Enforceable Security Policies Revisited

DAVID BASIN, ETH Zurich
VINCENT JUGÉ, MINES ParisTech
FELIX KLAEDTKE and EUGEN ZĂLINESCU, ETH Zurich

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 sees them but 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.

Categories and Subject Descriptors: D.2.0 [**Software Engineering**]: General—*Protection mechanisms*; D.2.4 [**Software Engineering**]: Software/Program Verification—*Validation; formal methods*; D.2.5 [**Software Engineering**]: Testing and Debugging—*Monitors*; F.1.2 [**Computation of Abstract Devices**]: Modes of Computation—*Interactive and reactive computation*; F.3.1 [**Logics and Meaning of Programs**]: Specifying and Verifying and Reasoning about Programs—*Specification techniques*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Temporal logic*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Reliability, Security, Verification

Additional Key Words and Phrases: Automata, monitoring, safety properties, security policies, temporal logic

ACM Reference Format:
Basin, D., Jugé, V., Klaedtke, F., and Zălinescu, E. 2013. Enforceable security policies revisited. *ACM Trans. Inf. Syst. Secur.* 16, 1, Article 3 (June 2013), 26 pages.
DOI:http://dx.doi.org/10.1145/2487222.2487225

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 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 [2000] began the investigation of which kinds of security

This article is an extended version of the conference paper Basin et al. [2012a].
This work was partly supported by Google Inc.
Authors' addresses: D. Basin, Institute of Information Security, ETH Zurich, Switzerland; V. Jugé, MINES ParisTech, France; F. Klaedtke (corresponding author) and E. Zălinescu, Institute of Information Security, ETH Zurich, Switzerland; email: felixkl@inf.ethz.ch.

DOI:http://dx.doi.org/10.1145/2487222.2487225

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. Whenever an action would result in a policy violation, the enforcement mechanism terminates the system. Note that there are alternative notions of enforceability where, rather than terminating execution just prior to a violation, one may raise exceptions, take corrective actions, and the like. For example, Ligatti and others [2005, 2009] consider enforcement mechanisms that can also insert actions into and delete actions from noncompliant behavior. Moreover, their enforcement mechanisms may even change compliant behavior, provided that the resulting behavior is semantically equivalent. We follow Schneider's account in this article; see also Section 5 for more discussion on this point.

In this article, we refine Schneider's setting, thereby overcoming several limitations. To explain the limitations, we first summarize Schneider's findings. Schneider [2000] shows that only those security policies that can be described by a safety property [Alpern and Schneider 1985; Lamport 1977; Paul et al. 1985] on traces are enforceable by execution monitoring. In other words, a policy is enforceable by execution monitoring only when: (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 policy-compliant 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 "can serve as the basis for an enforcement mechanism" [Schneider 2000, 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 [2000, page 41].

We provide a formalization of enforceability for mechanisms similar to Schneider's [2000], that is, monitors that observe system actions and terminate systems to prevent policy violations. 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 target 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, for example, that availability policies with 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 often only 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 [Alpern and Schneider 1985] 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 first present decidability results for the decision problem that asks whether a given security policy is enforceable. Furthermore, in case of decidability, we show how to synthesize an enforcement mechanism for

[1]Note that a trace property must also be a decidable set to be enforceable, as remarked later by Viswanathan [2000] and Hamlen et al. [2006]. Furthermore, it must be nonempty [Ligatti et al. 2009].

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. The underlying decision problem is EXPSPACE-complete and is PSPACE-complete without metric constraints.

Summarizing, we see our contributions as follows. We overcome limitations of Schneider's setting on policy enforcement based on execution monitoring [Schneider 2000]. First, we distinguish between controllable and observable system actions when monitoring executions. Second, we give conditions for policy enforcement based 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. Finally, we 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. Finally, we present examples illustrating our notion.

2.1. Policy Enforcement Based on Execution Monitoring

We take an abstract view of systems and their behaviors similar to Schneider [2000] and others [Ligatti and Reddy 2010; Ligatti et al. 2005, 2009], 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, then 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, then E can detect the violation but not prevent it. In such a case, E is not an enforcement mechanism for the policy P. Overall, in this interaction between a system S and an enforcement mechanism E, we extend Schneider's setting [Schneider 2000] by distinguishing between controllable and observable actions.

We make several remarks about this system architecture. 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, Basin et al. [2007], where it is assumed that all actions are controllable. The composed system terminates in a deadlock in case of a 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 actions are solely generated by the target 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 [2000], a necessary condition for enforcing a policy by execution monitoring is that policy compliance is determined only by the observed trace. We therefore require that a policy P is a *property of traces*, that is, $P \subseteq \Sigma^* \cup \Sigma^\omega$, 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 [Schneider 2000]. 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.

Finally, although implicit in Schneider's work [Schneider 2000], there are also *soundness* and *transparency* requirements for an enforcement mechanism [Erlingsson 2004; Hamlen et al. 2006; Ligatti et al. 2005, 2009]. 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 earlier 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 compliance is the intercepted action a together with the already executed trace τ. Since these checks should be carried out algorithmically, our formalization of enforceability requires the existence of a Turing machine. In particular, for every check, this 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 results in a policy violation. As a consequence, if a is observable, the Turing machine must accept τa. We do not restrict the Turing machine's time and space complexity since we are interested in the class of policies that are in principle enforceable, not necessarily efficiently enforceable. We also 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 sequences $\sigma, \sigma' \in \Sigma^\infty$, we say that σ is a *prefix* of σ' if there is a sequence $\tau \in \Sigma^\infty$ such that $\sigma' = \sigma\tau$. Note that a sequence is always a prefix of itself. 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 $\mathrm{pre}_*(\sigma)$, that is, $\mathrm{pre}_*(\sigma) := \mathrm{pre}(\sigma) \cap \Sigma^*$. 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 $L \cdot K := \{\sigma\tau \in \Sigma^\infty \mid \sigma \in L \text{ and } \tau \in K\}$.

Since not all sequences in Σ^∞ might correspond to reasonable system executions, we formalize enforceability relative to a *trace universe* U. A trace universe describes all possible system executions or overapproximates these, that is, U is a nonempty prefix-closed subset of Σ^∞. For example, when considering real-time systems [Alur and Henzinger 1992] and policies with timing constraints, we can discard sequences from Σ^∞ where time does not progress; see Example 2.2 and Section 4.2. Trace universes have been used for similar purposes in Henzinger [1992] and Clarkson and Schneider [2010].

Definition 2.1. 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[2] $\mathcal{M}$ 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 $(\text{trunc}(A) \cdot \Sigma) \cap U$.
(ii) $\mathcal{M}$ accepts the inputs in $(\text{trunc}(A) \cdot O) \cap U$.
(iii) $\text{cl}(\text{trunc}(A)) \cap U = P \cap U$.
(iv) $\varepsilon \in A$.

Intuitively, property (i) ensures 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 $\text{trunc}(A)$ 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 $A \supseteq (\text{trunc}(A) \cdot O) \cap U$ 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 $\text{cl}(\text{trunc}(A)) \cap U \subseteq P \cap U$ formalizes the soundness requirement for an enforcement mechanism and $\text{cl}(\text{trunc}(A)) \cap U \supseteq P \cap U$ formalizes the transparency requirement. Since P can contain infinite sequences, we need to consider the limit closure of $\text{trunc}(A)$, that is, $\text{cl}(\text{trunc}(A))$. With property (iv) we ensure that the system S is initially policy compliant.

Natural questions that arise from Definition 2.1 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 an enforcement mechanism from its description? Before investigating these questions, we illustrate Definition 2.1 by considering several examples.

2.3. Examples

We now present examples of both enforceable and nonenforceable policies.

Example 2.2. Consider the following two policies describing actions that must or must not happen within a fixed time interval. 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 shortly. We assume, for the ease of exposition, that actions

[2]Since Σ can be infinite, we require that $\mathcal{M}$'s transition function is computable. This is without loss of generality since we assume that Σ is finitely representable, that is, we can represent the actions in Σ by finite words over some finite alphabet Σ'. Instead of $\mathcal{M}$ we could use a deterministic Turing machine with the alphabet Σ'.

