Title: Automatic security policy enforcement in computer systems

Author: K. Adi, L. Hamza, L. Pene

PII: S0167-4048(17)30228-6
DOI: https://doi.org/10.1016/j.cose.2017.10.012
Reference: COSE 1224

To appear in: *Computers & Security*

Received date: 7-5-2017
Revised date: 29-9-2017
Accepted date: 29-10-2017

Please cite this article as: K. Adi, L. Hamza, L. Pene, Automatic security policy enforcement in computer systems, *Computers & Security* (2017), https://doi.org/10.1016/j.cose.2017.10.012.

Automatic Security Policy Enforcement in Computer Systems

K. Adi, L. Hamza, and L. Pene

Computer Security Research Laboratory

Computer Science and Engineering Department

Université du Québec en Outaouais, Québec, Canada.

Vitae

Kamel Adi holds a Master degree in theoretical computer science from Pierre et Marie Curie (Paris VI) University and a Ph. D. degree in computer security from Laval University, Quebec, Canada. He is currently a full professor in the Department of Computer Science and Engineering at the University of Quebec in Outaouais, Canada. Kamel Adi is also the co-director of the Computer Security Research Laboratory at Université du Québec en Outaouais, Canada. His research activities focus on the development and application of formal methods for solving problems related to computer security and computer networks.

Lamia Hamza is a Ph.D. student and assistant teacher at University of Bejaia, Algeria. She received an engineer diploma in computer sciences from the University of Setif, Algeria, and a M.Sc. in networking and distributed systems from the University of Bejaia, Algeria. Lamia joined the Computer Security Research Laboratory during her research internship at Université du Québec en Outaouais, Canada. She continues to collaborate with other team members on topics of common interest. Her current research involves computer security and formal methods.

Liviu Pene received his Master degree in computer science from Université du Québec en Outaouais, Canada. He is currently a Ph.D. student and a member of the Computer Security Research Laboratory team at Université du Québec en Outaouais, Canada. Liviu's most recent research explores subjects related to the verification and enforcement of computer and network security policies through formal methods.

Abstract. This paper proposes a formal framework for automatic security policy enforcement in computer systems. In this approach, systems and their interactions are formally modelled as process algebra expressions with a new dedicated calculus inspired from the ambient calculus. Security policies are specified with the aid of a dedicated modal logic. We demonstrate how, for a given security policy expressed by a logical formula, our calculus allows to verify whether the specification meets the security policy requirements. If it does not, the optimal enforcement for

the system is automatically generated using our enforcement operator. A software prototype has been implemented to show the practical feasibility and the effectiveness of our security policy enforcement framework.

Keywords: computer security, formal methods, process algebra, security policy, policy enforcement, ambient calculus, modal logic

1 Introduction

There are various solutions for securing computer systems, including operating system tools, third party applications, hardware devices, etc. In general, the difficulty of the issue is directly proportional with the complexity of the system. Some aspects of the security posture are straightforward and can be easily addressed by referring to best practices, templates and white papers. However, such documents only provide guidelines for the most common configurations, which are rarely fit for complex and large computer system. Once the requirements have been determined, the resulting policies are translated into available security mechanisms, implemented, tested and certified (or at least they should be). The human factor intervenes in policy definition, implementation and evaluation. It plays a crucial role in the more or less successful protection of computer networks. Although human intervention is necessary in the definition of security policies, its role in their enforcement on computer systems must be minimized and the task should preferably be performed by an automatic process. The final aim is to reduce or even eliminate implementation errors.

Formal methods are well positioned to address such concerns since they can be used to generate enforcement processes that can be proven correct. The scope of the present work is mainly focused on a security policy enforcement method based on the notions of protected boundaries and controlled process movement. The resulting framework allows us to specify systems, express security policies, assess policy compliance and automatically calculate necessary enforcements for non-compliant systems. Given a process P (representing a system) and a formula Φ (corresponding to a desired security policy), then changes (denoted by an enforcement process X) may be required so that the resulting system ($P \upharpoonright X$, read as P enforced by X) satisfies Φ. The concepts and techniques apply to small and large networks alike, regardless of the number of nodes and the complexity of the topology.

