text{for all } i \in \mathbb{N}, \text{ there is some } j \in \mathbb{N} \text{ with } j \geq i \text{ and } a_j = \mathit{tick}\}\,. \end{aligned}$$

When considering only the infinite traces, the trace property $P \cap \Sigma^\omega$ is not ω-safety. In fact, according to Alpern and Schneider [1], $P \cap \Sigma^\omega$ is a liveness property.

P is also not $(U, \emptyset)$-safety since any nonempty trace $a_0 \dots a_n$ with $a_n \neq \mathit{tick}$ is in $U \setminus P$ and can be extended to the trace $a_0 \dots a_n\, \mathit{tick}$, which is in $P \cap U$. However, when we exclude finite traces from U, then P is $(U \cap \Sigma^\omega, \emptyset)$-safety, since $P \cap \Sigma^\omega = U \cap \Sigma^\omega$.

Lemma 6 below characterizes (U, O)-safety in terms of prefix sets and limit closures. For a set of sequences $L \subseteq \Sigma^\infty$, we abbreviate $\bigcup_{\sigma \in L} \mathrm{pre}(\sigma)$ by $\mathrm{pre}(L)$ and $\bigcup_{\sigma \in L} \mathrm{pre}_*(\sigma)$ by $\mathrm{pre}_*(L)$.

Lemma 6. *Let $U \subseteq \Sigma^\infty$ be a trace universe and $O \subseteq \Sigma$. The set $P \subseteq \Sigma^\infty$ is (U, O)-safety iff $\mathrm{cl}(\mathrm{pre}_*(P \cap U) \cdot O^*) \cap U \subseteq P$.*

Proof. We rephrase Definition 3 in terms of set containment, from which we conclude the stated equivalence.

We first show that the set $P \subseteq \Sigma^\infty$ is (U, O)-safety iff $\forall \sigma \in U.\ \sigma \notin P \rightarrow \mathrm{pre}_*(\sigma) \not\subseteq \mathrm{pre}_*(P \cap U) \cdot O^*$. We start with the left to right implication. Suppose that P is (U, O)-safety and let $\sigma \in U$. Assume that $\sigma \notin P$. Then there is an index $i \in \mathbb{N}$ such that (1) $\sigma^{<i} \notin \Sigma^* \cdot O$ and (2) $\sigma^{<i}\tau \notin P \cap U$, for all $\tau \in \Sigma^\infty$. (2) establishes that $\sigma^{<i} \notin \mathrm{pre}_*(P \cap U)$ and, together with (1), that $\sigma^{<i} \notin \mathrm{pre}_*(P \cap U) \cdot O^*$. As $\sigma^{<i} \in \mathrm{pre}_*(\sigma)$, we obtain that $\mathrm{pre}_*(\sigma) \not\subseteq \mathrm{pre}_*(P \cap U) \cdot O^*$. We now prove the right to left implication. Let $\sigma \in U \setminus P$. Then $\mathrm{pre}_*(\sigma) \not\subseteq \mathrm{pre}_*(P \cap U) \cdot O^*$, and thus there is an index $i \in \mathbb{N}$ such that $\sigma^{<i} \notin \mathrm{pre}_*(P \cap U) \cdot O^*$. Let $\sigma_1, \sigma_2 \in \Sigma^*$ be such that $\sigma^{<i} = \sigma_1\sigma_2$, $\sigma_1 \notin \Sigma^* \cdot O$, and $\sigma_2 \in O^*$. Hence $\sigma_1 \notin \mathrm{pre}_*(P \cap U)$, that is $\sigma_1\tau \notin P \cap U$, for all $\tau \in \Sigma^\infty$. It follows that P is (U, O)-safety.

The statement $\forall \sigma \in U.\ \sigma \notin P \rightarrow \mathrm{pre}_*(\sigma) \not\subseteq \mathrm{pre}_*(P \cap U) \cdot O^*$ is equivalent to $\forall \sigma \in U.\ \sigma \in P \leftarrow \mathrm{pre}_*(\sigma) \subseteq \mathrm{pre}_*(P \cap U) \cdot O^*$. Since $\mathrm{pre}_*(P \cap U) \cdot O^*$ is prefix-closed, it is also equivalent to $\forall \sigma \in U.\ \sigma \in P \leftarrow \sigma \in \mathrm{cl}(\mathrm{pre}_*(P \cap U) \cdot O^*)$, i.e., $\mathrm{cl}(\mathrm{pre}_*(P \cap U) \cdot O^*) \cap U \subseteq P$. □

Note that $P \cap U \subseteq \mathrm{cl}(\mathrm{pre}_*(P \cap U) \cdot O^*) \cap U$, for any sets $P, U \subseteq \Sigma^\infty$ and $O \subseteq \Sigma$. Therefore, $P \subseteq \Sigma^\infty$ is (U, O)-safety iff $\mathrm{cl}(\mathrm{pre}_*(P \cap U) \cdot O^*) \cap U = P \cap U$.

3.2 Characterizing Enforceability

In the following, we generalize Schneider's [29] statement that ∞-safety is a necessary condition for a security policy to be enforceable by execution monitoring. First, we distinguish between controllable actions C and observable actions O. Second, we take a trace universe U into account. In Schneider's setting, $U = \Sigma^\infty$ and $O = \emptyset$. Third, we show that a policy $P \subseteq \Sigma^\infty$ must satisfy additional conditions to be enforceable. Finally, we show that our conditions are not only necessary, but also sufficient.

Theorem 7. *Let $U \subseteq \Sigma^\infty$ be a trace universe such that $U \cap \Sigma^*$ is a decidable set and let $O \subseteq \Sigma$. The set $P \subseteq \Sigma^\infty$ is (U, O)-enforceable iff the following conditions are satisfied:*

(1) P is (U, O)-safety,
(2) $\mathrm{pre}_*(P \cap U)$ is a decidable set, and
(3) $\varepsilon \in P$.

Proof. We start with the implication from left to right. Assume that $P \subseteq \Sigma^\infty$ is (U, O)-enforceable. Let $A \subseteq \Sigma^*$ be the set of inputs accepted by a Turing machine $\mathcal{M}$ determined by Definition 1. The set A satisfies the following properties: (a) $(\mathrm{trunc}(A) \cdot O) \cap U \subseteq A$, (b) $\mathrm{cl}(\mathrm{trunc}(A)) \cap U = P \cap U$, and (c) $\varepsilon \in A$.

First, we prove that P is (U, O)-safety. Let $\sigma \in U$ be a trace such that $\sigma \notin P$. Then, from (b), we have that $\sigma \notin \mathrm{cl}(\mathrm{trunc}(A))$. Hence there is an index $i \in \mathbb{N}$ such that $\sigma^{<i} \notin A$. Let i be the minimal index with this property. Then $i > 0$ and all proper prefixes of $\sigma^{<i}$ are in A, and hence $\sigma^{<i-1}$ is in $\mathrm{trunc}(A)$. Let $a \in \Sigma$ be such that $\sigma^{<i} = \sigma^{<i-1}a$. We have that $a \notin O$, as otherwise, from (a), $\sigma^{<i} \in A$, which is a contradiction. Hence $\sigma^{<i} \notin \Sigma^* \cdot O$. Moreover, as $\sigma^{<i} \notin A$, for any trace $\tau \in \Sigma^\infty$, we have that $\sigma^{<i}\tau \notin \mathrm{cl}(\mathrm{trunc}(A))$, that is, $\sigma^{<i}\tau \notin P$. This shows that σ satisfies the right hand side of the implication in Definition 3. Hence P is (U, O)-safety.