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 $\{\mathit{tick}, \mathit{fail}, \mathit{login}, \mathit{request}, \mathit{deliver}\}$. The trace universe $U \subseteq \Sigma^\infty$ consists of all 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

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

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

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

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 failures 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 2.1, 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 *tick* and *fail* actions 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 $\mathcal{M}$ as required in Definition 2.1, which must accept the trace $\mathit{request}\,\mathit{tick}^3 \in P_2$. But then, by condition (ii) of Definition 2.1, it also must accept the trace $\mathit{request}\,\mathit{tick}^4 \notin P_2$. Note that terminating the system before observing the fourth *tick* action would violate the transparency requirement.

Example 2.3. A common security principle is to limit users' permissions to prevent system misuse and fraud. An example is the following policy, which is a simplified variant of dynamic separation of duty in the RBAC standard [American National Standards Institute, Inc. 2004]: a user may be a member of any two exclusive roles as long as the user has not activated both of them in the same session. This policy contains two kinds of actions which are usually attributed to different agents. Namely, a user *activates* the roles in his own sessions and a system administrator *changes* the exclusiveness relation of roles. A slightly weaker policy requires that a user may only activate a role in a session if he is currently a member of that role and the role is not exclusive to any other currently active role in the session.

The first policy is only enforceable if both kinds of actions are controllable. In particular, an enforcement mechanism must prevent an administrative action that makes two roles exclusive whenever these roles are both currently activated in a session of some user. In contrast, the weaker policy is enforceable even when changing the exclusiveness relation of roles is only an observable action. Here, the enforcement mechanism must just prevent the activation of a role if the user is not a member of that role or the role is in the exclusiveness relation with some other active role in the user's session at the moment of its activation. For this, it suffices to observe the administrative actions and to update the exclusiveness relation accordingly. With respect to Definition 2.1, it is easy to provide a Turing machine that accepts a trace extended with an intercepted role activation action iff the role's activation does not lead to a policy violation.

We remark that the RBAC standard requires that such administrative actions are controllable so that the stronger variant of so-called dynamic separation-of-duty constraints is indeed enforceable. However, this comes at the cost that an enforcement mechanism must check the constraints for all current sessions before, for example, roles are made exclusive.

Note that terminating a target system is a rather harsh countermeasure to prevent policy violations. As the preceding example illustrates, it is sometimes preferable for an enforcement mechanism simply to deny the execution of the intercepted action and to notify the system about this. The target system may then try to continue with another action. As already remarked by Schneider [2000], extending the capabilities of an enforcement mechanism so that it can also notify the monitored system whether an action was permitted or denied does not enlarge the class of security policies that are enforceable.

Example 2.4. Our third example concerns distributed systems, where the enforcement mechanism controls only actions that originate from one of the distributed processes. Suppose that a user's mail client is monitored by an enforcement mechanism that controls actions like sending and encrypting the user's emails. However, a send action that originates from another mail client is not controllable. Indeed, it is only observable when the email is received by the user's mail client. A security policy stating that the user's mail correspondence must be encrypted is not enforceable, since the enforcement mechanism cannot prevent other mail clients from sending him unencrypted emails. However, policy violations can be detected. Obviously, the weaker policy stating that the user must not send unencrypted emails is enforceable.

As this third example suggests, policy enforcement is generally difficult to achieve in distributed environments where components' actions are out of the enforcement mechanism's control. Instead one must often settle for something weaker, namely observing policy violations. This is the challenge of *distributed usage control* [Pretschner et al. 2006].

3. RELATION BETWEEN ENFORCEABILITY AND SAFETY

In this section, we characterize the class of trace properties that are enforceable with respect to Definition 2.1. To provide this characterization, we first generalize the standard notions of safety properties [Alpern and Schneider 1985; Henzinger 1992].

3.1. Generalizing Safety

According to Lamport [1977], a safety property intuitively states that nothing bad ever happens. A widely accepted formalization of this, from Alpern and Schneider [1985], 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 3.1 that follows, 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 definitions of safety and liveness [Henzinger 1992], which extends the classical safety-liveness classification for properties of untimed systems [Alpern and Schneider 1985] to real-time settings. Furthermore, our definition is parametric in a 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. In other words, elements in O cannot be the source of a violation.

Definition 3.1. Let $U \subseteq \Sigma^\infty$ and $O \subseteq \Sigma$. The set $P \subseteq \Sigma^\infty$ is (U, O)-*safety* if

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

Remark 3.2. For the sake of completeness, we remark that Alpern and Schneider's [1985] widely accepted formalization of liveness, which captures the intuition that something good can happen [Lamport 1977], also has a natural generalization to the setting with a universe $U \subseteq \Sigma^\infty$ and a set of actions $O \subseteq \Sigma$. The universe U relativizes liveness to the sequences in U [Henzinger 1992]. The set of actions O further relativizes liveness such that good things must only be possible after actions not in O, instead of requiring that good things can always happen. We say that $P \subseteq \Sigma^\infty$ is (U, O)-*liveness* if

$$\forall \sigma \in U. \forall \sigma' \in \Sigma^*. \sigma' \in \mathrm{pre}(\sigma) \wedge \sigma' \notin \Sigma^* \cdot O \rightarrow \exists \tau \in \Sigma^\infty. \sigma'\tau \in P \cap U\,.$$

Again, as with our generalized notion of safety, for $U = \Sigma^\omega$ and $O = \emptyset$, this formalization matches the one by Alpern and Schneider [1985]. The intuition for P being (U, O)-liveness is that something good can happen in U after actions not in O. An example is that after a send action it must be possible to eventually perform a receive action. However, if a send action takes place while the receiver process is terminated (this would be the action in O), we do not impose that the corresponding receive action takes place.

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

Example 3.3. Both the policies P_1 and P_2 from Example 2.2 are ∞-safety. If a trace τ violates P_1 then the violation can be pinpointed to a position where a *login* action is executed. That is, 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, policy violations with respect to P_2 are caused by *tick* actions rather than *login* actions.

However, P_1 is (U, O)-safety while P_2 is not (U, O)-safety, where U and O are as in Example 2.2. A violation of P_1 is caused by executing a *login* action, which is controllable. 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*, which is only observable. 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 3.4. Recall the trace universe $U \subseteq \Sigma^\infty$ from Example 2.2, 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.

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

When considering only the infinite traces, the trace property $P \cap \Sigma^\omega$ is not ω-safety. In fact, according to Alpern and Schneider [1985], $P \cap \Sigma^\omega$ 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 $P \cap \Sigma^\omega = U \cap \Sigma^\omega$.

Although safety and liveness focus on complementary aspects of system behavior, Example 3.4 illustrates that, besides trivial corner cases, a liveness property can also be a safety property in an appropriate universe. Intuitively, for larger universes U and larger sets O, it is less likely that a set P of traces is (U, O)-safety. This intuition is captured by the following lemma showing that the safety of a trace property is preserved with respect to subset inclusion of universes U and sets of actions O.

LEMMA 3.5. *For every $P, U \subseteq \Sigma^\infty$ and $O \subseteq \Sigma$, if P is (U, O)-safety then P is (U', O')-safety, for every $U' \subseteq U$ and $O' \subseteq O$.*

PROOF. Let σ be a trace in $U' \setminus P$. We obviously also have that $\sigma \in U \setminus P$. Since P is (U, O)-safety, there is a position $i \in \mathbb{N}$ in the trace σ such that the action at i is not in O and from this position onwards we cannot fix the violation, that is, $\sigma^{<i}\tau \notin P \cap U$, for every $\tau \in \Sigma^\infty$. Since $O' \subseteq O$, the action at position i is not in O' either. Similarly, we have that $\sigma^{<i}\tau \notin P \cap U'$, for every $\tau \in \Sigma^\infty$. Therefore, P is (U', O')-safety. □

We next prove that any set P of traces is (U, O)-safety, for some trace universe U and some set O of actions. Furthermore, for each O, there is a maximal trace universe U, in terms of set inclusion, for which P is (U, O)-safety.

LEMMA 3.6. *For every $P \subseteq \Sigma^\infty$ such that $\varepsilon \in P$ and every $O \subseteq \Sigma$, there is a trace universe $U \subseteq \Sigma^\infty$ with the following properties:*

(i) P is (U, O)-safety.
(ii) For every trace universe $U' \subseteq \Sigma^\infty$, if P is (U', O)-safety then $U' \subseteq U$.