The remainder of this paper is organized as follows. Section 2 reviews related research work. Section 3 summarizes our approach. The new calculus we introduce in Section 4 is suitable for

specifying systems and policy enforcements. Section 5 details our dedicated logic for security policy specification. The quotient operator, defined in Section 6, describes our technique for deriving an enforcement from a given security policy and system's specification. The technique is illustrated through the case study depicted in Section 7. Section 8 presents a software prototype implementing the proposed approach. Finally, Section 9 expresses our conclusions and some directions for future work.

2 Related Work

This section reviews some of the closest related research papers and points out some of their shortcomings with respect to our purpose. The articles mentioned here are relevant for the elements of the framework we propose: process calculus, logic, and security policy enforcement.

One of the most successful formalisms used in the related literature to specify computer systems is the ambient calculus, which was first introduced by Cardelli and Gordon in [1]. An ambient is a delimited space that has a name, an interior (containing processes) and an exterior. The calculus captures the notion of mobility by allowing processes to move between different administrative sites. The movement of an ambient process is governed by its capabilities, including the possibility to move inside or outside another ambient.

Several research initiatives have employed the ambient calculus to address the issues related to modeling and validating computer systems. In [2], Adi *et al*. proposed a dedicated ambient-based calculus for the specification of distributed firewall policies. Process equivalence is used to prove the correctness of a local security policy with respect to the global security policy, but it does not ensure the enforcement of security policies. An interesting extension of the ambient calculus was proposed by Ferrari *et al*. [3]. They consider the concept of ambient monitoring and coordination policy through guardians attached to each ambient. The role of a guardian is to monitor the activity of processes and sub-ambients and the interaction with the external environment. Guardians have coordination abilities and they can successfully cooperate for an effective propagation of a policy change across an environment. A control in guardians involves specific entities with their own semantics, which adds to the complexity of the model and makes the verification and enforcement tasks more difficult.

A variety of logical formalisms can be used to specify security policies. In order to simplify the model checking process, the formalism should be related to the system's specification language. In [4, 5] Cardelli and Gordon defined the ambient logic as a modal logic for expressing

properties of processes described by the mobile ambient calculus. It allows expressing properties that hold at particular locations by using spatial modalities. The ambient logic has been further developed by Hirschkoff *et al.* [6] who demonstrate that it is a very expressive formalism and an intensional logic.

The subject of enforcing security policies, and enforcements on programs in particular, has also been tackled by several researchers. Static enforcement can be accomplished through techniques such as model checking [7], type systems [8, 9] and proof-carrying code (PPC) [10]. Schneider [11] initiated the research on runtime-enforceable security policies through security automata simulations. He defined a specific class of enforcement mechanisms, the EM (Execution Monitoring) class. The result was further enhanced by Schneider, Hamlen, *et al*. [12, 13] through the addition of a class enforceable properties by program rewriting. This can be done by transformation of formulas and processes, as demonstrated by Langar *et al* in [14] and by Sui *et al* in [15]. Among other notable research in this space we include the works of Bauer *et al.* [16, 17], Clarkson [18], Basin [19], and Khoury [20].

3 Our Approach

A common observation in the related research literature is that the existing approaches have limited applicability. Although researchers have made great strides in the past few years, most approaches adopt an informal manner for specifying input information of both system and policy representation. This impacts the accuracy of the specification. In practical terms, we need to make sure that none of the network components or system's interactions are missing. Access to network shares, for instance, implies the definition of components (users, shared volumes, communication channels, access control lists, etc.) and interactions (allowed operations such as file retrieval and submission, system response in case of insufficient access permissions, etc.). The absence of the aforementioned components or insufficient granularity in describing interactions would make the specification incomplete. It is therefore important to develop formal methods that allow appropriate representations of computer systems and their behavior. The technique we propose in this paper addresses the issue of policy compliance for computer networks through formal specification and assessment of the system's security configuration. The approach allows the automatic generation of enforcement processes that have the ability to rewrite a system specification to make it satisfy a security policy.