Second, note that A is not necessarily decidable, as $\mathcal{M}$ need not halt on all inputs in Σ^*. Since $U \cap \Sigma^*$ is decidable by assumption, there is a Turing machine $\mathcal{M}_U$ that terminates on Σ^* and that accepts $U \cap \Sigma^*$. Let $\mathcal{M}_{\mathrm{trunc}}$ be the following Turing machine. For an input $\sigma \in \Sigma^*$, $\mathcal{M}_{\mathrm{trunc}}$ executes steps 1 to 5 until it either accepts or rejects σ:

1. if $\mathcal{M}_U$ rejects σ, then $\mathcal{M}_{\mathrm{trunc}}$ rejects σ;
2. if $\sigma = \varepsilon$, then $\mathcal{M}_{\mathrm{trunc}}$ accepts σ;
3. if n is the length of σ and $\mathcal{M}_{\mathrm{trunc}}$ rejects $\sigma^{<n-1}$, then $\mathcal{M}_{\mathrm{trunc}}$ rejects σ;
4. if $\mathcal{M}$ accepts σ, then $\mathcal{M}_{\mathrm{trunc}}$ accepts σ;
5. otherwise, $\mathcal{M}_{\mathrm{trunc}}$ rejects σ.

It follows by induction over the length of σ that $\mathcal{M}_{\mathrm{trunc}}$ halts on σ and that $\mathcal{M}_{\mathrm{trunc}}$ accepts σ iff $\sigma \in \mathrm{trunc}(A) \cap U$. Therefore, $\mathrm{trunc}(A) \cap U$ is decidable. We have $\mathrm{pre}_*(P \cap U) = \mathrm{pre}_*(\mathrm{cl}(\mathrm{trunc}(A)) \cap U) = \mathrm{trunc}(A) \cap U \cap \Sigma^*$. Because both $\mathrm{trunc}(A) \cap U$ and $U \cap \Sigma^*$ are decidable, so is $\mathrm{pre}_*(P \cap U)$.

Third, as $\varepsilon \in A$ and $\varepsilon \in U$, we have $\varepsilon \in \mathrm{cl}(\mathrm{trunc}(A)) \cap U = P \cap U \subseteq P$.

We now prove the implication from right to left. Assume that P is (U, O)-safety, $\mathrm{pre}_*(P \cap U)$ is a decidable set, and $\varepsilon \in P$. We prove properties (i)–(iv) of Definition 1. As $\mathrm{pre}_*(P \cap U)$ is decidable, there is a Turing machine that halts on all inputs in Σ^* and accepts the set $A := \mathrm{pre}_*(P \cap U)$. Property (i) follows trivially. Property (iv) is also immediate as $\varepsilon \in P \cap U$. As $A = \mathrm{pre}_*(P \cap U)$ is prefix-closed, $\mathrm{trunc}(A) = A = \mathrm{pre}_*(P \cap U)$. It remains to be shown that (ii) $(\mathrm{pre}_*(P \cap U) \cdot O) \cap U \subseteq \mathrm{pre}_*(P \cap U)$ and (iii) $\mathrm{cl}(\mathrm{pre}_*(P \cap U)) \cap U = P \cap U$. By Lemma 6, and since P is (U, O)-safety, we have:

- $(\mathrm{pre}_*(P \cap U) \cdot O) \cap U \subseteq \mathrm{cl}(\mathrm{pre}_*(P \cap U) \cdot O^*) \cap U \cap \Sigma^* = P \cap U \cap \Sigma^* \subseteq \mathrm{pre}_*(P \cap U)$;
- $P \cap U \subseteq \mathrm{cl}(\mathrm{pre}_*(P \cap U)) \cap U \subseteq \mathrm{cl}(\mathrm{pre}_*(P \cap U) \cdot O^*) \cap U = P \cap U$.

Therefore, P is (U, O)-enforceable. □

4 Realizability

In this section, we investigate the realizability problem for enforcement mechanisms for security policies. We examine this problem for two policy specification formalisms, based on automata and temporal logic.

4.1 Automata-Based Specification Languages

Automata may be used to give direct, operational specifications of security policies [24, 25, 29]. For instance, Schneider [29] introduces security automata as a formalism for specifying and implementing the decision making of enforcement mechanisms. Given a deterministic security automaton $\mathcal{A}$, the enforcement mechanism E stores $\mathcal{A}$'s current state and whenever E intercepts an action, it updates the stored state using $\mathcal{A}$'s transition function. If there is no outgoing transition and the action is controllable, then E terminates the system. Nondeterministic security automata are handled analogously by storing and updating finite sets of states. In this case, E terminates the system if the set of states becomes empty during an update.

Roughly speaking, if all actions are controllable then the existence of a security automaton specifying a policy implies that the policy is enforceable. This is because security automata characterize the class of trace properties that are ∞-safety. However, if there are actions that are only observable, the existence of a security automaton is insufficient to conclude that the policy is enforceable. Additional checks are needed. We show that these checks can be carried out algorithmically for policies described by finite-state automata. In contrast to security automata, a finite-state automaton has a finite set of states and a finite alphabet, and not all its states are accepting. Furthermore, we delimit the boundary between decidability and undecidability by showing that for a more expressive automata model, namely, pushdown automata, the realizability problem is undecidable.

We start by defining pushdown and finite-state automata. Since trace properties are sets of finite and infinite sequences, we equip the automata with two sets of accepting states, one for finite sequences and the other for infinite sequences.

A *pushdown automaton (PDA)* $\mathcal{A}$ is a tuple $(Q, \Sigma, \Gamma, \delta, q_{\mathrm{I}}, F, B)$, where (1) Q is a finite set of states, (2) Σ is a finite nonempty alphabet, (3) Γ is a finite stack alphabet with $\# \in \Gamma$, (4) $\delta : Q \times \Sigma \times \Gamma \to 2^{Q \times \Gamma^*}$ is the transition function, where $\delta(q, a, b)$ is a finite set, for all $q \in Q$, $a \in \Sigma$, and $b \in \Gamma$, (5) $q_{\mathrm{I}} \in Q$ is the initial state, (6) $F \subseteq Q$ is the set of accepting states for finite sequences, and (7) $B \subseteq Q$ is the set of accepting states for infinite sequences. The *size* of $\mathcal{A}$, denoted by $\|\mathcal{A}\|$, is the cardinality of Q.

A *configuration* of $\mathcal{A}$ is a pair (q, u) with $q \in Q$ and $u \in \Gamma^*$. A *run* of $\mathcal{A}$ on the finite sequence $a_0 \dots a_{n-1} \in \Sigma^*$ is a sequence of configurations (q_0, u_0) $(q_1, u_1) \dots (q_n, u_n)$ with $(q_0, u_0) = (q_{\mathrm{I}}, \#)$ and for all $i \in \mathbb{N}$ with $i < n$, it holds that $u_i = vb$, $(q_{i+1}, w) \in \delta(q_i, a_i, b)$, and $u_{i+1} = vw$, for some $v, w \in \Gamma^*$ and $b \in \Gamma$. The run is *accepting* if $q_n \in F$. Runs over infinite sequences are defined analogously. The infinite sequence $(q_0, u_0)(q_1, u_1) \cdots \in (Q \times \Gamma^*)^\omega$ is a *run* on

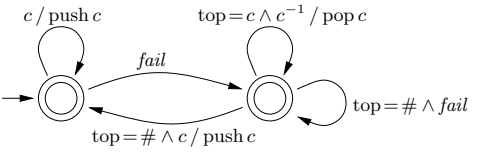

Fig. 1. Pushdown automaton, where c ranges over the elements in C

the infinite sequence $a_0 a_1 \cdots \in \Sigma^\omega$ if $(q_0, u_0) = (q_\mathrm{I}, \#)$ and for all $i \in \mathbb{N}$, it holds that $u_i = vb$, $(q_{i+1}, w) \in \delta(q_i, a_i, b)$, and $u_{i+1} = vw$, for some $v, w \in \Gamma^*$ and $b \in \Gamma$. The run is *accepting* if it fulfills the Büchi acceptance condition, i.e., for every $i \in \mathbb{N}$, there is some $j \in \mathbb{N}$ with $j \geq i$ and $q_j \in B$. In other words, the run visits a state in B infinitely often. We define $L(\mathcal{A}) := L_*(\mathcal{A}) \cup L_\omega(\mathcal{A})$, where

$$L_\circ(\mathcal{A}) := \{\sigma \in \Sigma^\circ \mid \text{there is an accepting run of } \mathcal{A} \text{ on } \sigma\},$$

for $\circ \in \{*, \omega\}$.