PROOF. First, observe that there is always a trace universe U in which P is (U, O)-safety. In particular, P is $(\{\varepsilon\}, O)$-safety.

We also note that for every trace universe V, if P is (V,O)-safety then $P \cap V$ is prefix-closed. Indeed, suppose that σ is a prefix of σ' and $\sigma \notin P \cap V$ and $\sigma' \in P \cap V$. As V is prefix-closed, we have that $\sigma \in V \setminus P$. Then, there is a position $i \in \mathbb{N}$ such that $\sigma^{<i} \notin \Sigma^* \cdot O$ and $\{\sigma^{<i}\} \cdot \Sigma^\infty \cap P \cap V = \emptyset$. This last statement is a contradiction as $\sigma' \in \{\sigma^{<i}\} \cdot \Sigma^\infty \cap P \cap V$.

Let $\mathcal{V}$ be the set of all trace universes V such that P is (V,O)-safety and let $U := \bigcup_{V \in \mathcal{V}} V$. We prove by contradiction that P is (U,O)-safety. Then obviously the set U also satisfies condition (ii) in the lemma's statement.

Now, consider the set $S := \bigcap_{i \in \mathbb{N}} \{\sigma \in U \setminus P \mid \sigma^{<i} \in \Sigma^* \cdot O \text{ or } \{\sigma^{<i}\} \cdot \Sigma^\infty \cap P \cap U \neq \emptyset\}$. Since P is not (U,O)-safety by assumption, S is not empty. Consider some trace $\sigma \in S$. Then, as $\sigma \in U$, we have that $\sigma \in V$, for some set $V \in \mathcal{V}$. Since P is (V,O)-safety, there is a position $j \in \mathbb{N}$ such that $\sigma^{<j} \notin \Sigma^* \cdot O$ and $\{\sigma^{<j}\} \cdot \Sigma^\infty \cap P \cap V = \emptyset$. As $\sigma \in S$, we have that $\sigma^{<j} \in \Sigma^* \cdot O$ or $\{\sigma^{<j}\} \cdot \Sigma^\infty \cap P \cap U \neq \emptyset$. Thus $\{\sigma^{<j}\} \cdot \Sigma^\infty \cap P \cap U \neq \emptyset$. It follows that $\{\sigma^{<j}\} \cdot \Sigma^\infty \cap P \cap V' \neq \emptyset$, for some $V' \in \mathcal{V}$ with $V' \neq V$. That is, there is $\sigma' \in P \cap V'$ with $\sigma^{<j}$ a prefix of σ'. As $P \cap V'$ is prefix-closed, we have that $\sigma^{<j} \in P \cap V'$, and as V is prefix-closed we also have that $\sigma^{<j} \in P \cap V$, which contradicts $\{\sigma^{<j}\} \cdot \Sigma^\infty \cap P \cap V = \emptyset$. □

The following lemma characterizes (U,O)-safety in terms of prefix sets and limit closures. We make use of this characterization in Section 3.2, where we link (U,O)-safety to (U,O)-enforceability. 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 3.7. *For every $P, U \subseteq \Sigma^\infty$ and $O \subseteq \Sigma$, it holds that P is (U,O)-safety iff* $\mathrm{cl}(\mathrm{pre}_*(P \cap U) \cdot O^*) \cap U \subseteq P$.

PROOF. We rephrase Definition 3.1 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 $\sigma \in U \setminus P$. Then there is a position $i \in \mathbb{N}$ 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 a position $i \in \mathbb{N}$ 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 $\forall \sigma \in U.\ \sigma \in P \leftarrow \mathrm{pre}_*(\sigma) \subseteq \mathrm{pre}_*(P \cap U) \cdot O^*$. As $\mathrm{pre}_*(P \cap U) \cdot O^*$ is prefix-closed, it is also equivalent to $\forall \sigma \in U.\sigma \in P \leftarrow \sigma \in \mathrm{cl}(\mathrm{pre}_*(P \cap U) \cdot O^*)$, that is, $\mathrm{cl}(\mathrm{pre}_*(P \cap U) \cdot O^*) \cap U \subseteq P$. □

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 statement that ∞-safety is a necessary condition for a security policy to be enforceable by execution monitoring [Schneider 2000]. 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 is Σ^∞ and O equals $\emptyset$. 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 3.8. *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 2.1. 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 a position $i \in \mathbb{N}$ such that $\sigma^{<i} \notin A$. Let i be the minimal position 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, $\sigma^{<i}\tau \notin P \cap U$. This shows that σ satisfies the right-hand side of the implication in Definition 3.1. 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 the steps 1 to 5 until it either accepts or rejects σ.

Step 1. If $\mathcal{M}_U$ rejects σ, then $\mathcal{M}_{\mathrm{trunc}}$ rejects σ.
Step 2. If $\sigma = \varepsilon$, then $\mathcal{M}_{\mathrm{trunc}}$ accepts σ.
Step 3. If n is the length of σ and $\mathcal{M}_{\mathrm{trunc}}$ rejects $\sigma^{<n-1}$, then $\mathcal{M}_{\mathrm{trunc}}$ rejects σ.
Step 4. If $\mathcal{M}$ accepts σ, then $\mathcal{M}_{\mathrm{trunc}}$ accepts σ.
Step 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 $\sigma \in \mathrm{trunc}(A) \cap U$. To prove this, we use in the step case property (i) of Definition 2.1, which guarantees that $\mathcal{M}$ halts in step 4. We obtain that $\mathrm{trunc}(A) \cap U$ is decidable. Since $\mathrm{pre}_*(P \cap U) = \mathrm{pre}_*(\mathrm{cl}(\mathrm{trunc}(A)) \cap U) = \mathrm{trunc}(A) \cap U$, 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) through (iv) of Definition 2.1. 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 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 3.7, 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. □

Note that if $U \cap \Sigma^*$ is not decidable, then the conditions (1), (2), and (3) in Theorem 3.8 are sufficient for the set $P \subseteq \Sigma^\infty$ to be (U, O)-enforceable. However, they are no longer necessary. To see this, let $U \subseteq \Sigma^*$ be an undecidable trace universe. Such undecidable trace universes exist. For instance, if $\Sigma = \{0, 1\}$ and if $A \subseteq \{0\}^*$ is an undecidable set, then $U := \mathrm{pre}(A \cdot \{1\}) \subseteq \Sigma^*$ is an undecidable trace universe. The Turing machine $\mathcal{M}$ that halts on and accepts every input satisfies the conditions of Definition 2.1 for the property of traces $P = \Sigma^\infty$ and the set $O = \emptyset$ of observable actions. Therefore, P is (U, O)-enforceable. However, condition (2) in Theorem 3.8 does not hold in this case, that is, the set $\mathrm{pre}_*(P \cap U) = U$ is undecidable.

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 [Ligatti et al. 2005, 2009; Schneider 2000]. For instance, Schneider [2000] introduces security automata—also known as truncation automata [Ligatti et al. 2005, 2009]—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 automaton model, namely, pushdown automata, the problem is undecidable.

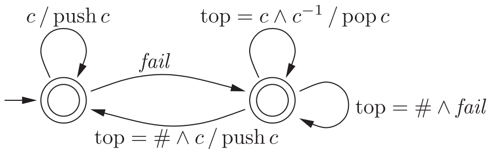
4.1.1. Automata. We start by defining pushdown and finite-state automata. Since trace properties are sets of finite and infinite sequences, we equip the automata with two sets of accepting states, one for finite sequences and the other for infinite sequences.

A *pushdown automaton (PDA)* $\mathcal{A}$ is a tuple $(Q, \Sigma, \Gamma, \delta, q_\mathrm{I}, F, B)$, where

— Q is a finite set of states,
— Σ is a finite nonempty alphabet,
— Γ is a finite stack alphabet with $\# \in \Gamma$,
— $\delta : Q \times \Sigma \times \Gamma \to 2^{Q \times \Gamma^*}$ is the transition function, where $\delta(q, a, b)$ is a finite set, for all $q \in Q$, $a \in \Sigma$, and $b \in \Gamma$,
— $q_\mathrm{I} \in Q$ is the initial state,
— $F \subseteq Q$ is the set of accepting states for finite sequences, and
— $B \subseteq Q$ is the set of accepting states for infinite sequences.

