Access Control By Tracking Shallow Execution History

Philip W. L. Fong

Technical Report CS-2003-9
November 4, 2003

Department of Computer Science
University of Regina
Regina, Saskatchewan, S4S 0A2
Canada

ISBN 0-7731-0458-5

Access Control By Tracking Shallow Execution History

Philip W. L. Fong
Department of Computer Science
University of Regina
Regina, Saskatchewan, Canada S4S 0A2
pwlfong@cs.uregina.ca

November 4, 2003

Abstract

Software execution environments like operating systems, mobile code platforms and scriptable applications must protect themselves against potential demages caused by malicious code. Monitoring the execution history of the latter provides an effective means for controlling the access pattern of system services. Several authors have recently proposed increasingly general automata models for characterizing various classes of security policies enforceable by execution monitoring. An open question raised by Bauer, Ligatti and Walker is whether one can further classify the space of security policies by *constraining* the capabilities of the execution monitor. This paper presents a novel information-based approach to address the research problem. Specifically, security policies are characterized by the information consumed by an enforcing execution monitor.

By restricting the execution monitor to track only a *shallow history* of previously granted access events, a precise characterization of a class of security policies enforceable by restricted access of information is identified. Although provably less expressive than the general class of policies enforceable by execution monitoring, this class does contain naturally occurring policies including Chinese Wall policy, low-water-mark policy, one-out-of-k authorization, assured pipelines, etc. Encouraged by this success, the technique is generalized to produce a lattice of policy classes. Within the lattice, policy classes are ordered by the information required for enforcing member policies. Such a fine-grained policy classification lays the semantic foundation for future studies on special-purpose policy languages.

1 Introduction

Software execution environments like operating systems, mobile code platforms and scriptable applications must protect themselves against potential demages caused by malicious code. Monitoring the execution history of the latter provides an effective means for controlling the access pattern of system services. Execution monitoring (EM) can be implemented either by interposing a reference monitor between system service entry points and the code providing the services [12, 3, 10], or by injecting monitoring code into client programs at load time [11, 22, 23, 26, 27, 17, 18]. Schneider [20] proposed an automata-theoretic characterization of security policies enforceable by EM. Specifically, an EM-enforceable policy prescribes access event sequences recognized by a Büchi automaton [1]. It is observed that Büchi-like security automata can only enforce safety properties, but not liveness properties. Subsequently, Bauer, Ligatti and Walker [2, 15] proposed a characterization of increasingly general classes of security policies enforceable by insertion, suppression and editing automata. These policy classes are provably more expressive than EM-enforceable policies.

An open question raised by the work of Bauer *et al* is whether or not one can further classify the space of EM-enforceable policies by *constraining* the capability of the execution monitor. Not only does such a fine-grained classification help us understand the inherent complexity of security policies, it also has a number of practical engineering ramifications. In an environment in which users invest a high degree of trust on the formulation of security policies, and in which the complexity of the security policies increases with the complexity of the software environment, one has to face the reality that policy engineering shares many challenges once considered unique to software engineering. Characterizing security policies as members of well-understood policy classes can facilitate policy engineering in the following ways:

- Special-purpose policy languages can be designed for a policy class to facilitate the correct formulation of member policies.
- Efficient decision procedures may exist for verifying the correctness of policies belonging to policy classes with rich internal structure.
- Some policy classes may exhibit structural properties that render their member policies decomposable into more manageable policy components. Discovery of such structural properties enables the composition of complex policies from reusable components.

This paper presents a novel information-based approach to address the research problem raised by Bauer *et al*. Instead of following their proposal, and classify security policies by constraining *computational resources* available to the execution monitor, this work classifies EM-enforceable security policies by the kind of *information* that needs to be tracked by the execution monitor. Such a fine-grained policy classification lays the semantic foundation for future studies on special-purpose policy languages.

Consider the Chinese Wall policy [7, 16, 19], a commercial policy for preventing accesses leading to conflict of interests. As observed by Brewer and Nash in their original formulation of the policy, successful enforcement of the Chinese Wall policy only requires the maintenance of a *shallow access history* of previously granted access events. Specifically, the decision on whether an access is to be granted is based solely on the *set*[1] of access events that have

[1] Brewer and Nash used a history matrix to track this set.

already been granted, and not on the actual *sequencing* of such access events. In contrast, a Büchi automaton could potentially have the full history of granted access events at its disposal when an access granting decision is made. This paper presents a characterization of security policies enforceable by tracking only the shallow access history of a system. Security policies in this class are recognizable by *shallow history automata* (SHA), expressiveness of which is provably more restrictive than that of Büchi-like security automata. Surprisingly, it is still possible to express a wide range of well-known and realistic security policies with SHA: Chinese Wall policy [7], low-water-mark policy [5], one-out-of-k authorization [9], assured pipelines [6, 28], etc. This demonstrates the feasibility of defining meaningful policy classes by constraining information accessible to execution monitors.

Motivated by the above success, the state abstraction techniques applied to characterize shallow access history tracking is generalized. A lattice of security policy classes is obtained as a result. At the top of the lattice is the class of policies enforceable by tracking the full history of access events, as in the case of Büchi-like security automata. As one moves down the lattice, one finds classes of policies that are enforceable by consuming less and less information, with SHA-enforceable policies somewhere in the middle, and memoryless policies at the bottom. This work has therefore laid the theoretical groundwork for studying special-purpose subclasses of EM-enforceable security policies.

This paper is organized as follows. Related works are reviewed in Section 2. SHA are defined in Section 3 to provide an information-based characterization of security policies enforceable by tracking shallow access history. A number of naturally occurring security policies are shown to be SHA-enforceable in Section 4. The SHA configuaration is generalized in Section 5 to yield a lattice of policy classes. Discussion can be found in Section 6. Section 7 concludes the paper.

2 Related Works

Schneider [20] pioneered the characterization of security policies enforceable by execution monitoring (EM). Specifically, an EM-enforceable policy prescribes access event sequences recognized by a Büchi automaton [1]. It is observed that Büchi-like security automata can only enforce safety properties, but not liveness properties. Viswanathan [24] points out that any reasonable characterization on execution monitoring must involve a computability constraint. Subsequently, Bauer, Ligatti and Walker [2, 15] proposed a characterization of increasingly general classes of security policies enforceable by insertion, suppression and editing automata, while Hamlen, Morrisett and Schneider [13] offers a characterization of security policies enforceable by code rewriting. These policy classes are provably more expressive than EM-enforceable policies. This work is the first one to provide a fine-grained, information-based characterization of subclasses of EM-enforceable policies.