We say that $\mathcal{A}$ is a *finite-state automaton (FSA)* if its transitions do not depend on the stack content, i.e., $\delta(q, a, b) = \delta(q, a, b')$, for all $q \in Q$, $a \in \Sigma$, and $b, b' \in \Gamma$. In this case, we may omit the stack alphabet Γ and assume that δ is of type $Q \times \Sigma \to 2^Q$. Runs over finite and infinite sequences simplify then to sequences in Q^* and Q^ω, respectively.

PDAs are more expressive than FSAs, as illustrated by the following example.

Example 8. Let C and C^{-1} be finite nonempty sets of actions with $C^{-1} = \{c^{-1} \mid c \in C\}$. That is, every action $c \in C$ has a corresponding "undo" action $c^{-1} \in C^{-1}$. Consider the policy stating that whenever a *fail* action is executed the system must backtrack before continuing. That is, consider the language $L := \mathrm{pre}(F^* \cdot C^\omega) \cup F^\omega$ over the alphabet $\Sigma := C \cup C^{-1} \cup \{\mathit{fail}\}$, with $F := \{c_1 \ldots c_n \mathit{fail}\, c_n^{-1} \ldots c_1^{-1} \mid n \in \mathbb{N} \text{ and } c_1, \ldots, c_n \in C\}$, where the superscripts $*$ and ω denote here the finite and infinite concatenation of languages, respectively. The PDA in Figure 1, where both states are accepting for both finite and infinite sequences, recognizes this language. However, no FSA accepts this language.

Observe that this policy is $(\Sigma^\infty, \emptyset)$-enforceable. Indeed, the conditions in Theorem 7 are satisfied: (1) L contains the empty sequence, (2) $\mathrm{pre}_*(L) = F^* \cdot (C^* \cup C^* \cdot F)$ is decidable, and (3) $\mathrm{cl}(\mathrm{pre}_*(L)) = F^\omega \cup F^* \cdot (C^\infty \cup C^* \cdot F) = L$ is $(\Sigma^\infty, \emptyset)$-safety. The policy is not $(\Sigma^\infty, \{\mathit{fail}\})$-enforceable, since an enforcement mechanism must terminate the system when intercepting the second *fail* action in the trace $c_1 c_2 \mathit{fail}\, c_2^{-1} \mathit{fail}\, c_1^{-1}$.

We now turn to the decision problem of checking whether a policy given as a PDA or FSA is enforceable. In each case, we first analyze the related decision problem of checking whether a policy is a safety property.

Theorem 9. *Let Σ be the alphabet $\{0, 1\}$. It is undecidable to determine for a PDA $\mathcal{A}$ with alphabet Σ whether $L(\mathcal{A})$ is $(\Sigma^\infty, \emptyset)$-safety.*

Proof. Recall that the universality problem for context-free grammars is undecidable [20]. That means, we cannot decide if $L_*(\mathcal{A}) = \Sigma^*$, for a given PDA $\mathcal{A}$.

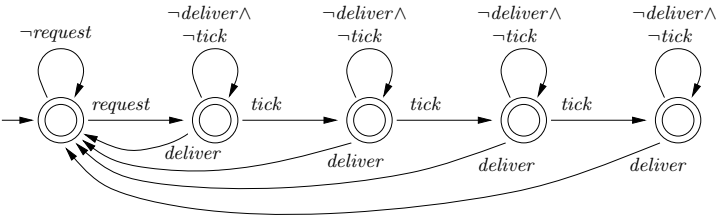

Fig. 2. Finite-state automaton

Given a PDA $\mathcal{A}$, we build a PDA $\mathcal{A}'$ with $L(\mathcal{A}') = L(\mathcal{A}) \cup \Sigma^\omega$. Thus we have that $L(\mathcal{A}') = L_*(\mathcal{A}) \cup \Sigma^\omega$ and $\mathrm{cl}(\mathrm{pre}_*(L(\mathcal{A}'))) = \Sigma^\infty$. Then, from Lemma 6, $L(\mathcal{A}')$ is $(\Sigma^\infty, \emptyset)$-safety iff $L_*(\mathcal{A}) = \Sigma^*$. □

Theorem 10. *Let Σ be the alphabet $\{0, 1\}$. It is undecidable to determine for a PDA $\mathcal{A}$ with alphabet Σ whether $L(\mathcal{A})$ is $(\Sigma^\infty, \emptyset)$-enforceable.*

Proof. From $\mathcal{A}$ we build a PDA $\mathcal{A}'$ with $L(\mathcal{A}') = L(\mathcal{A}) \cup \Sigma^\omega \cup \{\varepsilon\}$. Note that $\mathrm{pre}_*(L(\mathcal{A}')) = \Sigma^*$ is decidable and that $\varepsilon \in L(\mathcal{A})$. Moreover, one can decide whether $\varepsilon \in L_*(\mathcal{A})$ but not whether $L_*(\mathcal{A}) = \Sigma^*$. Hence one cannot decide whether $\Sigma^* = L_*(\mathcal{A}) \cup \{\varepsilon\}$. By Theorem 7, the language $L(\mathcal{A}')$ is $(\Sigma^\infty, \emptyset)$-enforceable iff $L(\mathcal{A}')$ is $(\Sigma^\infty, \emptyset)$-safety iff $\Sigma^* = L_*(\mathcal{A}) \cup \{\varepsilon\}$. □

It is straightforward to define FSAs that recognize the languages P_1 and P_2 from Example 2. For instance, the FSA depicted in Figure 2 recognizes P_2. Since this FSA is deterministic, it is easy to check that the recognized language is not (U, O)-safety and therefore also not (U, O)-enforceable, where U and O are as in Example 2. There is a state from which the observable *tick* action leads to nonacceptance of the input sequence. In general, the problem is PSPACE-complete as shown in Corollary 12 below.

Theorem 11. *Let $\mathcal{U}$ be an FSA over the alphabet Σ such that $L(\mathcal{U})$ is a trace universe and let $O \subseteq \Sigma$. The decision problem of determining, for an FSA $\mathcal{A}$ over Σ, whether $L(\mathcal{A})$ is $(L(\mathcal{U}), O)$-safety, is PSPACE-complete.*

Proof. Recall that the universality problem for FSAs, that is, deciding whether $L_*(\mathcal{A}) = \Sigma^*$ for a given FSA $\mathcal{A}$, is PSPACE-complete [20].

Given an FSA $\mathcal{A}$, we build an FSA $\mathcal{A}'$ with $L(\mathcal{A}') = L(\mathcal{A}) \cup \Sigma^\omega$. As in the proof of Theorem 9, $L(\mathcal{A}')$ is $(\Sigma^\infty, \emptyset)$-safety iff $L_*(\mathcal{A}) = \Sigma^*$. This proves that checking whether $L(\mathcal{A}')$ is $(L(\mathcal{U}), O)$-safety is PSPACE-hard.