The *size* of $\mathcal{A}$, denoted by $\|\mathcal{A}\|$, is the cardinality of Q. $\mathcal{A}$ is *deterministic* if $\delta(q, a, b)$ is either the empty set or a singleton, for every $q \in Q$, $a \in \Sigma$, and $b \in \Gamma$. If $\delta(q, a, b)$ is always nonempty, for $q \in Q$, $a \in \Sigma$, and $b \in \Gamma$, then $\mathcal{A}$ is *complete*.

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



c / push c
top $= c \wedge c^{-1}$ / pop c
fail
top $= \# \wedge$ *fail*
top $= \# \wedge c$ / push c

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

$(q_0, u_0) = (q_\mathrm{I}, \#)$ 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 the infinite sequence $a_0 a_1 \ldots \in \Sigma^\omega$ 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, that is, 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, that is, $\delta(q, a, b) = \delta(q, a, b')$, 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 then simplify to sequences in Q^* and Q^ω, respectively.

The following example illustrates PDAs, their expressiveness, and their use in specifying policies. Furthermore, we illustrate the characterization in Theorem 3.8 to determine whether a specified policy is enforceable.

Example 4.1. 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 recognizes this language.

Observe that this policy is $(\Sigma^\infty, \emptyset)$-enforceable. Indeed, the conditions in Theorem 3.8 are satisfied: (1) L contains the empty sequence, (2) $\mathrm{pre}_*(L) = F^* \cdot (C^* \cup \mathrm{pre}_*(F))$ is decidable, and (3) $\mathrm{cl}(\mathrm{pre}_*(L)) = F^* \cdot (C^* \cup \mathrm{pre}_*(F)) \cup F^* \cdot C^\omega \cup F^\omega = L$ 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 $c_1 c_2 \mathit{fail}\, c_2^{-1} \mathit{fail}\, c_1^{-1}$.

4.1.2. Decision Problems. 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 4.2. *Let Σ be the alphabet $\{0, 1\}$. For a PDA $\mathcal{A}$ with alphabet Σ, it is undecidable whether $L(\mathcal{A})$ is $(\Sigma^\infty, \emptyset)$-safety.*

PROOF. Recall that the universality problem for context-free grammars is undecidable [Hopcroft and Ullman 1979]. That means, we cannot decide if $L_*(\mathcal{A}) = \Sigma^*$, for a given PDA $\mathcal{A}$.

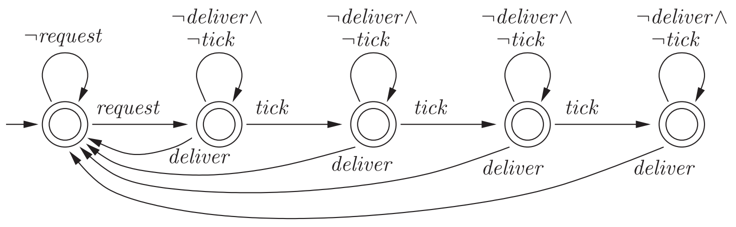


Fig. 2. Finite-state automaton.

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 3.7, $L(\mathcal{A}')$ is $(\Sigma^\infty, \emptyset)$-safety iff $L_*(\mathcal{A}) = \Sigma^*$. □

THEOREM 4.3. *Let Σ be the alphabet $\{0, 1\}$. For a PDA $\mathcal{A}$ with alphabet Σ, it is undecidable 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 3.8, 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 2.2. 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 2.2. There is a state from which the observable *tick* action leads to the nonacceptance of the input sequence. In general, the problem is PSPACE-complete as shown in Corollary 4.5 after Theorem 4.4.

THEOREM 4.4. *The decision problem of determining whether $L(\mathcal{A})$ is $(L(\mathcal{U}), O)$-safety is PSPACE-complete, where $\mathcal{A}$ and $\mathcal{U}$ are FSAs with alphabet Σ, $L(\mathcal{U})$ is a trace universe, and $O \subseteq \Sigma$.*

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 [Hopcroft and Ullman 1979].

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 4.2, $L(\mathcal{A}')$ is $(\Sigma^\infty, \emptyset)$-safety iff $L_*(\mathcal{A}) = \Sigma^*$. Hence 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 $\mathcal{Y} := (Q, \Sigma, \delta, q_\mathrm{I}, F_\mathcal{Y}, \emptyset)$ recognizes the language $L(\mathcal{Y}) = \mathrm{pre}_*(L(\mathcal{X}))$.
— If $\mathrm{pre}_*(L(\mathcal{X})) = L(\mathcal{X}) \cap \Sigma^*$, the FSA $\mathcal{Z} := (Q, \Sigma, \delta, q_\mathrm{I}, F, F)$ recognizes 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 $\mathcal{B}$ whose size is polynomial in $\|\mathcal{A}\| + \|\mathcal{U}\|$, such that $L(\mathcal{B}) = \mathrm{cl}(\mathrm{pre}_*(L(\mathcal{A}) \cap L(\mathcal{U})) \cdot O^*) \cap L(\mathcal{U})$. By Lemma 3.7, $L(\mathcal{A})$ is $(L(\mathcal{U}), O)$-safety iff $L(\mathcal{B}) \subseteq L(\mathcal{A})$. Since the inclusion problem for FSAs is in PSPACE (see, for example, Vardi [1995]), our problem is therefore also in PSPACE. □

COROLLARY 4.5. *The decision problem of determining whether $L(\mathcal{A})$ is $(L(\mathcal{U}), O)$-enforceable is PSPACE-complete, where $\mathcal{A}$ and $\mathcal{U}$ are FSAs with alphabet Σ, $L(\mathcal{U})$ is a trace universe, and $O \subseteq \Sigma$.*

PROOF. The proof, similar to that of Theorem 4.3, is an easy consequence of Theorems 3.8 and 4.4. □

4.1.3. Extensions. The usefulness of a yes-no answer to the question of whether a policy P is (U, O)-enforceable, for a trace universe U and a set O of observable actions is often limited. We can extend our algorithmic solution for this decision problem, where P and U are given as FSAs, so that a more detailed answer is provided.

Case I: P is (U, O)-enforceable. In this case, the FSAs $\mathcal{A}$ for P and $\mathcal{U}$ for U can serve as a basis for implementing an enforcement mechanism. Similar to how Schneider [2000] uses his security automata, we can use $\mathcal{A}$ and $\mathcal{U}$ directly to build an enforcement mechanism E for the policy P: E initially stores the singleton set consisting of $\mathcal{A}$'s initial state and the singleton set consisting of $\mathcal{U}$'s initial state. Whenever E intercepts a system action $a \in \Sigma$, it updates these sets by determining the successor states of the stored states using the transition functions of $\mathcal{A}$ and $\mathcal{U}$. We remove from $\mathcal{A}$'s updated set the states from which we do not accept any sequence that is accepted from some state from the updated set of $\mathcal{U}$. E terminates the system if $\mathcal{A}$'s set becomes empty. This set will only be empty when the intercepted action a is controllable since P is (U, O)-enforceable. Otherwise, E continues by intercepting the next system action. Note that we must take $\mathcal{U}$ into account since a trace might be extendable to sequences in $L(\mathcal{A})$, all of which are not in $L(\mathcal{U})$.

Alternatively, instead of using $\mathcal{A}$ and $\mathcal{U}$ directly, we can first construct a deterministic FSA $\mathcal{B}$ that recognizes the language $L(\mathcal{A}) \cap L(\mathcal{U})$ and use $\mathcal{B}$ instead of $\mathcal{A}$ and $\mathcal{U}$. We can ignore here $\mathcal{A}$'s and $\mathcal{U}$'s acceptance condition for the infinite sequences, since the enforcement mechanism does not need to handle infinite traces. We can therefore use the standard powerset construction and product construction for finite word automata [Rabin and Scott 1959] to build $\mathcal{B}$. We can also remove the states from which no accepting state is reachable. Furthermore, we can minimize the resulting FSA [Hopcroft 1971]. Note that with the deterministic FSA $\mathcal{B}$ we obtain a more efficient enforcement mechanism: whenever the enforcement mechanism intercepts an action it must only update $\mathcal{B}$'s current state instead of updating the sets of states of $\mathcal{A}$ and $\mathcal{U}$. However, $\mathcal{B}$'s size may be exponential in $\|\mathcal{A}\| + \|\mathcal{U}\|$. This exponential blow-up can be avoided when building a nondeterministic FSA recognizing the intersection of the languages $L_*(\mathcal{A})$ and $L_*(\mathcal{U})$. However, the enforcement mechanism then still needs to update a set of states.