The computer network analyzed is specified with our dedicated calculus, which is designed to easily capture the behaviors of the various network components. The security policy is specified using a dedicated logic. Policy compliance can be verified in terms of system and policy specifications. If a policy change is required, our quotient operator allows us to compute the enforcements for the non-compliant components of the system. Figure 1 provides an outline of our approach. The major steps involved are the following:

– the system is specified using the calculus defined in section 4; the result is a process P which models, at an abstract level, the system;

– the security policy is specified as a formula Φ with the aid of the logic defined in section 5;

– the system's specification P is evaluated for compliance with the formula Φ; if P does not satisfy Φ, an enforcement process X has to be calculated so that the resulting enforced system $P \upharpoonright X$ satisfies Φ.

Our Framework for Policy Enforcement (FPE) contains a sensible amount of original contributions. The concept of ambients of Cardelli and Gordon has been fundamentally changed. There are marked differences in our new algebra, such as the direction of movement, ambient protection through keys, and modalities for capabilities. Our logic involves capability and protected location primitives and the quotient operator is, to our knowledge, the first method for automatic enforcement calculation that involves ambients. Finally, the application is developed from scratch for the specific purpose of implementing the enforcement calculation algorithm.

4 Security Enforcement Calculus

In this section we define the syntax and the semantics of a calculus suited for specifying, at an abstract level, a given network with the behaviors of network components, including network protections.

4.1 Syntax

Let $\mathcal{N}$ be a set of domain names and let $\mathcal{K}$ be the set of keys used to access protected domains. Let $\mathcal{A}$ be a set of process actions and let $\mathcal{C}$ be a set of communication channels. The syntax of our specification calculus is presented in Table 1. This syntax defines the building blocks needed for specifying secured computer systems. System's security is mainly achieved through access control mechanisms. We define two process constants, 0 and 1, representing respectively a process deadlock (or blocking) and successful termination. We use the following

operators: "." denotes the sequence operator, "$|_\gamma$" denotes parallel composition, "+" stands for nondeterministic choice, "!" denotes the infinite replication operator, and "$\triangleright$" represents the enforcement operator. The expression $P \triangleright Q$ describes a process P enforced by a process Q.

Let $\mathcal{P}$ be the set of all processes that can be expressed by our calculus. Processes can be enclosed in protected ambients. An ambient is a named domain and its name *n* is used for identifying the ambient and for locating the domain resources. Domains are uniquely identified, meaning that there cannot be two distinct ambients sharing the same name. In our approach an ambient is always protected by an access key *k* used for controlling access to resources. The appropriate key must be used for entering an ambient. We define a partial order on the set of keys $\mathcal{K}$. Let $(\mathcal{K}, \geq)$ be a partial ordered set, and let *k*, *k′* in $\mathcal{K}$. The expression $k \geq k'$ means that *k* is comparable to *k′*, but more powerful, as it can open at least any ambient *k′* can open. We denote by δ the public key which is the $glb(\mathcal{K})$. The partial order on keys is introduced to allow the specification of different levels of privileges and to better reflect the hierarchy of passwords in a computer system: superuser password, user password, guest password, etc.. Moreover, the partial ordering on the set of keys allows for more concise specifications. For example, a single key can be used to cross several protected domains (e.g.: rule 4.15 in Table 4) if it is stronger than the protection keys used in these domains.

The public key allows modeling domains with unrestricted access. The notion of ambient protection by keys offers an effective mechanism for the enforcement of access control policies. This is comparable to a distributed Policy Decision Point - Policy Enforcement Point. Hence, in our framework, the enforcement of an access control policy is translated into a key-based protection mechanism: a process can move into a domain and use its resources only if it is in possession of the right key. We think that the use of such a mechanism is sufficient to allow the enforcement of a large class of safety properties.