To establish membership in PSPACE, we first show how to build, for a given FSA $\mathcal{X} = (Q, \Sigma, \delta, q_\mathrm{I}, F, B)$, two FSAs $\mathcal{Y}$ and $\mathcal{Z}$ such that $L(\mathcal{Y}) = \mathrm{pre}_*(L(\mathcal{X}))$ and, if $L(\mathcal{X}) \cap \Sigma^* = \mathrm{pre}_*(L(\mathcal{X}))$ then $L(\mathcal{Z}) = \mathrm{cl}(L(\mathcal{X}) \cap \Sigma^*)$:

- Let B' be the set of states $q \in B$ that are on a cycle in $\mathcal{X}$. Let $F_\mathcal{Y}$ be the set of states $q \in Q$ for which there is a path in $\mathcal{X}$ starting in q and ending in a state of $F \cup B'$. The FSA $\mathcal{Y} := (Q, \Sigma, \delta, q_\mathrm{I}, F_\mathcal{Y}, \emptyset)$ accepts the language $L(\mathcal{Y}) = \mathrm{pre}_*(L(\mathcal{X}))$.
- If $\mathrm{pre}_*(L(\mathcal{X})) = L(\mathcal{X}) \cap \Sigma^*$, the FSA $\mathcal{Z} := (Q, \Sigma, \delta, q_\mathrm{I}, F, F)$ accepts the language $L(\mathcal{Z}) = \mathrm{cl}(L(\mathcal{X}) \cap \Sigma^*)$.

Consider an FSA $\mathcal{A}$. Using the two previous constructions, we build an FSA $\mathcal{A}'$ whose size is polynomial in $\|\mathcal{A}\|$, such that $L(\mathcal{A}') = \mathrm{cl}(\mathrm{pre}_*(L(\mathcal{A}) \cap L(\mathcal{U})) \cdot O^*) \cap L(\mathcal{U})$. Note that $\|\mathcal{U}\|$ is a constant, as $\mathcal{U}$ is fixed. By Lemma 6 $L(\mathcal{A})$ is $(L(\mathcal{U}), O)$-safety iff $L(\mathcal{A}') \subseteq L(\mathcal{A})$. Since the inclusion problem for FSAs is in PSPACE [16], our problem is therefore also in PSPACE. □

Corollary 12. *Let $\mathcal{U}$ be an FSA over the alphabet Σ such that $L(\mathcal{U})$ is a trace universe and let $O \subseteq \Sigma$. The decision problem of determining, for an FSA $\mathcal{A}$ over Σ, whether $L(\mathcal{A})$ is $(L(\mathcal{U}), O)$-enforceable, is PSPACE-complete.*

Proof. The proof is similar to that of Theorem 10, the statement being an easy consequence of Theorems 7 and 11. □

4.2 Logic-Based Specification Languages

Temporal logics are prominent specification languages for expressing properties on traces [28]. In the following, we consider a linear-time temporal logic with future and past operators, and metric constraints [2, 22].

We fix a finite set $\mathcal{P}$ of propositions, where we assume that they are classified into observable propositions $O \subseteq \mathcal{P}$ and controllable propositions $\mathcal{P} \setminus O$. The syntax of the metric linear-time temporal logic MLTL is given by the grammar

$$\varphi ::= \mathit{true} \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \bullet_I \varphi \mid \bigcirc_I \varphi \mid \varphi \,\mathsf{S}_I\, \varphi \mid \varphi \,\mathsf{U}_I\, \varphi\,,$$

where p ranges over the propositions in $\mathcal{P}$ and I ranges over the nonempty intervals over $\mathbb{N}$, i.e., subsets of the form $\{n, n+1, \ldots, m\}$ and $\{n, n+1, \ldots\}$ with $n, m \in \mathbb{N}$ and $n \leq m$. The *size* of a formula φ, denoted by $\|\varphi\|$, is the number of φ's subformulas plus the sum of the representation sizes of the interval bounds occurring in φ, which are $\lceil \log(1 + \max I) \rceil$ for a finite interval I, and $\lceil \log(1 + \min I) \rceil$ for an infinite interval I.

We use standard syntactic sugar. For instance, $\varphi \wedge \psi$ abbreviates $\neg(\neg\varphi \vee \neg\psi)$, $\Diamond_I \varphi$ abbreviates $\mathit{true}\,\mathsf{U}_I\,\varphi$, and $\Box_I \varphi$ abbreviates $\neg \Diamond_I (\neg\varphi)$. We drop the interval attached to a temporal operator if it is $\mathbb{N}$ and we use constraints like $\leq n$ and $\geq n$ to describe intervals of the form $\{0, 1, \ldots, n\}$ and $\{n, n+1, \ldots\}$, respectively. Furthermore, we use standard conventions concerning the binding strength of operators to omit parentheses. For instance, $\neg$ binds stronger than $\wedge$, which in turn binds stronger than $\vee$. Boolean operators bind stronger than temporal ones.

The truth value of a formula φ is defined over timestamped sequences, where time is monotonically increasing and progressing. To formalize this, we introduce the following notation. We denote the length of a sequence σ by $|\sigma|$ and the letter at the $(i+1)$st position in σ by σ_i, where $i \in \mathbb{N}$ with $i < |\sigma|$. We define T as the set that consists of the sequences $t \in \mathbb{N}^\infty$ with the following properties:

(i) For each $i, j \in \mathbb{N}$ with $i \leq j < |t|$, $t_i \leq t_j$.
(ii) If t is infinite then for each $k \in \mathbb{N}$, there is an integer $i \in \mathbb{N}$ with $t_i \geq k$.

Furthermore, for sequences $\sigma \in (2^{\mathcal{P}})^\infty$ and $t \in T$ with $|\sigma| = |t|$, we define $\sigma \otimes t$ as the sequence of length $|\sigma|$ with $(\sigma \otimes t)_i := (\sigma_i, t_i)$, for $i \in \mathbb{N}$ with $i < |\sigma|$. For $L \subseteq (2^{\mathcal{P}})^\infty$, we define $L \otimes T := \{\sigma \otimes t \mid \sigma \in L,\ t \in T, \text{ and } |\sigma| = |t|\}$.

For $\sigma \in (2^{\mathcal{P}})^\infty$, $t \in T$, and $i \in \mathbb{N}$ with $|\sigma| = |t|$ and $i < |\sigma|$, we define the relation $\models$ inductively over the formula structure:

$$
\begin{array}{rll}
\sigma, t, i \models \mathit{true} & & \\
\sigma, t, i \models p & \text{iff} & p \in \sigma_i \\
\sigma, t, i \models \neg\varphi & \text{iff} & \sigma, t, i \not\models \varphi \\
\sigma, t, i \models \varphi \vee \psi & \text{iff} & \sigma, t, i \models \varphi \text{ or } \sigma, t, i \models \psi \\
\sigma, t, i \models \bullet_I \varphi & \text{iff} & i > 0 \text{ and } t_i - t_{i-1} \in I \text{ and } \sigma, t, i-1 \models \varphi \\
\sigma, t, i \models \circ_I \varphi & \text{iff} & i < |\sigma| - 1 \text{ and } t_{i+1} - t_i \in I \text{ and } \sigma, t, i+1 \models \varphi \\
\sigma, t, i \models \varphi \,\mathsf{S}_I\, \psi & \text{iff} & \text{there is an integer } j \in \mathbb{N} \text{ with } j \leq i \text{ such that} \\
 & & \quad t_i - t_j \in I \text{ and } \sigma, t, j \models \psi \text{ and} \\
 & & \quad \sigma, t, k \models \varphi, \text{ for all } k \in \mathbb{N} \text{ with } j < k \leq i \\
\sigma, t, i \models \varphi \,\mathsf{U}_I\, \psi & \text{iff} & \text{there is an integer } j \in \mathbb{N} \text{ with } i \leq j < |\sigma| \text{ such that} \\
 & & \quad t_j - t_i \in I \text{ and } \sigma, t, j \models \psi \text{ and} \\
 & & \quad \sigma, t, k \models \varphi, \text{ for all } k \in \mathbb{N} \text{ with } i \leq k < j
\end{array}
$$