Case II: P is not (U, O)-enforceable. In this case, we can return a witness that explains why the policy P is not enforceable. This witness can be computed analogously to finding counterexamples in finite-state model checking, where they are extremely helpful for understanding why a given system does not fulfill the specification and for fixing the system [Clarke et al. 2007].

Let $\mathcal{A}$ be the given FSA for the policy P and let $\mathcal{U}$ be the FSA for the trace universe U. If $\varepsilon \notin L(\mathcal{A})$, the empty sequence ε is such a witness; this suggests some vacuity with respect to the policy specification $\mathcal{A}$. If $\varepsilon \in L(\mathcal{A})$, a witness is either a sequence in $U \setminus P$ for which there is a suffix in P or a sequence in $U \setminus P$ that would not be prevented. The first case demonstrates that an enforcement mechanism cannot guarantee the transparency requirement. The second case demonstrates that the soundness requirement cannot be guaranteed while also guaranteeing the transparency requirement. We can use techniques from the automata-theoretic approach to finite-state model checking (see, for example, Vardi [1995, 2007] and references therein) to determine a witness.

One possibility is, given the FSA $\mathcal{B}$ from the proof of Theorem 4.4, to construct an FSA that recognizes the language $L(\mathcal{B}) \setminus L(\mathcal{A})$, which is the set of all witnesses. We can utilize standard constructions for finite and infinite word automata for complementation and intersection to construct such an FSA.

Alternatively, or in addition to returning a witness, we can determine the maximal trace universe $W \subseteq \Sigma^\infty$ in which P is (W, O)-enforceable, if $\varepsilon \in P$. It is easy to see by Lemma 3.6 that such a maximal trace universe W always exists. As shown by the following theorem, we can build an FSA for this trace universe.

THEOREM 4.6. *For every FSA $\mathcal{A}$ with $\varepsilon \in L(\mathcal{A})$, there is an FSA $\mathcal{W}$ with the following properties.*

— *$L(\mathcal{W})$ is a trace universe.*
— *$L(\mathcal{A})$ is $(L(\mathcal{W}), O)$-enforceable.*
— *For every trace universe $V \subseteq \Sigma^\infty$, if $L(\mathcal{A})$ is (V, O)-enforceable then $V \subseteq L(\mathcal{W})$.*

PROOF. We sketch the construction of the FSA $\mathcal{W}$. We first determinize $\mathcal{A}$ using the powerset construction for finite word automata, where we ignore $\mathcal{A}$'s acceptance condition for infinite sequences. Let F be the set of accepting states for the finite sequences of the resulting FSA and $\bar{F}$ its complement.

From this deterministic and complete FSA, we obtain the FSA $\mathcal{W}_1$ as follows. We remove every transition that reaches a state in $\bar{F}$ from a state in F via an action in O. We also remove every transition that reaches a state in F from a state in $\bar{F}$. Finally, we make every state accepting for the finite sequences. The FSA $\mathcal{W}_1$ recognizes the trace universe that is maximal for $L(\mathcal{A})$'s enforceability with respect to finite sequences. For the infinite sequences, we build the FSAs $\mathcal{W}_2$ and $\mathcal{W}_3$, which we obtain from $\mathcal{W}_1$: Let $\mathcal{W}_2$ be the FSA $\mathcal{W}_1$ with the empty set of accepting states for the finite sequences and where F is the set of accepting states for the infinite sequences. Analogously, let $\mathcal{W}_3$ be the FSA $\mathcal{W}_1$ with the empty set of accepting states for the finite sequences and where $\bar{F}$ is the set of accepting states for the infinite sequences.

Let $\mathcal{W}$ be the FSA that recognizes the language $L_*(\mathcal{W}_1) \cup (L_\omega(\mathcal{W}_2) \cap L_\omega(\mathcal{A})) \cup (L_\omega(\mathcal{W}_3) \setminus L_\omega(\mathcal{A}))$, which we can obtain by standard constructions for finite and infinite word automata. It is straightforward to check that $L(\mathcal{W})$ is a trace universe and that $L(\mathcal{A})$ is $(L(\mathcal{W}), O)$-enforceable. By Lemma 3.6, there is a maximal trace universe $W \subseteq \Sigma^\infty$ in which $L(\mathcal{A})$ is (W, O)-safety. It suffices to show that $W = L(\mathcal{W})$.

We start with the inclusion $L(\mathcal{W}) \subseteq W$. Consider some $\sigma \in L(\mathcal{W})$. If $\sigma \in L(\mathcal{A})$, then either $\sigma \in \Sigma^*$ or $\sigma \in \Sigma^\omega \cap L_\omega(\mathcal{A})$, that is, $\sigma \in L_*(\mathcal{W}_1) \cap L_*(\mathcal{A})$ or $\sigma \in L_\omega(\mathcal{W}_2)$. In both cases, we have $\mathrm{pre}_*(\sigma) \subseteq L(\mathcal{A})$ and therefore $\sigma \in W$. If $\sigma \notin L(\mathcal{A})$, then either $\sigma \in \Sigma^*$ or $\sigma \in \Sigma^\omega \setminus L_\omega(\mathcal{A})$, that is, $\sigma \in L_*(\mathcal{W}_1) \setminus L_*(\mathcal{A})$ or $\sigma \in L_\omega(\mathcal{W}_3)$. In both cases, there is some $i \in \mathbb{N}$ such that $\sigma^{<i} \notin L(\mathcal{A})$. Let i be the smallest such nonnegative integer. Note that $i \geq 1$. Then, we have $\sigma^{<i} \notin \Sigma^* \cdot O$, and for all $j \in \mathbb{N}$, $\sigma^{<j} \in L(\mathcal{A})$ iff $j < i$. It follows that $\sigma \in W$.

It remains to show the inclusion $W \subseteq L(\mathcal{W})$. First, consider some $\sigma \in \Sigma^\infty$ with $\mathrm{pre}_*(\sigma) \subseteq L_*(\mathcal{W}_1)$ as well as its unique run, that is, the unique sequence of states $s_0 s_1 \ldots$ of the run of $\mathcal{W}_1$ on σ. Note that if $s_i \in \bar{F}$, for some $i \in \mathbb{N}$, then $s_j \in \bar{F}$, for every $j \geq i$. From this we infer the equalities $L_\omega(\mathcal{W}_2) = \mathrm{cl}(L_*(\mathcal{W}_1)) \cap \mathrm{cl}(L_*(\mathcal{A}))$ and $L_\omega(\mathcal{W}_3) = \mathrm{cl}(L_*(\mathcal{W}_1)) \setminus \mathrm{cl}(L_*(\mathcal{A}))$.

Now, consider some $\sigma \in W$. Since W is prefix-closed, we have that $\sigma \in W$ iff $\mathrm{pre}(\sigma) \subseteq W$, for every $\sigma \in \Sigma^\infty$. It follows from the Lemmas 3.5 and 3.6 that $\sigma \in W$ iff $L(\mathcal{A})$ is $(\mathrm{pre}(\sigma), O)$-safety. This is equivalent to $\forall i \leq |\sigma|. \sigma^{<i} \notin L(\mathcal{A}) \to \exists j \in \mathbb{N}. j \leq i \land \sigma^{<j} \notin \Sigma^* \cdot O \land \forall k \geq j. \sigma^{<k} \notin L(\mathcal{A})$. It follows from the preceding equivalence that $L(\mathcal{A}) \cap \mathrm{pre}(\sigma)$

is prefix-closed. Moreover, if $\sigma \notin L(\mathcal{A})$, then there is some $i \in \mathbb{N}$ such that $\sigma^{<i} \notin L(\mathcal{A})$. If we consider the smallest such i, then $\sigma^{<i} \notin \Sigma^* \cdot O$. Therefore, $\mathrm{pre}_*(\sigma) \subseteq L_*(\mathcal{W}_1)$. We have the following cases.