3 Access Control By Shallow History Tracking

This section introduces shallow history automata, the definition of which provides an information-based characterization of security policies enforceable by tracking shallow access history. The class of security policies expressible by such automata is proven to be a proper subset of the general class of EM-enforceable security policies, thereby confirming the claim that subclasses of of EM-enforceable policies can be defined through the restriction of information accessible to the execution monitor. To fix thoughts, the notion of EM-enforceable policies and its characterization via security automata are provided in Section 3.1 and 3.2 respectively. Shallow history automata are then discussed in Section 3.3.

3.1 EM-Enforceable Security Policies

Let Σ be a finite or countably infinite set of *access events*. A *policy* is a set $P \subseteq \Sigma^*$ of finite sequences of access events. An *EM-enforceable policy*[2] is a prefix-closed policy, that is, a policy P satisfying the following condition:

$$\forall u \in \Sigma^* \ : \ u \notin P \Rightarrow (\forall v \in \Sigma^* \ : \ uv \notin P)$$

Let *prefix*(w) be the set of all prefixes of w, including ϵ and w itself. That is, $\mathit{prefix}(w) = \{u \in \Sigma^* \mid \exists v \in \Sigma^* \text{ s.t. } uv = w\}$. It is easy to see that the following is an equivalent characterization of prefix-closed policy:

$$\forall w \in \Sigma^* \ : \ w \in P \Rightarrow \mathit{prefix}(w) \subseteq P \tag{1}$$

In the following, we consider only EM-enforceable policies.

3.2 Security Automata

A variant of Büchi automata is defined here. A *security automaton (SA)* is a quadruple $\langle \Sigma, Q, q_0, \delta \rangle$, where

- Σ is a finite or countably infinite set of access events,
- Q is a finite or countably infinite set of *automaton states*,
- $q_0 \in Q$ is an *initial state*,
- $\delta : Q \times \Sigma \to Q$ is a (possibly partial) *transition function*[3].

[2] For the purpose of this work, the definition of an EM-enforceable policy as adopted here is different from the one used by Schneider [20] in the following ways: (1) Schneider differentiates between general security policies and a special class of policies that he calls security properties. Only those policies that are properties are considered in this paper. (2) Schneider considers infinite sequences of access events. Following the practice of Bauer *et al* [2], only finite sequences of access events are considered in this paper.

[3] The formulation of security automata as given here differs from that of Bauer *et al*: As our focus is information rather than resource constraints, no tractability restriction is imposed on the transition function. See, however, [24] for the need of such constraints in the general cases.

The notion of acceptance for a SA is different from that for a regular finite state machine, in which a final state is explicitly identified. An access event sequence is accepted by a SA if a transition is defined for every event in the sequence. The notion is formalized as follows. Given a SA $M = \langle \Sigma, Q, q_0, \delta \rangle$, the following notations are defined for $q, q' \in Q$, $a \in \Sigma$ and $w \in \Sigma^*$:

$$
\begin{aligned}
& q \xrightarrow{a}_M q' \quad \text{if } \delta(q, a) = q' \\
& q \xrightarrow{\epsilon}_M q \\
& q \xrightarrow{wa}_M q' \quad \text{if there exists } q'' \in Q \text{ s.t. } q \xrightarrow{w}_M q'' \text{ and } q'' \xrightarrow{a}_M q'
\end{aligned}
$$

We say that M accepts an access event sequence w if $q_o \xrightarrow{w}_M q$ for some $q \in Q$. The policy $\mathcal{P}(M)$ *recognized* by the SA M is then defined as the set of all sequences accepted by M:

$$\{w \in \Sigma^* \mid \exists q \in Q \ : \ q_0 \xrightarrow{w}_M q\}.$$

It is easy to see that such a set is always prefix-closed, that is, $\mathcal{P}(M)$ satisfies condition (1). Conversely, given any prefix-closed policy P, there is a SA M so that $P = \mathcal{P}(M)$. To see this, consider the SA $\langle \Sigma, \Sigma^*, \epsilon, \delta_P \rangle$, where $\delta_P(w, a)$ is defined to be wa if $w, wa \in P$, and is otherwise undefined. Such a SA recognizes P. Consequently, the class of EM-enforceable policies coincides with the class of policies recognized by a SA. We call the above SA constructed to recognize P the *canonical SA* for policy P, and denote it by SA(P).

Intuitively, the state of a SA represents the information that is tracked by the corresponding execution monitor. It represents the internal data structure maintained by the execution monitor across subsequent access granting decisions. The image of the transition function captures the updating procedure of the internal data structure, while domain of the transition function captures the logic of access granting decisions. Notice that the canonical SA tracks the full history of previously granted access events.

3.3 Shallow History Automata

Let $\mathcal{F}(S)$ be the set of all *finite* subsets of a set S. A *shallow access history* (or simply *shallow history*) is a finite subset of Σ, that is, a member of $\mathcal{F}(\Sigma)$. Our goal is to define a class of automata that track only the shallow history of previously granted access events.

A *shallow history automaton (SHA)* is a SA of the form $\langle \Sigma, \mathcal{F}(\Sigma), H_0, \delta \rangle$, where

- Σ is a finite or countably infinite set of access events,
- The state set $\mathcal{F}(\Sigma)$ contains all possible shallow access histories.
- $H_0 \in \mathcal{F}(\Sigma)$ is an *initial access history*, and
- The transition function δ is such that $\delta(H, a) = H \cup \{a\}$ if δ is defined at $\langle H, a \rangle$.

Intuitively, a SHA tracks only a shallow access history, and bases its access granting decisions solely on this information. Sequencing information about previously granted access events are not retained for subsequent access granting decisions.

Notice that the image of δ is always $H \cup \{a\}$ if it is defined at $\langle H, a\rangle$. Consequently, defining δ amounts to specifying its domain as a subset of $\mathcal{F}(\Sigma) \times \Sigma$. That is, a SHA transition function is uniquely specified by listing all the points at which it is defined.