An ambient also possesses an interface *i* specifying a set of communication channels used for interaction with the rest of its environment. Two ambients can communicate only if their interfaces share at least a common channel. General process interactions are expressed through a communication function, γ, which is a partial function of $\mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$ that satisfies the two following conditions:

1. $\forall a, b \in \mathcal{A} \ : \ \gamma(a,b) = \gamma(b,a)$ (communtativity)

2. $\forall a,b,c \in \mathcal{A} \ : \ \gamma(\gamma(a,b),c) = \gamma(a,\gamma(b,c))$ (associativity)

There are two types of actions: regular and enforcing. The regular actions enable the execution of process capabilities. The enforcing actions are used for modifying process behaviour. We consider the following particular regular actions in order to define process capabilities: $\mathtt{mov}_n^k$ (movement) and $\mathtt{prot}_n^k$ (protection). Movement represents the ability of a process to circulate in and out of ambients, provided it uses the appropriate key. For instance, the process $\mathtt{mov}_n^k.\mathtt{P}$ will enter an ambient *n* protected by the access key *k′* such that $k \geq k'$ and then will behave like P. Protection refers to an access key change, and is always applied from within an ambient. The process $\mathtt{prot}_n^k.\mathtt{P}$ within an ambient *n* protected by *k′* will modify its parent ambient's key to *k* and then continue executing as P. We use the modalities "•a", "▲a", and "▼a" to depict enforcing actions. They preserve, prevent or to enforce the execution of some action *a*, respectively. The •a modality used within an enforcement process ensures that the first action a of a process a.P is preserved and executes as scheduled. The opposite effect is achieved through ▲a, which triggers the removal of the action a from the same process. The ▼a modality allows us to insert the action a as the first executable action of a process P, effectively transforming it into a.P. Movement actions can be used to describe a system behaviour, so they can be part of a system specification. They are also used for transporting enforcements to the desired location, so movements can also be part of enforcement processes. The action modalities can only exist during the enforcement phase, so they cannot be part of a system specification.

Movement between two ambients can be defined by specifying sets of intermediary ambients. Consider a move of a process to an ambient *m* from a current location. Let *s* be the set of intermediary ambients between the origin and destination ambient. We denote by $\mathtt{mov}_{s,m}^k$ the sequence of movements to *m* through the intermediary ambients in *s*. More formally:

$$\begin{cases} \mathtt{mov}_{\emptyset,m}^k = \mathtt{mov}_m^k \\ \mathtt{mov}_{s,m}^k = \sum_{s_i \in s} (\mathtt{mov}_{s_i}^k . \mathtt{mov}_{s\setminus\{s_i\},m}^k), \text{ if } s \neq \emptyset \end{cases}$$

Moreover, in order to keep the specification light, we will denote $\mathtt{mov}_n^\delta$ by $\mathtt{mov}_n$.

4.2 Semantics

The operational semantics of our calculus is defined by two main components: the structural congruence, denoted by "≡" and the reduction relation, denoted by "→". Table 2 presents a structural congruence on processes, which is introduced to streamline the definition of the reduction relation in Table 4. The first twelve cases (2.1 - 2.12) reflect common algebraic properties, such as reflexivity, symmetry, transitivity, and commutativity. Cases (2.13 - 2.15) show different equivalences when a blocking process is involved. Indeed, the effect of a blocking process on a sequence is obvious. If there is a choice between a non-blocking process and a deadlock, the choice defaults to continuing the execution. Therefore, the non-blocking process will execute. The enforcement of a deadlock, used to enforce the *ff* policy, will however block the whole targeted process. The successful termination process, covered by (2.16 - 2.19), is a neutral element for sequence, parallelism, enforcement, and choice. The cases (2.20 - 2.31) highlight some fundamental properties of our syntactic operators: commutativity, distributivity, idempotence, etc.. We use "$\rightarrow^*$" as a transitive closure of "→". Note that enforcement, unlike parallelism, is not symmetrical. Whenever an enforcement is applied to a process, the right hand term has to terminate before the left hand term starts executing.