(1) If $\sigma \in \Sigma^*$, then $\sigma \in L_*(\mathcal{W}_1)$.
(2) If $\sigma \in \Sigma^\omega \cap L(\mathcal{A})$, then $\mathrm{pre}_*(\sigma) \subseteq L_*(\mathcal{W}_1) \cap L_*(\mathcal{A})$. In this case, $\sigma \in L_\omega(\mathcal{W}_2)$ and thus $\sigma \in L_\omega(\mathcal{W}_2) \cap L_\omega(\mathcal{A})$.
(3) If $\sigma \in \Sigma^\omega \setminus L(\mathcal{A})$, then $\mathrm{pre}_*(\sigma) \subseteq L_*(\mathcal{W}_1)$ but $\mathrm{pre}_*(\sigma) \not\subseteq L_*(\mathcal{A})$. In this case, $\sigma \in L_\omega(\mathcal{W}_3)$ and thus $\sigma \in L_\omega(\mathcal{W}_3) \setminus L_\omega(\mathcal{A})$.

This proves that $W \subseteq L(\mathcal{W})$. □

We can use the FSA $\mathcal{W}$ to check whether the policy P can be enforced on a given system S, where O is the set of observable actions and $\Sigma \setminus O$ is the set of controllable actions. To answer this question, it suffices to check that every trace of S is in $L(\mathcal{W})$. If S is a finite-state system, this can be done again by automata-theoretic techniques. Note that the trace universe is not fixed here. Instead, we first construct the largest trace universe in which P is enforceable and then check whether the behavior of S is included in it.

4.2. Logic-Based Specification Languages

Temporal logics are prominent specification languages for expressing properties on traces [Pnueli 1977]. In the following, we consider the realizability of an enforcement mechanism for policies specified in a linear-time temporal logic with future and past operators, and metric constraints [Alur and Henzinger 1992; Koymans 1990].

4.2.1. Temporal Logic. 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

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

where p ranges over the propositions in $\mathcal{P}$ and I ranges over the nonempty intervals over $\mathbb{N}$, that is, subsets of the form $\{n, n+1, \dots, m\}$ and $\{n, n+1, \dots\}$ 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 $\lceil \log(1 + \min I + \max I) \rceil$ for a finite interval I, and $\lceil \log(1 + \min I) \rceil$ for an infinite interval I.

The truth value of a formula φ is defined over timestamped sequences of propositional models, 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 $i \in \mathbb{N}$ and $i < |\sigma|$. 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 $i \in \mathbb{N}$ 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.

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

The temporal operators $\bullet_I$ ("previous"), $\circ_I$ ("next"), S_I ("since"), and U_I ("until") allow us to express both quantitative and qualitative properties with respect to the ordering of elements in the sequence σ and their timestamps in the sequence t. Note that the temporal operators are labeled with intervals I and the sequences σ and t only satisfy a formula of the form $\bullet_I\, \varphi$, $\circ_I\, \varphi$, $\varphi\, \mathsf{S}_I\, \psi$, or $\varphi\, \mathsf{U}_I\, \psi$ at the time point i, if it is satisfied within the bounds given by the interval I of the respective temporal operator, which are relative to the current timestamp t_i.

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 [Eisner et al. 2003], 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 [Alur and Henzinger 1992] and Basin et al. [2012b] 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 monitor the system continuously, but only at specific points in time.

In the following, we use standard syntactic sugar. For instance, $\varphi \wedge \psi$ abbreviates $\neg(\neg\varphi \vee \neg\psi)$, $\Diamond_I\, \varphi$ ("eventually") abbreviates $\mathit{true}\, \mathsf{U}_I\, \varphi$, and $\Box_I\, \varphi$ ("always") 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.

Example 4.7. We return to the policies from Example 2.2. 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

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

The trace properties described by φ_1 and φ_2 differ from the trace properties P_1 and P_2 from Example 2.2 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, that is, 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 2.2 contained only one system action at a time. Here, we consider traces in which multiple system actions can happen at the same time point. 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 2.2 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 particular, the empty set is not controllable. We define $\hat{O} := \{a \in 2^{\mathcal{P}} \mid a \subseteq O\}$. The rational behind this definition is that whenever the enforcement mechanism intercepts a set a of actions in which at least one action is controllable, it can terminate the target system and prevent all the actions in a from happening.

4.2.2. Realizability. 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 4.8. *The decision problem of determining whether $L(\varphi)$ is $(L(\mathcal{U}), \hat{O})$-enforceable is PSPACE-complete, where φ is an LTL formula, $\mathcal{U}$ is an FSA such that $L(\mathcal{U}) \subseteq (2^{\mathcal{P}})^\infty$ is a trace universe, and $O \subseteq \mathcal{P}$.*

PROOF. By Theorem 3.8 $L(\varphi)$ 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 and therefore 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 [Sistla and Clarke 1985]. 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 3.7, 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 [Dax et al. 2010; Vardi and Wolper 1994]. Using standard automata constructions and the constructions from the proof of Theorem 4.4, 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 [Jones 1975] 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 4.8 to obtain an enforcement mechanism for $L(\varphi)$. Building a deterministic FSA from $\mathcal{A}$ and $\mathcal{U}$ for the enforcement mechanism, as discussed in Case I of Section 4.1.3, might be prohibitive here since the size of $\mathcal{A}$ is in the worst case already exponential in $\|\varphi\|$.

For MLTL, the corresponding decision problem can be solved by reducing it to the one for LTL. However, the decision problem for MLTL has higher complexity, namely, EXPSPACE-complete, as shown in the following theorem. Intuitively, this blow-up is caused by translating MLTL to LTL formulas by unfolding the metric constraints that are represented by intervals attached to the temporal operators. This unfolding causes an exponential blow-up in the formula size. Note that the (syntactic) size of an interval is logarithmic in its bounds.

THEOREM 4.9. *The decision problem of determining whether $L(\varphi)$ is $(L(\mathcal{U}) \otimes T, \hat{O} \times \mathbb{N})$-enforceable is EXPSPACE-complete, where φ is an MLTL formula, $\mathcal{U}$ is an FSA such that $L(\mathcal{U}) \subseteq (2^{\mathcal{P}})^\infty$ is a trace universe, and $O \subseteq \mathcal{P}$.*

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
— $\tau \in L(\mathcal{A}) \otimes T$ 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 $\overline{\tau} := \{tick\}^\omega$.
— If $\tau \neq \varepsilon$ is finite, then $\overline{\tau} := \{tick\}^{t_0}\sigma_0\{tick\}^{d_1}\sigma_1\{tick\}^{d_2}\sigma_2 \ldots \sigma_{|\tau|-1}\{tick\}^\omega$.

Here $d_i := t_i - t_{i-1}$, $\{tick\}^i$ 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 $\{\overline{\tau} \in \overline{\Sigma}^\infty \mid \tau \in L\}$. This mapping is one-to-one and therefore 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 $\ulcorner \varphi \mathsf{U}_I \psi \urcorner$.
— $\overline{\varphi} := (\Box\, tick) \vee (tick \mathsf{U} (\neg tick \wedge \ulcorner \varphi \urcorner))$.

By induction over φ, one verifies that $\sigma, t, i \models \varphi$ iff $\overline{\tau}, t', i + t_i \models \ulcorner \varphi \urcorner$ for all $i < |\tau|$, where $\tau = \sigma \otimes t$ and t' is any sequence of timestamps. Note that the timestamps in t' are irrelevant since the temporal operators in $\ulcorner \varphi \urcorner$ do not contain any metric constraints, that is, the interval attached to any temporal operator is $\mathbb{N}$. It follows that $\tau \in L(\varphi)$ iff $\overline{\tau} \in L(\overline{\varphi})$.

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 $\overline{\tau} \in L(\overline{\mathcal{A}})$. Note that $L(\overline{\mathcal{A}}) \subseteq \overline{\mathcal{T}}$.