As expected, SHA is strictly less expressive than SA:

Theorem 1 *There is a SA M so that no SHA N is such that $\mathcal{P}(M) = \mathcal{P}(N)$.*

Proof: Let $\Sigma = \{a, b, c, d\}$. Consider the policy $P = \mathit{prefix}(abcd) \cup \mathit{prefix}(badc)$. The policy P is prefix-closed by construction, and is thus recognizable by its canonical SA. Suppose that P is recognized by a SHA M. Let H_0 be the initial state of M. The following transitions are valid:

$$H_0 \xrightarrow{a}_M \{a\} \cup H_0 \xrightarrow{b}_M \{a, b\} \cup H_0 \xrightarrow{c}_M \{a, b, c\} \cup H_0 \xrightarrow{d}_M \{a, b, c, d\} \cup H_0$$
$$H_0 \xrightarrow{b}_M \{b\} \cup H_0 \xrightarrow{a}_M \{a, b\} \cup H_0 \xrightarrow{d}_M \{a, b, d\} \cup H_0 \xrightarrow{c}_M \{a, b, c, d\} \cup H_0$$

However, with the above transitions, M also accepts *abdc* and *bacd*:

$$H_0 \xrightarrow{a}_M \{a\} \cup H_0 \xrightarrow{b}_M \{a, b\} \cup H_0 \xrightarrow{d}_M \{a, b, d\} \cup H_0 \xrightarrow{c}_M \{a, b, c, d\} \cup H_0$$
$$H_0 \xrightarrow{b}_M \{b\} \cup H_0 \xrightarrow{a}_M \{a, b\} \cup H_0 \xrightarrow{c}_M \{a, b, c\} \cup H_0 \xrightarrow{d}_M \{a, b, c, d\} \cup H_0$$

By way of contradiction, P is not SHA-enforceable.

3.4 Summary

A subclass of security automata, SHA, is successfully defined to capture the notion of execution monitoring by tracking only shallow access history. The separation result in Theorem 1 confirms that SHA are strictly less expressive than general SA.

4 Security Policies Enforceable By Tracking Shallow History

Defining artificial subclasses of security policies is pointless if the classes do not correspond to security policies found in real-life applications. This section demonstrates that the class of SHA-enforceable policies does include nontrivial security policies such as the Chinese Wall policy (Section 4.1), the low-water-mark policy (Section 4.2), one-out-of-k authorization (Section 4.3), and assured pipelines (Section 4.4). The goal is to show that the defintion of SHA yields a naturally occurring class of security policies.

4.1 Chinese Wall Policy

Set in a commercial context, in which a consultant shall not advise clients whom he or she has insider knowledge of a competitor, the Chinese Wall security policy is designed to avoid any conflict of interest that may arise due to the unchecked flow of information across datasets belonging to competing parties. Let O be a set of data objects, S a set of subjects, G a set of company datasets, and T a set of conflict of interest classes. Associated with each data object $o \in O$ is a permanent label $\mathit{group}[o] \in G$ describing the company dataset in which o belong. Similarly, a permanent label $\mathit{type}[g] \in T$ is assigned to each company dataset $g \in G$; the label describes the conflict of interest class in which a company dataset belongs. A subject s may access a data object o only if one of the following holds:

- Subject s has already accessed another object o' belonging to the same company dataset of o, that is, $\mathit{group}[o] = \mathit{group}[o']$.
- Every object o' that subject s has accessed so far belongs to a company dataset whose conflict of interest class is different from that of the company dataset in which o belongs, that is, $\mathit{type}[\mathit{group}[o]] \neq \mathit{type}[\mathit{group}[o']]$.

A reference monitor enforcing the Chinese Wall policy may do so by keeping track of the set of objects previously accessed by each subject. The policy is therefore SHA-enforceable. This can be demonstrated formally by the following construction. Let the set of access events be $\Sigma = S \times O$, so that a pair $\langle s, o\rangle$ refers to the event of subject s accessing object o. Define SHA $N = \langle \Sigma, \mathcal{F}(\Sigma), \emptyset, \delta_{\mathit{group},\mathit{type}}\rangle$, where the SHA transition function $\delta_{\mathit{group},\mathit{type}}$ is defined at $\langle H, \langle s, o\rangle\rangle$ whenever the following holds:

- there exists $\langle s, o'\rangle \in H$ s.t. $\mathit{group}[o] = \mathit{group}[o']$, or
- for all $\langle s, o'\rangle \in H$, $\mathit{type}[\mathit{group}[o]] \neq \mathit{type}[\mathit{group}[o']]$

By construction, the SHA N enforces the Chinese Wall policy.

With refinement to the above construction, it is also possible to show that the variation of Chinese Wall policy as proposed by Lin [16] is SHA-enforceable.

4.2 Low-Water-Mark Policy (for Subjects)

The low-water-mark policy is one of the three lattice-based integrity policies proposed by Biba [5]. Defined in the Biba's security model are a set S of subjects, a set O of objects and a set L of integrity levels, which are partially ordered by a binary relation $\leq$. At any point of time, a label $l[s] \in L$ is assigned to every subject $s \in S$, and likewise $l[o] \in L$ is assigned to every object $o \in O$. Intuitively, the label $l[\cdot]$ describes the *trustworthiness* of a subject or an object. Three kinds of access events are defined: $\mathbf{write}(s, o)$, $\mathbf{read}(s, o)$ and $\mathbf{exec}(s, s')$. The low-water-mark policy grants accesses according to the following rules:

1. $\mathbf{read}(s, o)$ is always permitted, with the side effect of $l[s] \leftarrow l[s] \wedge l[o]$, where $\wedge$ denotes the glb between integrity levels.
2. $\mathbf{write}(s, o)$ is permitted if $l[o] \leq l[s]$.
3. $\mathbf{exec}(s, s')$ is permitted if $l[s'] \leq l[s]$.