Table 3 defines the predicate "_↓", which is used in the definition the reduction relation. `P`↓ means that `P` has the option to terminate successfully. The cases (3.2) and (3.3) are obvious: if one of processes `P` and `Q` has the option to terminate successfully, then so does the choice of the two. Cases (3.4) and (3.5) concern the sequence and parallelism operators. An expression `P.Q` or `P|Q` has the option to terminate successfully if both `P` and `Q` do so. The case (3.6) states that the ambient boundaries have no influence on a process's option to terminate successfully.

The reduction relation in Table 4 captures all possible process evolutions. Rules (4.1 - 4.5, 4.7, 4.8) capture the standard operational semantics of our process algebra. Rule (4.6) allows an evolution of the process representing the system only if the enforcement process has the opportunity to terminate. Our syntax allows a complete control over process behaviour with the aid of enforcement processes. Given a process `a.P`, an enforcement can allow `a` to run as scheduled (4.10) or remove it (4.11). The use of the remove modality is shown in the following example:

$$\mathrm{mov}_n^k.(P+Q) \uparrow \blacktriangle \mathrm{mov}_n^k.R \xrightarrow{\blacktriangle \mathrm{mov}_n^k} (P+Q) \uparrow R$$

Alternately, an enforcement process can insert a completely new action to be executed as the first action of a process (4.12). A combination of removal and insertion is useful, for example, when the network topology changes due to a link being added or failing. In such a case, security policies need to be updated to reflect the affected communication interfaces and alternate paths. They will in turn be implemented through enforcement processes that prescribe the corresponding movement capabilities.

The actual mechanisms for movement inside, out, and across ambients are captured by rules (4.13) to (4.15). The example below illustrates both (4.13) and (4.14).

$$\mathsf{mov}_n^k.\mathsf{P} \mid_{\gamma n}^{k} [\mathsf{Q} \mid_{\gamma} \mathsf{R}]^i \mid_{\gamma m}^{k'} [\mathsf{mov}_m^{k'}.\mathsf{S} \mid_{\gamma} \mathsf{T}]^j \xrightarrow{\mathsf{mov}_n^k, \mathsf{mov}_m^{k'}} {}_n^k[\mathsf{P} \mid_{\gamma} \mathsf{Q} \mid_{\gamma} \mathsf{R}]^i \mid_{\gamma} \mathsf{S} \mid_{\gamma m}^{k'} [\mathsf{T}]^j$$

Ambient access keys can be modified as well, whenever required by a policy change, in order to lower or elevate the ambient's protection level (4.16).

Definition 1 (Normal form). *A process* $\mathtt{P}$ *is in its normal form, denoted by* $\mathtt{P}\Downarrow$*, iff there is no action* $\mathrm{a} \in \mathcal{A}$ *so that* $\mathsf{P} \xrightarrow{a} \mathsf{P}'$.

We extend the structural congruence relation as follows:

$$(\mathsf{P} \upharpoonright \mathsf{Q})_{\Downarrow} \Rightarrow \mathsf{P} \upharpoonright \mathsf{Q} \equiv \mathsf{P} \qquad (2.32)$$

5 Logic for Security Policy Specification

In this section, we define a dedicated logic suited to specify security policies for computer systems described with our calculus. The intent is to have a formalism that allows us to specify safety properties. An example of such a property could be the prohibition for a computer to communicate with another host located inside or outside of its domain. Since the properties can be significantly more complex, the logic must be expressive enough to specify any safety property. The proposed logic is inspired from the work of Cardelli and Gordon [4, 5]. It allows manipulation of spatial and temporal modalities, i.e. the current place (location) in the system and the order of execution of various actions.