Finally, for a formula φ, we define $L(\varphi) := \{\varepsilon\} \cup \{\sigma \otimes t \in (2^{\mathcal{P}})^\infty \otimes T \mid \sigma, t, 0 \models \varphi\}$. We also define $L_\omega(\varphi)$ and $L_*(\varphi)$ that consist of the infinite and finite sequences in $L(\varphi)$, respectively. Note that different semantics exist for linear-time temporal logics over finite traces [10], each with their own artifacts. Since our semantics is not defined for the empty sequence, we include it in $L(\varphi)$.

The time model over which MLTL's semantics is defined is discrete and point-based. See Alur and Henzinger's survey [2] for an overview of alternative time models and their relationships. We briefly justify our chosen time model. The use of the discrete time domain $\mathbb{N}$ instead of a dense time domain like $\mathbb{Q}_{\geq 0}$ or even $\mathbb{R}_{\geq 0}$ is justified by the fact that clocks with arbitrarily fine precision do not exist in practice. The choice of a point-based time model is justified by our action-based view of system executions, where an action happens at some point in time. Furthermore, an enforcement mechanism does not continuously monitor the system but only at specific points in time.

Example 13. We return to the policies from Example 2. Let $\mathcal{P}$ be the proposition set $\{\mathit{fail}, \mathit{login}, \mathit{request}, \mathit{deliver}\}$. The formula

$$\varphi_1 := \Box\, \mathit{fail} \rightarrow \Box_{\leq 3}\, \neg \mathit{login}$$

formalizes the first policy and the second policy is formalized by the formula

$$\varphi_2 := \Box\, \mathit{request} \rightarrow \Diamond_{\leq 3}(\mathit{deliver} \vee \neg \circ\, \mathit{true})\,.$$

The trace properties described by φ_1 and φ_2 differ from the trace properties P_1 and P_2 from Example 2 in the following respects. First, the progression of time in P_1 and P_2 was explicitly modeled by *tick* actions. In $L(\varphi_1)$ and $L(\varphi_2)$ time is modeled by timestamping the letters in the sequences in $(2^{\mathcal{P}})^\infty$. We only consider timestamped sequences that adequately model time, i.e., the sequences in the trace universe $(2^{\mathcal{P}})^\infty \otimes T$, which is a subset of $(2^{\mathcal{P}} \times \mathbb{N})^\infty$. Second, the traces in Example 2 contained only one system action at a time. Here, we consider traces in which multiple system actions can happen at the same point in time. Instead

of using the trace universe $(2^{\mathcal{P}})^{\infty} \otimes T$, we can alternatively use the trace universe $\mathcal{P}^{\infty} \otimes T$ by filtering out the traces where a letter $(a,t) \in 2^{\mathcal{P}} \times \mathbb{N}$ occurs and a is not a singleton. However, the trace universe $\mathcal{P}^{\infty} \otimes T$ is more restrictive.

The trace properties described by φ_1 and φ_2 match the trace properties P_1 and P_2 from Example 2 with respect to enforceability. Here $O = \{\mathit{fail}\}$ and a letter $(a,t) \in 2^{\mathcal{P}} \times \mathbb{N}$ is only observable iff a does not contain any controllable actions, that is, iff $a = \emptyset$ or $a = \{\mathit{fail}\}$. To see, for instance, that $L(\varphi_2)$ is not enforceable, consider the trace $\sigma = (\{\mathit{request}\}, 0)$ and the letter $a = (\emptyset, 4)$. Then $\sigma \in L(\varphi_2)$ and $\sigma a \notin L(\varphi_2)$, while a is only observable.

In general, we assume that $a \in 2^{\mathcal{P}}$ is observable if $a \subseteq O$. In other words, $a \in 2^{\mathcal{P}}$ is controllable if it contains at least one controllable proposition. In particular, the empty set is not controllable. We define $\hat{O} := \{a \in 2^{\mathcal{P}} \mid a \subseteq O\}$.

In the remainder of this section, we analyze the complexity of two related realizability problems where policies are specified in MLTL. We start with the realizability problem for the untimed fragment of MLTL, which we call LTL. The interval attached to a temporal operator occurring in a formula of this fragment is $\mathbb{N}$. Hence, an LTL formula does not specify any timing constraints and, instead of $(2^{\mathcal{P}})^{\infty} \otimes T$, we consider trace universes that are subsets of $(2^{\mathcal{P}})^{\infty}$.

Lemma 14. *Let $O \subseteq \mathcal{P}$ and let $\mathcal{U}$ be an FSA such that $L(\mathcal{U}) \subseteq (2^{\mathcal{P}})^{\infty}$ is a trace universe. The decision problem of checking for an LTL formula φ whether $L(\varphi)$ is $(L(\mathcal{U}), \hat{O})$-enforceable is PSPACE-complete.*

Proof. By Theorem 7 we have that $L(\varphi)$ is $(L(\mathcal{U}), \hat{O})$-enforceable iff $L(\varphi)$ is $(L(\mathcal{U}), \hat{O})$-safety: note that $\varepsilon \in L(\varphi)$ by definition and $\mathrm{pre}_*(L(\varphi) \cap L(\mathcal{U}))$ is regular, hence decidable. Hence it suffices to show that determining whether $L(\varphi)$ is $(L(\mathcal{U}), \hat{O})$-safety is PSPACE-complete.

We first prove that the problem is PSPACE-hard. Recall that the satisfiability problem for LTL over infinite sequences is PSPACE-complete [30]. Given an LTL formula φ, we define $\varphi' := \varphi \vee \Diamond \neg \bigcirc \mathit{true}$. Then $L(\varphi') = L(\varphi) \cup (2^{\mathcal{P}})^*$. Moreover, using Lemma 6, we have that $L(\varphi')$ is $((2^{\mathcal{P}})^{\infty}, \emptyset)$-safety iff $L_{\omega}(\varphi) = (2^{\mathcal{P}})^{\omega}$ iff $L_{\omega}(\neg\varphi) = \emptyset$. Hence determining if $L(\varphi)$ is $((2^{\mathcal{P}})^{\infty}, \emptyset)$-safety is PSPACE-hard.