To construct an SHA for enforcing the low-water-mark policy, notice that the label of a subject is a function of the objects it has read, while that of an objct is permanent. Let Σ be the set of all access events. Given an initial label assignment $l[\cdot]$, define a SHA $N_l = \langle \Sigma, \mathcal{F}(\Sigma), H_l, \delta_l \rangle$, where the initial history H_l is further specified as follows:

$$H_l = \{\mathbf{read}(s, o) \mid s \in S, o \in O, l[s] = l[o]\},$$

and the SHA transition function δ_l is defined at exactly the following points:

- $\langle H, \mathbf{read}(s, o) \rangle$ for all $H \in \mathcal{F}(\Sigma)$, $s \in S$ and $o \in O$, and
- $\langle H, \mathbf{write}(s, o) \rangle$ for all $H \in \mathcal{F}(\Sigma)$, $s \in S$ and $o \in O$ so that for all $\mathbf{read}(s, o') \in H$, $l[o] \leq l[o']$, and
- $\langle H, \mathbf{exec}(s, s') \rangle$ for all $H \in \mathcal{F}(\Sigma)$, $s, s' \in S$ so that for all $\mathbf{read}(s, o) \in H$, there is a $\mathbf{read}(s', o') \in H$, $l[o'] \leq l[o]$.

By construction, the SHA N_l enforces the low-water-mark policy.

4.3 One-Out-Of-k Authorization

One-out-of-k authorization [9] classifies applications into equivalence classes based on the kind of access rights required to complete tasks. For example, the following application classification is given in [9]:

- A **browser** is a program that connects to remote sites, creates temporary local files in a user-specified directory, reads files it has created, and displays them to the user.
- An **editor** is a program that creates local files in a user-specified directory, reads/modifies files it has created, and interacts with the user.
- A **shell** is a program that interacts with with the user and creates subprocesses.

The goal is to dynamically classify an executing program into one of the application classes based on the access requests it makes. Once classified into an application class, a program is only allowed to exercise access rights granted to the class.

The one-out-of-k constraint can easily be enforced by a SHA. Let Σ be the set of all possible accesses made by an application. An application class i is completely characterized by the set $C_i \subseteq \Sigma$ of permitted accesses. Let $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ be the set of all application classes. Define SHA $N_{\mathcal{C}} = \langle \Sigma, \mathcal{F}(\Sigma), \emptyset, \delta_{\mathcal{C}} \rangle$ s.t. the SHA transition function $\delta_{\mathcal{C}}$ is defined at $\langle H, a \rangle$ iff $H \cup \{a\} \subseteq C_i$ for some $C_i \in \mathcal{C}$. By construction, $N_{\mathcal{C}}$ enforces one-out-of-k authorization.

4.4 Assured Pipelines

Assured pipelines [6, 28] arise in the context of ensuring data integrity when data objects are processed by (usually linear) pipelines of transformation procedures. Let O be a set of data objects. Let S be a set of transformation procedure. Assume that S contains a distinguished member **create**. Define the set of access events to be $\Sigma = S \times O$, where the member $\langle s, o \rangle$ denotes the application of transformation procedure s to data object o. An assured pipeline policy is specified as an *enabling relation* $e \subseteq S \times S$, with the following restrictions:

1. No circularity: the binary relation defines a directed acyclic graph (DAG).
2. No pair of the form $\langle s, \textbf{create} \rangle$ may be included: **create** is the sole source node of the acyclic graph.

The intention is that $\langle s, s' \rangle \in e$ means that access $\langle s', o \rangle$ is granted only if $\langle s, o \rangle$ has already been executed. Also, each event $\langle s, o \rangle$ may occur at most once.

An assured pipeline policy can be enforced by a SHA $N_e = \langle \Sigma, \mathcal{F}(\Sigma), \emptyset, \delta_e \rangle$, where the SHA transition function δ_e is defined at exactly the points $\langle H, \langle s, o \rangle \rangle$ satisfying the following:

- $\langle s, o \rangle \notin H$, and
- either one of the following holds:
  - $s = \textbf{create}$, or
  - for some $s' \in S$, all of the following hold:
    - $\langle s', o \rangle \in H$, and
    - $\langle s', s \rangle \in e$, and
    - there is no $s'' \in S$ s.t. $\langle s'', o \rangle \in H$ and $\langle s', s'' \rangle \in e$.

By construction N_e enforces the assured pipeline policy specified by enabling relation e.

4.5 Summary

Four naturally occurring security policies have been shown to be SHA-enforceable, thereby demonstrating that information restriction could indeed be employed to classify real-life security policies.

5 Obtaining Policy Classes By Abstraction

Given that it is possible to define a meaningful subclass of EM-enforceable security policies, the next question is: can the above technique be generalized to obtain other subclasses of EM-enforceable policies? Section 5.1 offers an affirmative answer to the question, with the help of concepts borrowed from automata theory [8, 14]. The structural properties of EM-enforceable policy classes are then studied in Section 5.2 and 5.3.

5.1 Abstraction By Homomorphism

Given an arbitrary restriction on the information accessible to an execution monitor, the following procedure can be systematically carried out to obtain an automata-theoretic characterization of the policies thus enforceable:

1. An information restriction constraint is specified as a set $\mathcal{A}$ of *abstract states*, which represents the kind of information that the execution monitor is allowed to track (e.g., shallow histories as finite subsets of access events).

2. An interpretation of the abstract states in $\mathcal{A}$ is defined, so they refer to states of the canonical SA in a consistent manner (e.g., each shallow history documents the set of previously granted access events in an execution sequence).

3. Under the above interpretation, a subclass of SA is defined so that member automata behave consistently according to the interpretation (e.g., shallow history automata).

This plan is executed as follows.

Let $\mathcal{A}$ be a finite or countably infinite set of abstract states. A function $\alpha : \Sigma^* \to \mathcal{A}$ is an *abstraction* if it satisfies the following *compatibility property*:

$$\alpha(w) = \alpha(w') \Rightarrow \alpha(wa) = \alpha(w'a)$$

An abstraction mashes distinct states of the canonical SA into a single abstract state, rendering them indistinguishable. The compatibility property guarantees that the loss of information does not introduce confusion.

The *homomorphic image* $\mathrm{SA}_\alpha(P)$ of the canonical security automaton $\mathrm{SA}(P)$ induced by abstraction α is the security automaton $\langle \Sigma, \mathcal{A}, \alpha(\epsilon), \delta_{P/\alpha} \rangle$, where

$$\delta_{P/\alpha}(\alpha(w), a) = \alpha(wa) \quad \text{if} \quad \delta_P(w, a) = wa$$

and δ_P is the transition function of $\mathrm{SA}(P)$. Since α satisfies the compatibility property, $\delta_{P/\alpha}$ is a well-defined (partial) function.