5.1 Syntax

We define a modal logic with standard propositional connectives for negation and disjunction ($\neg$, $\vee$), and a capability operator ($\langle\ \rangle$). Furthermore, we define spatial connectives ($|$, $[\ ]$). The syntax of our logic is summarized in Table 5. The set of logical formulas specified in our logic is denoted by $\mathcal{F}$. We define some standard macros, as follows:

$$\Phi \wedge \Psi \equiv \neg(\neg\Phi \vee \neg\Psi) \qquad \Phi \rightarrow \Psi \equiv \neg\Phi \vee \Psi \qquad ff \equiv \neg tt$$

5.2 Semantics

Let $\{\Phi, \Psi\} \subseteq \mathcal{F}$, $n \in \mathcal{N}$, $i \subseteq \mathcal{C}$ and $k \in \mathcal{K}$. The semantics of our logic is given by the meaning function $[\![\_]\!]: \mathcal{F} \rightarrow 2^{\mathcal{P}}$ defined inductively on the structure of formulas. Table 6 shows the logic's semantics. We say that a process $\mathtt{P}$ satisfies the formula Φ and we note $\mathtt{P} \models \Phi$ if $\mathtt{P} \in [\![\Phi]\!]$. The formula *tt* is satisfied by all processes, except for the blocking process (which satisfies *ff*). The negative form of the policy $\neg\Phi$ is satisfied by processes that are not part of the semantics of the policy Φ. A process satisfies the formula $\Phi \vee \Psi$ if it satisfies either the formula Φ or the formula Ψ. If $\mathtt{P}$ satisfies Φ, then the formula $\langle \mathtt{a} \rangle \Phi$ is satisfied by $\mathtt{a}.\mathtt{P}$. A process $\mathtt{P}|\mathtt{Q}$ satisfies the formula $\Phi|\Psi$ if process $\mathtt{P}$ satisfies Φ and process $\mathtt{Q}$ satisfies Ψ. The protected location logical formula reflects the case when a specific behaviour is required inside an ambient. Both the ambient's parameters (name, key, and interface) and the logic formula inside the ambient need to be satisfied by the process.

Lemma 1. *Let Φ and Ψ be two logical formulas and let $\mathtt{P}$ be a process. Then:*

$$\mathtt{P} \in [\![\Phi]\!] \quad \wedge \quad [\![\Phi]\!] \subseteq [\![\Psi]\!] \Rightarrow \mathtt{P} \models \Psi$$

Proof. The proof is trivial.

It is worth mentioning that since both the logic and the calculus are endowed with spatial operators, it is easy to produce modular specifications.

5.3 Elimination of the form $\neg\Phi$

We need to propagate the negation operator inside the formulas in order to limit its scope to atomic actions. This transformation will be of great help to simplify the definition of the quotient operator given in Table 8 in the next section. Table 7 presents a rewriting system that allows the calculation of such transformations. It is easy to verify that all applied transformations lead to equivalent logical formulas. The following example demonstrates how the rewriting system can be applied to push the negation operator down to atomic actions.

$$\begin{aligned}
\neg(\langle \mathtt{mov}_n^k \rangle \langle \mathtt{mov}_m^{k'} \rangle tt) &\xrightarrow{(7.4)} (\neg\langle \mathtt{mov}_n^k \rangle) tt \vee \langle \mathtt{mov}_n^k \rangle \neg(\langle \mathtt{mov}_m^{k'} \rangle tt) \\
&\xrightarrow{(7.4)} (\neg\langle \mathtt{mov}_n^k \rangle) tt \vee \langle \mathtt{mov}_n^k \rangle ((\neg\langle \mathtt{mov}_m^{k'} \rangle) tt) \vee \langle \mathtt{mov}_m^{k'} \rangle \neg tt) \\
&\xrightarrow{(7.1)} (\neg\langle \mathtt{mov}_n^k \rangle) tt \vee \langle \mathtt{mov}_n^k \rangle (\neg\langle \mathtt{mov}_m^{k'} \rangle) tt \vee \langle \mathtt{mov}_n^k \rangle \langle \mathtt{mov}_m^{k'} \rangle f\ f
\end{aligned}$$

6 Security Policy Enforcement

Once the system and the security policy have been specified with our algebraic calculus and logic, the next important step is to extract automatically an enforcement process that imposes the

behavior of the system's model as stated by the policy. The main intent of this approach is to automatically identify the enforcement components that allow to impose a given policy on a system.