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 $\operatorname{pre}_*(L(\varphi))$ iff $\overline{\tau}^{<|\tau|+t_{|\tau|-1}}$ is in $\operatorname{pre}_*(L(\overline{\varphi}) \cap \overline{\mathcal{T}})$. Since $\operatorname{pre}_*(L(\overline{\varphi}) \cap \overline{\mathcal{T}})$ is decidable, so is $\operatorname{pre}_*(L(\varphi) \cap U_T)$. 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 [Alur and Henzinger 1994]. 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 $S_\varphi := \operatorname{cl}(\operatorname{pre}_*(L(\varphi) \cap U_T) \cdot (\hat{O} \times \mathbb{N})^*) \cap U_T$. We have that S_φ is mapped to $\overline{S_\varphi} = \big(\operatorname{pre}_*(L(\overline{\varphi}) \cap \overline{U_T}) \cdot \hat{O}_t^{\,\omega} \cup \big(\operatorname{cl}(\operatorname{pre}_*(L(\overline{\varphi}) \cap \overline{U_T})) \cap L(\theta')\big)\big) \cap \overline{U_T}$. 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 4.4, 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 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 $\overline{\mathcal{A}}$ and $\overline{\mathcal{U}}$ from the proof of Theorem 4.9 to obtain an enforcement mechanism E. See also Case I in Section 4.1.3. We first construct an FSA $\mathcal{C}$ that recognizes the intersection of $L(\overline{\mathcal{A}})$ and $L(\overline{\mathcal{U}})$. 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 to the progression of time, that is, 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. Otherwise, it continues by intercepting the next action.

5. RELATED WORK

Schneider [2000] 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 automaton model, called security automata, that recognizes ∞-safety properties. This spurred considerable follow-up research both practical and theoretical. For example, Erlingsson and Schneider [1999, 2000] implemented and evaluated enforcement mechanisms based on execution monitoring.

With respect to theoretical follow-ups, Ligatti et al. [2005, 2009] introduced edit automata, which are transducers with infinitely many states. Edit automata can recognize trace properties that are not ∞-safety. They were motivated by enforcement mechanisms that can insert and delete system actions in addition to terminating a system in case of a policy violation. However, it remains unclear how to use edit automata as general-purpose enforcement mechanisms, in particular, how an edit automaton and a system interact with each other in general. Ligatti and Reddy [2010] 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. Fong [2004] analyzed classes of security policies that can be recognized by shallow-history automata, a restricted class of security automata. Talhi et al. [2008] extended the work by Fong [2004] by introducing so-called bounded-history automata, which are security automata and edit automata with bounded memory. Falcone et al. [2011] studied the trace properties that can be recognized by different automaton models in terms of the safety-progress hierarchy [Chang et al. 1992] of regular languages and classical finite-state automaton models. Hamlen et al. [2006] related the policies that can be enforced by program rewriting to those that can be recognized by security automata. The notion of policy enforcement relative to a trace universe was briefly mentioned by Ligatti et al. [2009] and further developed by Chabot et al. [2011].

With the exception of Talhi et al. [2008] and Chabot et al. [2011], none of the preceding works examines the problem of realizing an enforcement mechanism from a policy description and the computational complexity of the associated decision problem. For a finite-state automaton, which describes the security policy, the algorithm presented by Talhi et al. [2008] decides whether a bounded-history automaton for enforcement exists, if the language of the given finite-state automaton is from a certain subclass of the regular languages, the so-called locally testable languages [McNaughton and Papert 1971]. If the described language is not locally testable, it returns unknown. The algorithm presented by Chabot et al. [2011] takes as input a finite-state automaton and a finite-state transition system, which respectively describe the security policy and the target system. It then checks whether the policy can be enforced on the given system. Moreover, if the policy is enforceable, it returns a secured finite-state transition system.

All the preceding 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. To the best of our knowledge, this is the first investigation of the impact of this distinction in the domain of policy enforcement.

Note that similar classifications of system actions, signals, transitions, and states into ones that are controllable and observable are common in other areas, like control theory and software testing. In particular, the Ramadge-Wonham framework from control theory [Ramadge and Wonham 1987] has several similarities with our setting, and the domain of policy enforcement in general. In this framework, processes are modeled as deterministic transition systems. A process S, called the supervisor, can block a transition of another process G, called the generator, if it is labeled with a controllable event. Transitions labeled with observable events cannot be blocked by the supervisor S. Furthermore, S has no means of forcing G to take a transition. S plays the role of an enforcement mechanism in the domain of policy enforcement and G's role is that of a target system. The characteristics of controllable and observable events are similar to our classification of system actions. The fundamental problem in supervisory control theory is whether a supervisor S exists for a given process G such that the behavior of their combination, denoted by S/G, fulfills a specification that is given by two languages L_g and L_a, where L_g contains the legal behavior and L_a contains the minimal acceptable behavior, that is, the language of S/G is a subset of L_g and a superset of L_a. This problem is closely related to the problem of whether a policy P is enforceable in our setting. In particular, when identifying L_g with P and L_a with the behavior of G that is policy compliant, one asks the question whether the policy P can be enforced on the target system G. Note that in this question the target system G is part of the input instance and is a deterministic transition system. If one further identifies the behavior of G with the trace universe U, one asks the question whether P is enforceable on systems for which their executions are in U. It remains to be seen whether and how the domain of policy enforcement can benefit from the Ramadge-Wonham framework and the results around it (and also vice versa).

The problem of checking whether a system's observed behavior is compliant with security policies, regulations, and laws has recently attracted considerable attention. This problem is simpler than policy enforcement since one needs only to detect and report policy violations. For this, system actions need only to be observable. Monitoring approaches have proved useful here, based either on offline algorithms [Garg et al. 2011] or online algorithms [Basin et al. 2008; Hallé and Villemaire 2012]. See also Basin et al. [2010, 2011]. A closely related topic to the enforcement of security policies by execution monitoring and compliance checking is runtime verification, where processes analyze systems during their execution. In particular, the monitoring processes detect and report violations with respect to specifications, which are often given by temporal logic formulas. See, for example, Havelund and Roşu [2001], Kim et al. [1999], Barringer et al. [2004], and Havelund [2000] for some of the pioneering work in this field.

Another generalization of the standard definition of safety [Alpern and Schneider 1985] has been given by Ehlers and Finkbeiner [2011]. 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 [Ligatti and Reddy 2010].

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 some actions cannot be controlled. 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 decision problem.

Note that our system architecture, described in Section 2.1, implicitly assumes that an enforcement mechanism E does not affect the executed trace other than shortening it due to system termination. This means that no observable action can be executed by the system S during the time E processes a previously intercepted action. This assumption on S and E's interaction is analogous to the synchronicity hypotheses [Benveniste and Berry 1991] in reactive languages, where one assumes that systems produce their outputs synchronously with their inputs. If we wish to build enforcement mechanisms for systems with hard real-time requirements then the notion of enforceability needs to be refined to account for the enforcement mechanism's processing times. In particular, *tick* actions can occur while E processes an action. As future work we plan to investigate the problem of enforceability in the general setting where observable actions may happen during E's execution.

As future work, we will also 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

Alpern, B. and Schneider, F. B. 1985. Defining liveness. *Inf. Process. Lett. 21*, 4, 181–185.

Alur, R. and Henzinger, T. A. 1992. Logics and models of real time: A survey. In *Proceedings of the REX Workshop on Real-Time: Theory in Practice.* Lecture Notes in Computer Science, vol. 600, Springer, 74–106.

Alur, R. and Henzinger, T. A. 1994. A really temporal logic. *J. ACM 41*, 1, 181–203.

American National Standards Institute, Inc. 2004. *Role Based Access Control*. American National Standards Institute, Inc., Washington, DC.

Barringer, H., Goldberg, A., Havelund, K., and Sen, K. 2004. Rule-based runtime verification. In *Proceedings of the 5th International Conference on Verification, Model Checking and Abstract Interpretation*. Lecture Notes in Computer Science, vol. 2937, Springer, 44–57.

Basin, D., Olderog, E.-R., and Sevinç, P. E. 2007. Specifying and analyzing security automata using CSP-OZ. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security*. ACM Press, New York, 70–81.

Basin, D., Klaedtke, F., Müller, S., and Pfitzmann, B. 2008. Runtime monitoring of metric first-order temporal properties. In *Proceedings of the 28th Conference on Foundations of Software Technology and Theoretical Computer Science*. Leibniz International Proceedings in Informatics Series, vol. 2, Schloss Dagstuhl - Leibniz Center for Informatics, 49–60.