Notice that not every policy P is recognized by $\mathrm{SA}_\alpha(P)$ — in general, $P \subseteq \mathcal{P}(\mathrm{SA}_\alpha(P))$ (see Corollary 6). Those policies P recognizable by $\mathrm{SA}_\alpha(P)$ are the policies that can be enforced by consuming only information left behind by the abstraction α. A policy P is said to be *enforceable* by abstraction α iff P is recognized by the security automaton $\mathrm{SA}_\alpha(P)$, that is, iff $P = \mathcal{P}(\mathrm{SA}_\alpha(P))$. Fixing the set Σ of access events, the class of all policies enforceable by abstraction α is denoted by EM_α.

In summary, given any abstraction α, it is now possible to define exactly the class of security policies enforceable by tracking information permitted by the abstraction.

5.2 Abstractions As Congruence Relations

It turns out that the notion of information abstraction is very robust: it can be defined independent of the choice of abstract states.

Every abstraction α induces an equivalence relation $\equiv_\alpha$ as follows:

$$\text{For all } w, w' \in \Sigma^*,\ w \equiv_\alpha w' \Leftrightarrow \alpha(w) = \alpha(w').$$

It can be shown that such an equivalence relation satisfies the following *substitution property*:

$$\text{For all } w, w' \in \Sigma^* \text{ and } a \in \Sigma,\ w \equiv w' \Rightarrow wa \equiv w'a \tag{2}$$

Conversely, given any equivalence relation $\equiv$ over Σ^* that satisfies the the substitution property, it can be shown that the mapping $\alpha_\equiv : \Sigma^* \to \Sigma^*/\equiv$ defined as follows:

$$\alpha_\equiv(w) = [w]_\equiv.$$

is in fact an abstraction. That is, $\alpha_\equiv$ satisfies the compatibility property. Consequently, the notion of information abstraction can be characterized either by an abstract state mapping satisfying the compatibility property, or by a partition of the access sequence space with an equivalence relation satisfying the substitution property. The latter characterization has a clear advantage over the former — it is independent of the choice of abstract states. In fact, different abstraction mappings may induce the same equivalence relation, making the latter a better means of capturing the essence of the notion of information abstraction.

Let us call an equivalence relation satisfying the substitution property a *congruence relation*. We write $\mathrm{SA}_\equiv(P)$ as a shorthand for $\mathrm{SA}_{\alpha_\equiv}(P)$. A policy P is *enforceable* by a congruence relation $\equiv$ if P is recognized by $\mathrm{SA}_\equiv(P)$. We also write $\mathrm{EM}_\equiv$ as a shorthand for $\mathrm{EM}_{\alpha_\equiv}$.

Defining abstractions in terms of congruence relations gives us a simple way of comparing their information complexities. Intuitively, more policies are enforceable by $\equiv_1$ than by $\equiv_2$ iff $\equiv_1$ is more differentiating than $\equiv_2$.

Theorem 2 *Let $\equiv_1$ and $\equiv_2$ be two congruence relations over Σ^*. If $\equiv_1 \subseteq \equiv_2$, then $EM_{\equiv_2} \subseteq EM_{\equiv_1}$. Moreover, the latter containment is proper if the former is proper.*

Consult the appendix for a proof of this theorem.

5.3 Lattice of Policy Classes

Define the binary operator $\sqcup$, the *join*, on the space of all congruence relations over Σ^*:

$$\equiv_1 \sqcup \equiv_2 = \equiv_1 \cap \equiv_2$$

That is, the join operator combines the differentiating power of its operands. Similarly, define the binary operator $\sqcap$, the *meet*, as follows:

$$\equiv_1 \sqcap \equiv_2 = (\equiv_1 \cup \equiv_2)^*$$

that is, the transitive closure of the union of the two operand congruence relations. In other words, a meet captures the common differentiating power of the two operands. The following theorem is a well-known result in automata theory:

Theorem 3 *The binary operators $\sqcup$ and $\sqcap$ define a lattice on the space of all congruence relations over Σ^*. The lattice has both a top element $\equiv_\top$ and a bottom element $\equiv_\bot$.*

Proof: It is mechanical to check that $\sqcup$ and $\sqcap$ produce congruence relations and define a lattice. The top element $\equiv_\top$ is the congruence relation $\emptyset$ in which all members of Σ^* are distinct, while the bottom element $\equiv_\bot$ is the congruence relation $\Sigma^* \times \Sigma^*$, in which all members of Σ^* are equivalent.

Intuitively, the top element $\equiv_\top$ induces the class $\text{EM}_{\equiv_\top}$ of all EM-enforceable security policies. In contrast, the bottom element $\equiv_\bot$ induces the class $\text{EM}_{\equiv_\bot}$ of security policies enforceable by SA with only one state. Such SA are *memoryless*: they do not track historical information at all, and grant access in a static manner. The rest of the congruence relations are ordered in decreasing differentiating power as we move down the lattice. As a consequence of Theorem 2, this lattice of congruence relations induces an isomorphic lattice of EM-enforceable policy classes ordered by class containment.

5.4 Summary

This section generalizes the construction of SHA to obtain a lattice of policy classes. The notion of an information-based characterization of security policies is shown to be more general than shallow access history tracking. New policy classes may be characterized through the specification of either an abstraction mapping or a congruence relation. Theorems 2 and 3 also confirm the following intuition: the more information the execution monitor is allowed to track, the more security policies it is able to enforce.

6 Discussion

The notion of information complexity is in fact very sensitive to the choice of access events. We have seen that, with a *fixed* set Σ of access events, there are EM-enforceable policies that SHA cannot enforce. Now, if we *instrument* the access events so that an event is a pair of the form $\langle a, i \rangle$, in which i is the time index of access a in the event sequence, then a SHA can enforce every policy that a SA could enforce without the instrumentation. Consequently, one must fix the set of access events to get a fair comparison of the expressiveness of different abstractions. Also, one should attempt to pick the most "*natural*" formulation of access events, one that conveys the least information to the security automata.