To show membership in PSPACE, let φ be an LTL formula of size $n \in \mathbb{N}$. There exist FSAs $\mathcal{A}$ and $\mathcal{A}'$ with $L(\mathcal{A}) = L(\varphi)$, $L(\mathcal{A}') = L(\neg\varphi)$, and $\|\mathcal{A}\|, \|\mathcal{A}'\| \in 2^{\mathcal{O}(n)}$. These two FSAs can be obtained by straightforwardly extending the translations of LTL over infinite sequences into nondeterministic Büchi automata [8, 31]. Using standard automata constructions and the constructions from the proof of Theorem 11, we build an FSA $\mathcal{B}$ with $\|\mathcal{B}\| \in 2^{\mathcal{O}(n)}$ and $L(\mathcal{B}) = L(\mathcal{A}') \cap L(\mathcal{U}) \cap \mathrm{cl}(\mathrm{pre}_*(L(\mathcal{A}) \cap L(\mathcal{U})) \cdot \hat{O}^*) \setminus \{\varepsilon\}$. It follows that $L(\varphi)$ is $(L(\mathcal{U}), \hat{O})$-safety iff $\mathrm{cl}(\mathrm{pre}_*(L(\varphi) \cap L(\mathcal{U})) \cdot \hat{O}^*) \cap L(\mathcal{U}) \subseteq L(\varphi)$ iff $L(\mathcal{B}) = \emptyset$. Since the emptiness problem for FSAs is in NLOGSPACE [21] and since we can construct $\mathcal{B}$ on the fly, our problem is in PSPACE. □

If $L(\varphi)$ is $(L(\mathcal{U}), \hat{O})$-enforceable, we can use the FSA $\mathcal{U}$ and the FSA $\mathcal{A}$ constructed in the proof of Lemma 14 to obtain an enforcement mechanism for $L(\varphi)$. Namely, we construct the product automaton $\mathcal{C}$ of $\mathcal{U}$ and $\mathcal{A}$ that accepts

the intersection of $L(\mathcal{U})$ and $L(\mathcal{A})$. The enforcement mechanism E initially stores the singleton set consisting of $\mathcal{C}$'s initial state. Whenever E intercepts a system action $a \in 2^{\mathcal{P}}$, it updates this set by determining the successor states of the stored states using $\mathcal{C}$'s transition function. We remove from the updated set the states from which we do not accept any sequence. E terminates the system if the set becomes empty provided that the intercepted action a is controllable. Otherwise, it continues by intercepting the next system action.

Theorem 15. *Let $O \subseteq \mathcal{P}$ and let $\mathcal{U}$ be an FSA such that $L(\mathcal{U}) \subseteq (2^{\mathcal{P}})^\infty$ is a trace universe. The decision problem of checking for an MLTL formula φ whether $L(\varphi)$ is $(L(\mathcal{U}) \otimes T, \hat{O} \times \mathbb{N})$-enforceable is EXPSPACE-complete.*

Proof. Let $tick \notin \mathcal{P}$ be a new proposition modeling clock ticks. Let $\Sigma := 2^{\mathcal{P}}$, $\overline{\Sigma} := 2^{\mathcal{P} \cup \{tick\}}$, $U_T := L(\mathcal{U}) \otimes T$, and $\mathcal{T} := \Sigma^\infty \otimes T$. We first map each MLTL formula φ to an LTL formula $\overline{\varphi}$, each FSA $\mathcal{A}$ to an FSA $\overline{\mathcal{A}}$, and each trace τ in $\mathcal{T}$ to a trace $\overline{\tau}$ in $\overline{\Sigma}^\omega$ such that

- $\tau \in L(\varphi)$ iff $\overline{\tau} \in L(\overline{\varphi})$ and
- $\tau \in L(\mathcal{A})$ iff $\overline{\tau} \in L(\overline{\mathcal{A}})$.

For a trace $\tau = \sigma \otimes t$ in $\mathcal{T}$, we define the trace $\overline{\tau}$ in $\overline{\Sigma}^\infty$ as follows:

- if τ is infinite, then $\overline{\tau} := \{tick\}^{t_0}\sigma_0\{tick\}^{d_1}\sigma_1\{tick\}^{d_2}\sigma_2 \ldots$,
- if $\tau = \varepsilon$, then $\overline{\tau} := \{tick\}^\omega$, and
- if $\tau \neq \varepsilon$ is finite, then $\overline{\tau} := \{tick\}^{t_0}\sigma_0\{tick\}^{d_1}\sigma_1\{tick\}^{d_2}\sigma_2 \ldots \sigma_{|\tau|-1}\{tick\}^\omega$,

where $d_i := t_i - t_{i-1}$, $\{tick\}^i$ is the sequence $\{tick\} \ldots \{tick\}$ of length i and $\{tick\}^\omega$ is the infinite sequence $\{tick\}\{tick\} \ldots$. For a set of traces $L \subseteq \mathcal{T}$, we abbreviate by $\overline{L}$ the set $\{\overline{\tau} \in \overline{\Sigma}^\infty \mid \tau \in L\}$. Note that this mapping is one-to-one, so that it induces a bijection from L to $\overline{L}$.

For an MLTL formula φ, we define the formulas $\ulcorner\varphi\urcorner$ and $\overline{\varphi}$ as follows:

- $\ulcorner true \urcorner := true$,
- $\ulcorner p \urcorner := p$ if $p \in \mathcal{P}$,
- $\ulcorner \neg\varphi \urcorner := \neg \ulcorner\varphi\urcorner$,
- $\ulcorner \varphi \vee \psi \urcorner := \ulcorner\varphi\urcorner \vee \ulcorner\psi\urcorner$,
- $\ulcorner \bigcirc_I \varphi \urcorner := \ulcorner \bigcirc_I true \urcorner \wedge \ulcorner \bigcirc \varphi \urcorner$ if $I \neq \mathbb{N}$ and $\varphi \neq true$,
- $\ulcorner \bigcirc_I true \urcorner := \bigcirc(tick \wedge \ulcorner \bigcirc_{I-1} true \urcorner)$ if $0 \notin I$, where $I - 1 := \{t - 1 \mid t \in I\}$,
- $\ulcorner \bigcirc_{[0,a]} true \urcorner := \bigcirc(\neg tick \vee \ulcorner \bigcirc_{[0,a-1]} true \urcorner)$ if $a \geq 1$,
- $\ulcorner \bigcirc_{[0,0]} true \urcorner := \bigcirc \neg tick$,
- $\ulcorner \bigcirc \varphi \urcorner := \bigcirc(tick \,\mathsf{U}\, (\neg tick \wedge \ulcorner\varphi\urcorner))$,
- $\ulcorner \varphi \,\mathsf{U}_I\, \psi \urcorner := (\neg tick \wedge \ulcorner\varphi\urcorner) \,\mathsf{U}\, (tick \wedge \bigcirc(\ulcorner \varphi \,\mathsf{U}_{I-1}\, \psi \urcorner))$ if $0 \notin I$,
- $\ulcorner \varphi \,\mathsf{U}_{[0,a]}\, \psi \urcorner := (\neg tick \wedge \ulcorner\varphi\urcorner) \,\mathsf{U}\, ((\neg tick \wedge \ulcorner\psi\urcorner) \vee (tick \wedge \bigcirc(\ulcorner \varphi \,\mathsf{U}_{[0,a-1]}\, \psi \urcorner)))$ if $a \geq 1$,
- $\ulcorner \varphi \,\mathsf{U}_{[0,0]}\, \psi \urcorner := (\neg tick \wedge \ulcorner\varphi\urcorner) \,\mathsf{U}\, (\neg tick \wedge \ulcorner\psi\urcorner)$,
- $\ulcorner \varphi \,\mathsf{U}\, \psi \urcorner := (tick \vee \ulcorner\varphi\urcorner) \,\mathsf{U}\, (\neg tick \wedge \ulcorner\psi\urcorner)$,
- $\ulcorner \bullet_I \varphi \urcorner$ and $\ulcorner \varphi \,\mathsf{S}_I\, \psi \urcorner$ are defined analogously to $\ulcorner \bigcirc_I \varphi \urcorner$ and $\ulcorner \varphi \,\mathsf{U}_I\, \psi \urcorner$,
- $\overline{\varphi} := (\Box\, tick) \vee (tick \,\mathsf{U}\, (\neg tick \wedge \ulcorner\varphi\urcorner))$.