Basin, D., Klaedtke, F., and Müller, S. 2010. Monitoring security policies with metric first-order temporal logic. In *Proceedings of the 15th ACM Symposium on Access Control Models and Technologies*. ACM Press, New York, 23–33.

Basin, D., Harvan, M., Klaedtke, F., and Zălinescu, E. 2011. Monitoring usage-control policies in distributed systems. In *Proceedings of the 18th International Symposium on Temporal Representation and Reasoning*. IEEE Computer Society, 88–95.

Basin, D., Jugé, V., Klaedtke, F., and Zălinescu, E. 2012a. Enforceable security policies revisited. In *Proceedings of the 1st Conference on Principles of Security and Trust*. Lecture Notes in Computer Science, vol. 7215, Springer, 309–328.

Basin, D., Klaedtke, F., and Zălinescu, E. 2012b. Algorithms for monitoring real-time properties. In *Proceedings of the 2nd International Conference on Runtime Verification*. Lecture Notes in Computer Science, vol. 7186, Springer, 260–275.

Benveniste, A. and Berry, G. 1991. The synchronous approach to reactive and real-time systems. *Proc. IEEE 79*, 9, 1270–1282.

Chabot, H., Khoury, R., and Tawbi, N. 2011. Extending the enforcement power of truncation monitors using static analysis. *Comput. Secur. 30*, 4, 194–207.

Chang, E. Y., Manna, Z., and Pnueli, A. 1992. Characterization of temporal property classes. In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*. Lecture Notes in Computer Science, vol. 623, Springer, 474–486.

Clarke, E. M., Emerson, E. A., and Sifakis, J. 2007. Model checking: Algorithmic verification and debugging. *Comm. ACM 52*, 11, 75–84.

Clarkson, M. R. and Schneider, F. B. 2010. Hyperproperties. *J. Comput. Secur. 18*, 6, 1157–1210.

Dax, C., Klaedtke, F., and Lange, M. 2010. On regular temporal logics with past. *Acta Inf. 47*, 4, 251–277.

Ehlers, R. and Finkbeiner, B. 2011. Reactive safety. In *Proceedings of 2nd International Symposium on Games, Logics and Formal Verification*. Electronic Proceedings in Theoretical Computer Science, vol. 54, 178–191.

Eisner, C., Fisman, D., Havlicek, J., Lustig, Y., McIsaac, A., and Van Campenhout, D. 2003. Reasoning with temporal logic on truncated paths. In *Proceedings of the 15th International Conference on Computer Aided Verification*. Lecture Notes in Computer Science, vol. 2725, Springer, 27–39.

Erlingsson, Ú. 2004. The inlined reference monitor approach to security policy enforcement. Ph.D. thesis, Cornell University, Ithaca, NY.

Erlingsson, Ú. and Schneider, F. B. 1999. SASI enforcement of security policies: A retrospective. In *Proceedings of the Workshop on New Security Paradigms*. ACM Press, New York, 87–95.

Erlingsson, Ú. and Schneider, F. B. 2000. IRM enforcement of Java stack inspection. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society, 246–255.

Falcone, Y., Mounier, L., Fernandez, J.-C., and Richier, J.-L. 2011. Runtime enforcement monitors: Composition, synthesis, and enforcement abilities. *Form. Methods Syst. Des. 38*, 2, 223–262.

Fong, P. W. 2004. Access control by tracking shallow execution history. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society, 43–55.

Garg, D., Jia, L., and Datta, A. 2011. Policy auditing over incomplete logs: Theory, implementation and applications. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*. ACM Press, New York, 151–162.

Hallé, S. and Villemaire, R. 2012. Runtime enforcement of web service message contracts with data. *IEEE Trans. Serv. Comput. 5*, 2, 192–206.

Hamlen, K. W., Morrisett, G., and Schneider, F. B. 2006. Computability classes for enforcement mechanisms. *ACM Trans. Progr. Lang. Syst. 28*, 1, 175–205.

Havelund, K. 2000. Using runtime analysis to guide model checking of java programs. In *Proceedings of the 7th International SPIN Workshop*. Lecture Notes in Computer Science, vol. 1885, Springer, 245–264.

Havelund, K. and Roşu, G. 2001. Monitoring programs using rewriting. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering*. IEEE Computer Society, 135–143.

Henzinger, T. A. 1992. Sooner is safer than later. *Inform. Process. Lett. 43*, 3, 135–141.

Hopcroft, J. E. 1971. An n log n algorithm for minimizing the states in a finite automaton. In *Proceedings of the International Symposium on Theory of Machines and Computations*. Z. Kohavi and A. Paz Eds., Academic Press, 189–196.

Hopcroft, J. E. and Ullman, J. D. 1979. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley Longman, Boston, MA.

Jones, N. D. 1975. Space-bounded reducibility among combinatorial problems. *J. Comput. Syst. Sci. 11*, 1, 68–85.

Kim, M., Viswanathan, M., Ben-Abdallah, H., Kannan, S., Lee, I., and Sokolsky, O. 1999. Formally specified monitoring of temporal properties. In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*. IEEE Computer Society, 114–122.

Koymans, R. 1990. Specifying real-time properties with metric temporal logic. *Real-Time Syst. 2*, 4, 255–299.

Lamport, L. 1977. Proving the correctness of multiprocess programs. *IEEE Trans. Softw. Engin. 3*, 2, 125–143.

Ligatti, J. and Reddy, S. 2010. A theory of runtime enforcement, with results. In *Proceedings of the 15th European Symposium on Research in Computer Security*. Lecture Notes in Computer Science, vol. 6345. Springer, 87–100.

Ligatti, J., Bauer, L., and Walker, D. 2005. Edit automata: Enforcement mechanisms for run-time security policies. *Int. J. Inf. Secur. 4*, 1–2, 2–16.

Ligatti, J., Bauer, L., and Walker, D. 2009. Run-time enforcement of nonsafety policies. *ACM Trans. Inf. Syst. Secur. 12*, 3.

McNaughton, R. and Papert, S. 1971. *Counter-Free Automata*. Research Monograph Series, vol. 65, The MIT Press, Cambridge, MA.

Paul, M., Siegert, H. J., Alford, M. W., Ansart, J. P., Hommel, G., Lamport, L., Liskov, B., Mullery, G. P., and Schneider, F. B. 1985. *Distributed Systems: Methods and Tools for Specification: An Advanced Course*. Lecture Notes Computer Science, vol. 190, Springer.

Pnueli, A. 1977. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 46–57.

Pretschner, A., Hilty, M., and Basin, D. 2006. Distributed usage control. *Comm. ACM 49*, 9, 39–44.

Rabin, M. O. and Scott, D. 1959. Finite automata and their decision problems. *IBM J. Res. Dev. 3*, 2, 114–125.

Ramadge, P. J. and Wonham, W. M. 1987. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim. 25*, 1, 206–230.

Schneider, F. B. 2000. Enforceable security policies. *ACM Trans. Inf. Syst. Secur. 3*, 1, 30–50.

Sistla, A. P. and Clarke, E. M. 1985. The complexity of propositional linear temporal logic. *J. ACM 32*, 3, 733–749.

Talhi, C., Tawbi, N., and Debbabi, M. 2008. Execution monitoring enforcement under memory-limitation constraints. *Inf. Comput. 206*, 2–4, 158–184.

Vardi, M. Y. 1995. An automata-theoretic approach to linear temporal logic. In *Proceedings of the 8th Banff Higher Order Workshop on Logics for Concurrency: Structure Versus Automata*. Lecture Notes Computer Science, vol. 1043, Springer, 238–266.

Vardi, M. Y. 2007. Automata-theoretic model checking revisited. In *Proceedings of the 8th International Conference on Verification, Model Checking, and Abstract Interpretation*. Lecture Notes in Computer Science, vol. 4349, Springer, 137–150.

Vardi, M. Y. and Wolper, P. 1994. Reasoning about infinite computations. *Inf. Comput. 115*, 1, 1–37.

Viswanathan, M. 2000. Foundations for the run-time analysis of software systems. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA.

Received June 2012; revised December 2012; accepted February 2013