To simplify discussion, all event sequences from Σ^* are treated as being equally plausible. In reality, the execution environment may generate only sequences belonging to a subset of Σ^*. For example, in the case of assured pipelines, the requirement that every access event occurs only once is in fact a part of the behavior characteristics of the execution environment, and not a part of the security policy per se. To focus on the essence of the security policy rather than the idiosyncrasy of the execution environment, one could have defined a security automaton alternatively as a 5-tuple $\langle \Sigma, \Psi, Q, q_0, \delta \rangle$, in which the newly introduced second component $\Psi \subseteq \Sigma^*$ is the set of all event sequences that could be generated by the execution environment. For an elaboration of this treatment, consult the works of Schneider [20] and Bauer *et al* [2].

A number of theoretical apparatus, including that of automata homomorphism, congruence relations, compatibility property, substitution property, and lattices of congruence relations, are all borrowed from Büchi's approach to automata theory [8, 14]. Instead of using these tools to study the minimization and decomposition of individual automata, they are applied to obtain a novel, information-based characterization of security policies:

1. Automata homomorphisms and their counterparts, congruence relations, are used for defining automata classes with restricted access to history information.

2. Each automata class is then used to characterize a class of security policies enforceable with the corresponding information constraint.

3. The well-known lattice of congruence relations is then shown to induce an isomorphic lattice of policy classes.

An abstraction is basically a syntactic means for defining the data structure tracked by the execution monitor. The corresponding congruence is the semantics of this syntax. Seen in this light, the present theoretical framework offers a very precise semantic infrastructure for defining syntactic constructs that represent the data structure tracked by an execution monitor. A future direction is to employ the current framework to define special-purpose policy languages for mobile code systems. Specifically, there is a close connection between the current framework and the model checking of control flow properties [4, 25, 21]. By bringing in automata theory into the study, it is hoped that automata decomposition results from the theory could faciliate the development of reusable policy components. These are lines of research the author would like to pursue in the future.

7 Conclusion

A novel approach is proposed to address the open question raised by Bauer *et al.* The space of EM-enforceable security policies is classified according to the information consumed by the execution monitor. The feasibility of this approach is demonstrated by the characterization of security policies enforceable by tracking shallow execution history. Although the class is provably less expressive than the general class of EM-enforceable policies, it nevertheless contains a number of naturally occurring security policies. Generalization of the technique allows one to define a complete lattice of security policy classes, in which member classes are ordered by the amount of information that must be tracked by an enforcing execution monitor.

References

[1] Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2(3):117–126, 1987.

[2] Lujo Bauer, Jarred Ligatti, and David Walker. More enforceable security policies. In *Proceedings of the Workshop on Foundations of Computer Security (FCS'02)*, Copenhagen, Denmark, July 2002.

[3] Massimo Bernaschi, Emanuele Gabrielli, and Luigi V. Mancini. REMUS: A security-enhanced operating system. *ACM Transactions on Information and System Security*, 5(1):36–61, February 2002.

[4] Frederic Besson, Thomas Jensen, and Daniel Le Metayer. Model checking security properties of control flow graphs. *Journal of Computer Security*, 9(3):217–250, 2001.

[5] K. Biba. Integrity considerations for secure computer systems. Technical Report 76–372, U. S. Air Force Electronic Systems Division, 1977.

[6] W. E. Boebert and R. Y. Kain. A practical alternative to hierarchical integrity policies. In *Proceedings of the 8th National Computer Security Conference*, pages 18–27, October 1985.

[7] David F. C. Brewer and Michael J. Nash. The Chinese Wall security policy. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 206–214, Oakland, California, May 1989. Also available at `http://www.gammassl.co.uk/topics/chinesewall.html`.

[8] J. Richard Büchi. *Finite Automata, Their Algebra and Grammars*. Springer-Verlag, 1988.

[9] Guy Edjladi, Anurag Acharya, and Vipin Chaudhary. History-based access control for mobile code. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, San Francisco, CA, USA, 1998.

[10] Anthony Edwards, Trent Jaeger, and Xiaolan Zhang. Runtime verification of authorization hook placement for the Linux security modules framework. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 225–234, Washington, DC, November 2002.

[11] David Evans and Andrew Twyman. Flexible policy-directed code safety. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 32–45, Oakland, California, May 1999.

[12] Timothy Fraser. LOMAC: Low water-mark integrity protection for COTS environments. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, Oakland, California, 2000.

[13] Kevin W. Hamlen, Greg Morrisett, and Fred B. Schneider. Computability classes for enforcement mechanisms. Technical Report TR 2003-1908, Computer Science Department, Cornell University, August 2003.

[14] Michael A. Harrison. *Introduction to Switching and Automata Theory*. McGraw-Hill, 1965.

[15] Jay Ligatti, Lujo Bauer, and David Walker. Edit automata: Enforcement mechanisms for run-time security policies. *International Journal of Information Security*, 2003. To appear.

[16] Tsau Young Lin. Chinese Wall security policy — an aggressive model. In *Proceedings of the Fifth Annual Computer Security Applications Conference*, pages 282–289, December 1989.

[17] Algis Rudys and Dan S. Wallach. Termination in language-based systems. *ACM Transactions on Information and System Security*, 5(2):138–168, May 2002.

[18] Algis Rudys and Dan S. Wallach. Transactional rollback for language-based systems. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'02)*, pages 439–448, Washington, D.C., June 2002.

[19] Ravi S. Sandhu. A lattice interpretation of the Chinese Wall policy. In *Proceedings of the 15th NIST-NCSC National Computer Security Conference*, pages 221–235, Baltimore, MD, October 1992.

[20] Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, February 2000.

[21] R. Sekar, V. N. Venkatakrishnan, Samik Basu, , Sandeep Bhatkar, and Daniel C. DuVarney. Model-carrying code: A practical approach for safe execution of untrusted applications. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, Bolton Landing, New York, October 2003.

[22] Úlfar Erlingsson and Fred B. Schneider. SASI enforcement of security policies: A retrospective. In *Proceedings of the 1999 New Security Paradigms Workshop*, pages 87–95, Caledon Hills, Ontario, Canada, September 1999.

[23] Úlfar Erlingsson and Fred B. Schneider. IRM enforcement of java stack inspection. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 246–255, Berkeley, California, May 2000.

[24] Mahesh Viswanathan. *Foundations for the Run-time Analysis of Software systems*. PhD thesis, University of Pennsylvania, December 2000.

[25] David Wagner and Drew Dean. Intrusion detection via static analysis. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 156–168, Oakland, California, May 2001.