For an FSA $\mathcal{A} = (Q, \Sigma, \delta, q_{\mathrm{I}}, F, B)$, we define the FSA $\overline{\mathcal{A}} := (\overline{Q}, \overline{\Sigma}, \overline{\delta}, \overline{q}_{\mathrm{I}}, \overline{F}, \overline{B})$ with $\overline{Q} := Q \times \{0, 1, 2\}$, $\overline{q}_{\mathrm{I}} := (q_{\mathrm{I}}, 0)$, $\overline{F} := \emptyset$, $\overline{B} := (B \times \{0\}) \cup (F \times \{2\})$, and for any $q \in Q$, $i \in \{0, 1, 2\}$, and $a \in \overline{\Sigma}$,

$$\overline{\delta}((q,i),a) := \begin{cases} \{(q',0) \mid q' \in \delta(q,a)\} & \text{if } a \in \Sigma \text{ and } i \in \{0,1\}, \\ \{(q,1),(q,2)\} & \text{if } a = \{tick\} \text{ and } i = 0, \\ \{(q,i)\} & \text{if } a = \{tick\} \text{ and } i \in \{1,2\}, \\ \emptyset & \text{otherwise.} \end{cases}$$

It is easy to check that $\tau \in L(\mathcal{A}) \otimes T$ iff $\overline{\tau} \in L(\overline{\mathcal{A}}) \cap \overline{\mathcal{T}}$. In addition, by induction over φ, one verifies that $\sigma, t, i \models \varphi$ iff $\overline{\tau}, i+t_i \models \ulcorner\varphi\urcorner$ for all $i < |\tau|$, where $\tau = \sigma \otimes t$. Therefore, $\tau \in L(\varphi)$ iff $\overline{\tau} \in L(\overline{\varphi})$.

Note that $\overline{\mathcal{T}} = L(\theta) \cap \overline{\Sigma}^\omega$, where $\theta := (\Box \Diamond tick) \wedge \Box(tick \rightarrow \bigwedge_{p \in \mathcal{P}} \neg p)$. Then $\overline{U_T} = L(\overline{\mathcal{U}}) \cap \overline{\mathcal{T}}$. Moreover, $U_T \cap (\Sigma \times \mathbb{N})^*$ is decidable, and a finite trace τ in U_T is in $\mathrm{pre}_*(L(\varphi))$ iff $\overline{\tau}^{<|\tau|+t_{|\tau|-1}}$ is in $\mathrm{pre}_*(L(\overline{\varphi}) \cap \overline{\mathcal{T}})$. Since $\mathrm{pre}_*(L(\overline{\varphi}) \cap \overline{\mathcal{T}})$ is decidable, so is $\mathrm{pre}_*(L(\varphi) \cap U_T)$. Thus $L(\varphi)$ is $(U_T, \hat{O} \times \mathbb{N})$-enforceable iff $L(\varphi)$ is $(U_T, \hat{O} \times \mathbb{N})$-safety.

Recall now that the satisfiability problem for MLTL with infinite timed words is EXPSPACE-hard [3]. Given an MLTL formula φ, we define the formula $\varphi' := \varphi \vee \Diamond \neg \bigcirc true$. We have $L(\varphi') = L(\varphi) \cup (\mathcal{T} \cap (\Sigma \times \mathbb{N})^*)$. $L(\varphi')$ is $(U_T, \hat{O} \times \mathbb{N})$-safety iff $L_\omega(\varphi) = \mathcal{T} \cap (\Sigma \times \mathbb{N})^\omega$ iff $L_\omega(\neg\varphi) = \emptyset$. This proves that checking whether $L(\varphi)$ is $(U_T, \hat{O} \times \mathbb{N})$-safety is EXPSPACE-hard.

To prove membership in EXPSPACE, consider an MLTL formula φ of size $n \in \mathbb{N}$. It is easy to see by induction over φ that $\|\overline{\varphi}\| \in 2^{\mathcal{O}(n)}$. Moreover, note that $\overline{\mathcal{T} \cap (\Sigma \times \mathbb{N})^\omega} = L(\theta') \cap \overline{\Sigma}^\omega$, where $\theta' := \theta \wedge (\Box \Diamond \neg tick)$. For convenience, we also let $O_t := O \cup \{tick\}$ and $S_\varphi := \mathrm{cl}(\mathrm{pre}_*(L(\varphi) \cap U_T) \cdot (\hat{O} \times \mathbb{N})^*) \cap U_T$. We have that S_φ is mapped to $\overline{S_\varphi} = \big(\mathrm{pre}_*(L(\overline{\varphi}) \cap \overline{U_T}) \cdot \hat{O}_t^{\ \omega} \cup \big(\mathrm{cl}(\mathrm{pre}_*(L(\overline{\varphi}) \cap \overline{U_T})) \cap L(\theta')\big)\big) \cap \overline{U_T}$. Therefore, $L(\varphi)$ is $(U_T, \hat{O} \times \mathbb{N})$-enforceable iff $\overline{S_\varphi} \subseteq L(\overline{\varphi})$.

As in the proof of Theorem 11, we build an FSA $\mathcal{B}$ of size $2^{2^{\mathcal{O}(n)}}$ such that $L(\mathcal{B}) = \overline{S_\varphi} \cap L(\neg\overline{\varphi})$. Then $L(\varphi)$ is $(U_T, \hat{O} \times \mathbb{N})$-enforceable iff $L(\mathcal{B}) = \emptyset$. As the emptiness problem for FSAs is in NLOGSPACE [21] and since we can build $\mathcal{B}$ on the fly, checking whether $L(\varphi)$ is $(U_T, \hat{O} \times \mathbb{N})$-safety is in EXPSPACE. □

If $L(\varphi)$ is $(L(\mathcal{U}) \otimes T, \hat{O} \times \mathbb{N})$-enforceable, we can use—similar to the LTL case—the FSAs $\overline{\mathcal{U}}$ and $\overline{\mathcal{A}}$ from the proof of Theorem 15 to obtain an enforcement mechanism E. We construct the product automaton $\mathcal{C}$ accepting $L(\overline{\mathcal{U}}) \cap L(\overline{\mathcal{A}})$. The enforcement mechanism E initializes the state set to the singleton set consisting of $\mathcal{C}$'s initial state. Additionally, E stores the current timestamp, which is initially 0. Whenever E intercepts a system action $(a, t) \in 2^{\mathcal{P}} \times \mathbb{N}$, it performs the following updates on the state set and the current timestamp.

1. E updates the state set with respect the progression of time, i.e., E determines the states reachable by the sequence $tick^d$, where d is the difference of the timestamp t and the stored timestamp.
2. E stores t as the current timestamp.
3. E updates the state set with respect to the system action a.
4. E removes the states from the state set from which $\mathcal{C}$ does not accept any sequence.

E terminates the system if the state set becomes empty and the intercepted action a is controllable. Otherwise, it continues by intercepting the next action.

5 Related Work

Schneider [29] initiated the study of which security policies are enforceable. He showed that every security policy enforceable by execution monitoring must be a property of traces and an ∞-safety property. Furthermore, he introduced an automata model, called security automata, that recognizes ∞-safety properties. Fong [15] analyzed classes of security policies that can be recognized by shallow-history automata, a restricted class of security automata. Hamlen et. al [18] related the policies that can be enforced by program rewriting to those that can be recognized by security automata. Ligatti et al. [24, 25] introduced edit automata, which are transducers with infinitely many states. Edit automata can recognize trace properties that are not ∞-safety. However, it remains unclear how to use edit automata as enforcement mechanisms, in particular, how an edit automaton and a system interact with each other in general. Ligatti and Reddy [26] recently introduced mandatory-result automata for enforcement and analyzed their expressive power. In contrast to edit automata, mandatory-result automata have an interface for interacting with a system. Namely, a mandatory-result automaton obtains requests from the system and sends outputs back to the system. Before sending output, it can interact with the execution platform. Falcone et al. [14] study the trace properties that can be recognized by security, edit, and shallow-history automata in terms of the safety-progress hierarchy [7] of regular languages and classical finite-state automata models.