[26] Dan S. Wallach, Andrew W. Appel, and Edward W. Felten. SAFKASI: A security mechanism for language-based systems. *ACM Transactions on Software Engineering and Methodology*, 9(4):341–378, October 2000.

[27] Ian Welch and Robert J. Stroud. Using reflection as a mechanism for enforcing security policies on compiled code. *Journal of Computer Security*, 10(4):399–432, 2002.

[28] W. D. Young, P. A. Telega, and W. E. Boebert. A verified labler for the secure Ada target. In *Proceedings of the 9th National Computer Security Conference*, pages 55–61, September 1986.

Appendix: Proof of Theorem 2

Observation 4 *Let P be a prefix-closed policy, and $\equiv$ be a congruence relation. Then, for any $w \in \Sigma^*$ and $a \in \Sigma$, we have:*

$$w \xrightarrow{a}_{SA(P)} wa \quad \Rightarrow \quad [w]_\equiv \xrightarrow{a}_{SA_\equiv(P)} [wa]_\equiv \tag{3}$$

and

$$[w]_\equiv \xrightarrow{a}_{SA_\equiv(P)} [wa]_\equiv \quad \Rightarrow \quad \exists\, w' \in [w]_\equiv \;:\; w' \xrightarrow{a}_{SA(P)} w'a \tag{4}$$

and also

$$w \xrightarrow{a}_{SA(\mathcal{P}(SA_\equiv(P)))} wa \quad \Rightarrow \quad [w]_\equiv \xrightarrow{a}_{SA_\equiv(P)} [wa]_\equiv \tag{5}$$

Proposition 5 *Let P be a prefix-closed policy, and $\equiv_1$ and $\equiv_2$ congruence relations such that $\equiv_1 \subseteq \equiv_2$. Then*

$$\mathcal{P}(SA_{\equiv_1}(P)) \subseteq \mathcal{P}(SA_{\equiv_2}(P)).$$

Proof: The statement can be easily demonstrated by induction with the help of the following lemma.

Lemma: *Denote $SA_{\equiv_1}(P)$ and $SA_{\equiv_2}(P)$ by M_1 and M_2 respectively. For any $w \in \Sigma^*$ and $a \in \Sigma$, we have:*

$$[w]_{\equiv_1} \xrightarrow{a}_{M_1} [wa]_{\equiv_1} \quad \Rightarrow \quad [w]_{\equiv_2} \xrightarrow{a}_{M_2} [wa]_{\equiv_2}.$$

Proof:

$$\begin{array}{ll}
\quad [w]_{\equiv_1} \xrightarrow{a}_{M_1} [wa]_{\equiv_1} & \\
\Rightarrow w' \xrightarrow{a}_{\mathrm{SA}(P)} w'a & \text{for some } w' \in [w]_{\equiv_1}\text{, by (4)} \\
\Rightarrow [w']_{\equiv_2} \xrightarrow{a}_{M_2} [w'a]_{\equiv_2} & \text{by (3)} \\
\Rightarrow [w]_{\equiv_2} \xrightarrow{a}_{M_2} [wa]_{\equiv_2} & \because \equiv_1 \subseteq \equiv_2 \text{ and } w' \equiv_1 w \text{ and (2)}
\end{array}$$

Corollary 6 *Let P be a prefix-closed policy, and $\equiv$ a congruence relation. Then*

$$P \subseteq \mathcal{P}(SA_{\equiv}(P)).$$

Proof: Let $\equiv_{\top} = \emptyset$ be the congruence relation in which every sequence belong to a distinct equivalence class. Then $P = \mathcal{P}(\mathrm{SA}(P)) = \mathcal{P}(\mathrm{SA}_{\equiv_{\top}}(P))$. The result follows from Proposition 5.

Proposition 7 *Let P_1 and P_2 be prefix-closed policies so that $P_1 \subseteq P_2$, and let $\equiv$ be a congruence relation. Then*

$$\mathcal{P}(SA_{\equiv}(P_1)) \subseteq \mathcal{P}(SA_{\equiv}(P_2)).$$

Proof: The statement can be demonstrated easily by induction with the help of the following lemma.

Lemma: *Denote $SA_{\equiv}(P_1)$ and $SA_{\equiv}(P_2)$ by M_1 and M_2 respectively. Then for $w \in \Sigma^*$ and $a \in \Sigma$, we have*

$$[w]_{\equiv} \xrightarrow{a}_{M_1} [wa]_{\equiv} \quad \Rightarrow \quad [w]_{\equiv} \xrightarrow{a}_{M_2} [wa]_{\equiv}$$

Proof:

$$\begin{array}{lll} & [w]_{\equiv} \xrightarrow{a}_{M_1} [wa]_{\equiv} & \\ \Rightarrow & w' \xrightarrow{a}_{\mathrm{SA}(P_1)} w'a & \text{for some } w' \in [w]_{\equiv}, \text{ by (4)} \\ \Rightarrow & w' \xrightarrow{a}_{\mathrm{SA}(P_2)} w'a & \because P_1 \subseteq P_2 \\ \Rightarrow & [w']_{\equiv} \xrightarrow{a}_{M_2} [w'a]_{\equiv} & \text{by (3)} \\ \Rightarrow & [w]_{\equiv} \xrightarrow{a}_{M_2} [wa]_{\equiv} & \because w \equiv w' \text{ and (2)} \end{array}$$

Proposition 8 *Let P be a prefix-closed policy, and $\equiv_1$ and $\equiv_2$ congruence relations such that $\equiv_1 \subseteq \equiv_2$. Then*

$$\mathcal{P}(SA_{\equiv_2}(\mathcal{P}(SA_{\equiv_1}(P)))) = \mathcal{P}(SA_{\equiv_2}(P)).$$

Proof: By Corollary 6 and Proposition 7, we already have $\mathcal{P}(\mathrm{SA}_{\equiv_2}(P)) \subseteq \mathcal{P}(\mathrm{SA}_{\equiv_2}(\mathcal{P}(\mathrm{SA}_{\equiv_1}(P))))$. It therefore suffices to show that $\mathcal{P}(\mathrm{SA}_{\equiv_2}(\mathcal{P}(\mathrm{SA}_{\equiv_1}(P)))) \subseteq \mathcal{P}(\mathrm{SA}_{\equiv_2}(P))$. The inclusion can be demonstrated easily by induction with the help of the following lemma:

Lemma: *Denote $SA_{\equiv_1}(P)$ and $SA_{\equiv_2}(P)$ by M_1 and M_2 respectively. Denote $SA_{\equiv_2}(\mathcal{P}(M_1))$ by $M_{2,1}$. Then, for any $w \in \Sigma^*$ and $a \in \Sigma$, we have:*

$$[w]_{\equiv_2} \xrightarrow{a}_{M_{2,1}} [wa]_{\equiv_2} \quad \Rightarrow \quad [w]_{\equiv_2} \xrightarrow{a}_{M_2} [wa]_{\equiv_2}.$$

Proof:

$$
\begin{array}{lll}
& [w]_{\equiv_2} \xrightarrow{a}_{M_{2,1}} [wa]_{\equiv_2} & \\
\Rightarrow & w' \xrightarrow{a}_{\mathrm{SA}(\mathcal{P}(M_1))} w'a & \text{for some } w' \equiv_2 w \text{, by (4)} \\
\Rightarrow & [w']_{\equiv_1} \xrightarrow{a}_{M_1} [w'a]_{\equiv_1} & \text{by (5)} \\
\Rightarrow & w'' \xrightarrow{a}_{\mathrm{SA}(P)} w''a & \text{for some } w'' \equiv_1 w' \text{, by (4)} \\
\Rightarrow & [w'']_{\equiv_2} \xrightarrow{a}_{M_2} [w''a]_{\equiv_2} & \text{by (3)} \\
\Rightarrow & [w]_{\equiv_2} \xrightarrow{a}_{M_2} [wa]_{\equiv_2} & \because \equiv_1 \subseteq \equiv_2 \text{ and } w \equiv_2 w' \equiv_1 w'' \text{ and (2)}
\end{array}
$$

Corollary 9 *Let P be a prefix-closed policy and $\equiv$ a congruence relation. Then*

$$\mathcal{P}(SA_{\equiv}(\mathcal{P}(SA_{\equiv}(P)))) = \mathcal{P}(SA_{\equiv}(P)).$$

Therefore, $\mathcal{P}(SA_{\equiv}(P)) \in EM_{\equiv}$.

Proof of Theorem 2

Suppose $\equiv_1 \subseteq \equiv_2$. From Proposition 5 and Corollary 6, we have:

$$P \subseteq \mathcal{P}(\mathrm{SA}_{\equiv_1}(P)) \subseteq \mathcal{P}(\mathrm{SA}_{\equiv_2}(P)).$$

It follows that $P = \mathcal{P}(\mathrm{SA}_{\equiv_2}(P))$ implies $P = \mathcal{P}(\mathrm{SA}_{\equiv_1}(P))$. Therefore, $\mathrm{EM}_{\equiv_2} \subseteq \mathrm{EM}_{\equiv_1}$.

Now, suppose further that $\equiv_1 \subset \equiv_2$. We want to show that there is a policy in $\mathrm{EM}_{\equiv_1}$ that is not in $\mathrm{EM}_{\equiv_2}$.

Let w be a shortest sequence in Σ^* such that $[w]_{\equiv_1} \subset [w]_{\equiv_2}$. Let w' be a shortest sequence in $[w]_{\equiv_2} \backslash [w]_{\equiv_1}$. We then also have $[w']_{\equiv_1} \subset [w']_{\equiv_2}$.

Let $a \in \Sigma$ be an arbitrary access event. Define prefix-closed policy $P = \mathit{prefix}(wa) \cup \mathit{prefix}(w')$. Define also prefix-closed policy $P' = \mathcal{P}(\mathrm{SA}_{\equiv_1}(P))$. We claim that, although P' is obviously a member of $\mathrm{EM}_{\equiv_1}$ (Corollary 9), it does not belong to $\mathrm{EM}_{\equiv_2}$, that is, $P' \neq \mathcal{P}(\mathrm{SA}_{\equiv_2}(P'))$. To demonstrate this, we show that (1) $w'a \notin P'$, but (2) $w'a \in \mathcal{P}(\mathrm{SA}_{\equiv_2}(P'))$.

1. $w'a \notin P'$: If $w'a \in \mathrm{SA}_{\equiv_1}(P)$, then there must be some $u \in P$, so that $u \in [w']_{\equiv_1}$ and $ua \in P$. The following case analysis demonstrates that this is impossible.

   - $u \neq w'$, for $w'a \neq P$.
   - $u \neq wa$, for $waa \neq P$.
   - $u \neq w$, or else $w \equiv_1 w'$, contradicting the definition of w'.
   - $u \notin \mathit{prefix}(w') \setminus \{w'\}$, or else u would be a sequence in $[w]_{\equiv_2} \setminus [w]_{\equiv_1}$ strictly shorter than w', a contradiction.
   - $u \notin \mathit{prefix}(w) \setminus \{w\}$, or else u would be a sequence strictly shorter than w so that $[u]_{\equiv_1} \subset [u]_{\equiv_2}$, again a contradiction.

2. $w'a \in \mathcal{P}(\mathrm{SA}_{\equiv_2}(P'))$: Notice that, as $P \subseteq P'$, it follows from Proposition 7 that $\mathcal{P}(\mathrm{SA}_{\equiv_2}(P)) \subseteq \mathcal{P}(\mathrm{SA}_{\equiv_2}(P'))$. Consequently, it suffices to show $w'a \in \mathcal{P}(\mathrm{SA}_{\equiv_2}(P))$.

   As $w, wa \in P$, we have $[w]_{\equiv_2} \xrightarrow{a}_{\mathrm{SA}_{\equiv_2}(P)} [wa]_{\equiv_2}$. Also, $w' \in P$ implies that $[\epsilon]_{\equiv_2} \xrightarrow{w'}_{\mathrm{SA}_{\equiv_2}(P)} [w']_{\equiv_2}$. Since $w \equiv_2 w'$, $[w]_{\equiv_2}$ and $[w']_{\equiv_2}$ are the same equivalence class, and, by the substitution property, so are $[wa]_{\equiv_2}$ and $[w'a]_{\equiv_2}$. Therefore, $[\epsilon]_{\equiv_2} \xrightarrow{w'}_{\mathrm{SA}_{\equiv_2}(P)} [w']_{\equiv_2} \xrightarrow{a}_{\mathrm{SA}_{\equiv_2}(P)} [w'a]_{\equiv_2}$.