All the above works assume that all system actions are controllable. In contrast, we distinguish between actions that are controllable and those that are only observable by an enforcement mechanism. Furthermore, the above works also do not consider the realizability of an enforcement mechanism from a policy description and its computational complexity. Note that classifications of system actions, signals, and states with a flavor similar to ours are common in other areas like control theory and software testing. However, to the best of our knowledge, this is the first such investigation in the domain of policy enforcement.

Recently, the problem of checking whether system behaviors are compliant with security policies, regulations, and laws has attracted considerable attention. This problem is simpler than policy enforcement, since one need only detect and report policy violations. Monitoring approaches have proved useful here, based either on offline [17] or online [4] algorithms. See also [5].

Another generalization of the standard definition of safety [1] has been recently given by Ehlers and Finkbeiner [9]. They distinguish between the inputs and outputs of a reactive system. The corresponding decision problems are EXPTIME-complete and 2EXPTIME-complete when the properties are given as automata and LTL formulas, respectively. Since enforcement mechanisms based on execution monitoring do not produce outputs, their generalization does not apply to our setting. However, a combination of their safety generalization and ours seems promising when considering more powerful enforcement mechanisms like those based on mandatory-result automata [26].

6 Conclusion

We have refined Schneider's setting for policy enforcement based on execution monitoring by distinguishing between controllable and observable system actions. This allows us to reason about enforceability in systems where not all actions can be controlled, for example, the passage of time. Using our characterization, we have provided, for the first time, both necessary and sufficient conditions for enforceability. We have also examined the problem of determining whether a specified policy is enforceable, for different specification languages, and provided results on the complexity of this realizability decision problem.

As future work, we will investigate the realizability problem for more powerful enforcement mechanisms and for more expressive specification languages, such as those not limited to finite alphabets. We would also like to provide tool support for synthesizing enforcement mechanisms from declarative policy specifications.

References

1. Alpern, B., Schneider, F.B.: Defining liveness. Inform. Process. Lett. 21(4), 181–185 (1985)
2. Alur, R., Henzinger, T.A.: Logics and Models of Real Time: A Survey. In: Huizing, C., de Bakker, J.W., Rozenberg, G., de Roever, W.-P. (eds.) REX 1991. LNCS, vol. 600, pp. 74–106. Springer, Heidelberg (1992)
3. Alur, R., Henzinger, T.A.: A really temporal logic. J. ACM 41(1), 181–203 (1994)
4. Basin, D., Harvan, M., Klaedtke, F., Zălinescu, E.: Monitoring usage-control policies in distributed systems. In: Proceedings of the 18th International Symposium on Temporal Representation and Reasoning, pp. 88–95. IEEE Computer Society (2011)
5. Basin, D., Klaedtke, F., Müller, S.: Monitoring security policies with metric first-order temporal logic. In: Proceedings of the 15th ACM Symposium on Access Control Models and Technologies, pp. 23–33. ACM Press (2010)
6. Basin, D., Olderog, E.-R., Sevinç, P.E.: Specifying and analyzing security automata using CSP-OZ. In: Proceedings of the 2007 ACM Symposium on Information, Computer and Communications Security, pp. 70–81. ACM Press (2007)
7. Chang, E.Y., Manna, Z., Pnueli, A.: Characterization of Temporal Property Classes. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 474–486. Springer, Heidelberg (1992)
8. Dax, C., Klaedtke, F., Lange, M.: On regular temporal logics with past. Acta Inform. 47(4), 251–277 (2010)
9. Ehlers, R., Finkbeiner, B.: Reactive safety. In: Proceedings of 2nd International Symposium on Games, Logics and Formal Verification. Electronic Proceedings in Theoretical Computer Science, vol. 54, pp. 178–191 (2011), `eptcs.org`
10. Eisner, C., Fisman, D., Havlicek, J., Lustig, Y., McIsaac, A., Van Campenhout, D.: Reasoning with Temporal Logic on Truncated Paths. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 27–39. Springer, Heidelberg (2003)
11. Erlingsson, Ú.: The inlined reference monitor approach to security policy enforcement. PhD thesis, Cornell University, Ithaca, NY, USA (2004)
12. Erlingsson, Ú., Schneider, F.B.: SASI enforcement of security policies: A retrospective. In: Proceedings of the 1999 Workshop on New Security Paradigms, pp. 87–95. ACM Press (1999)

13. Erlingsson, Ú., Schneider, F.B.: IRM enforcement of Java stack inspection. In: Proceedings of the 2000 IEEE Symposium on Security and Privacy, pp. 246–255. IEEE Computer Society (2000)
14. Falcone, Y., Mounier, L., Fernandez, J.-C., Richier, J.-L.: Runtime enforcement monitors: composition, synthesis, and enforcement abilities. Form. Methods Syst. Des. 38(2), 223–262 (2011)
15. Fong, P.W.: Access control by tracking shallow execution history. In: Proceedings of the 2004 IEEE Symposium on Security and Privacy, pp. 43–55. IEEE Computer Society (2004)
16. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company (1979)
17. Garg, D., Jia, L., Datta, A.: Policy auditing over incomplete logs: Theory, implementation and applications. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, pp. 151–162. ACM Press (2011)
18. Hamlen, K.W., Morrisett, G., Schneider, F.B.: Computability classes for enforcement mechanisms. ACM Trans. Progr. Lang. Syst. 28(1), 175–205 (2006)
19. Henzinger, T.A.: Sooner is safer than later. Inform. Process. Lett. 43(3), 135–141 (1992)
20. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)
21. Jones, N.D.: Space-bounded reducibility among combinatorial problems. J. Comput. Syst. Sci. 11(1), 68–85 (1975)
22. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Syst. 2(4), 255–299 (1990)
23. Lamport, L.: Proving the correctness of multiprocess programs. IEEE Trans. Software Eng. 3(2), 125–143 (1977)
24. Ligatti, J., Bauer, L., Walker, D.: Edit automata: enforcement mechanisms for run-time security policies. Int. J. Inf. Secur. 4(1-2), 2–16 (2005)
25. Ligatti, J., Bauer, L., Walker, D.: Run-time enforcement of nonsafety policies. ACM Trans. Inform. Syst. Secur. 12(3) (2009)
26. Ligatti, J., Reddy, S.: A Theory of Runtime Enforcement, with Results. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 87–100. Springer, Heidelberg (2010)
27. Paul, M., Siegert, H.J., Alford, M.W., Ansart, J.P., Hommel, G., Lamport, L., Liskov, B., Mullery, G.P., Schneider, F.B.: Distributed Systems—Methods and Tools for Specification: An Advanced Course. LNCS, vol. 190. Springer, Heidelberg (1985)
28. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science, pp. 46–57. IEEE Computer Society (1977)
29. Schneider, F.B.: Enforceable security policies. ACM Trans. Inform. Syst. Secur. 3(1), 30–50 (2000)
30. Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logic. J. ACM 32(3), 733–749 (1985)
31. Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. Inf. Comput. 115(1), 1–37 (1994)
32. Viswanathan, M.: Foundations for the run-time analysis of software systems. PhD thesis, University of Pennsylvania, Philadelphia, PA, USA (2000)