This can be viewed as an alternative representation of the interface equation problem [21–24]. The idea is to derive a quotient process from two given processes. The derived quotient process represents, then, the missing part for the two processes to be equivalent. In this paper, the quotient is defined in terms of a process and a logical formula. Therefore, the equation we need to solve has the form:

$P \upharpoonright X \models \Phi$

where P is the formal description of a system and Φ is the security policy to be enforced.

In the equation shown above, X can be expressed as a quotient. The solution X of this equation is the enforcement we look for:

$$X = \frac{\Phi}{P}$$

We have shown in Table 2 that enforcements don't have any effect on a deadlock process. Also, it is obvious that there is no need for an enforcement if a process already satisfies the policy. For all other situations, the quotient operator $\frac{-}{-}$ is formally defined in Table 8. Several examples of process enforcements are also provided in this section.

All processes, except the blocking process, satisfy the formula *tt* (8.1), so there is nothing to enforce (i.e. the value of the quotient is 1). Since no valid process can satisfy *ff*, the generated enforcement at (8.2) is 0, which has the effect of blocking the system. Rules (8.3 - 8.7) refer to policies that require a particular action. Consequently, the targeted action of the process is preserved, added or removed. The rule (8.3) applies when the process starts by the action specified in the formula. The enforcement process must then keep the action in place, therefore the action $\bullet a$ is generated for the enforcement process.

The rule (8.6) is quite similar, with $\bullet b$ preserving in this case an action which is different from the action a prohibited by the policy. If the desired action is different, as in (8.4), that particular action needs to be enforced. This means that the non-compliant action b needs to be neutralized first, followed by the insertion of the new action a. A fitting example would be the use of a new access key *k″* for a movement action instead of the obsolete key *k*. Let $P = \mathtt{mov}_n^k.\mathtt{mov}_m^{k'}.1$ and

let $\Phi = \langle \mathrm{mov}_n^{k''} \rangle tt$. Notice that the formula specifies that the movement action needs to use the new key. The enforcement in this case is given by the expression:

$$\frac{\Phi}{\mathrm{P}} = \mathrm{X} = \blacktriangle \mathrm{mov}_n^k . \frac{tt}{\mathrm{mov}_m^{k'}.1} . \blacktriangledown \mathrm{mov}_n^{k''} = \blacktriangle \mathrm{mov}_n^k .1. \blacktriangledown \mathrm{mov}_n^{k''} \equiv \blacktriangle \mathrm{mov}_n^k . \blacktriangledown \mathrm{mov}_n^{k''}$$

Therefore, $\mathrm{P} \upharpoonright \mathrm{X} = \mathrm{mov}_m^{k''}.\mathrm{mov}_m^{k'}.1$.

The rule (8.5) follows the same reasoning, but in this case we seek satisfaction of the complementary action. The rule (8.7) applies to the enforcement of a policy on the process 1. The rule (8.8) concerns the enforcement of a disjunction of two formulas forming the policy. Either $\frac{\Phi}{\mathrm{P}}$ or $\frac{\Psi}{\mathrm{P}}$ will produce a compliant process, when enforced on P. In the case of the indeterminate choice (8.9), the formula can be applied to either P or Q. Let $\mathrm{P} = \mathrm{mov}_n^k.\mathrm{P}'$, $\mathrm{Q} = \mathrm{mov}_m^{k'}.\mathrm{Q}'$ and let $\Phi = \langle \mathrm{mov}_n^k \rangle tt$. The enforcement for $\mathrm{P} + \mathrm{Q}$ is:

$$\frac{\Phi}{\mathrm{P}+\mathrm{Q}} = \bullet \mathrm{mov}_n^k . \frac{tt}{\mathrm{P}'} + \blacktriangle \mathrm{mov}_m^{k'} . \frac{tt}{\mathrm{Q}'} . \blacktriangledown \mathrm{mov}_n^k = \bullet \mathrm{mov}_n^k .1 + \blacktriangle \mathrm{mov}_m^{k'} . \blacktriangledown \mathrm{mov}_n^k$$

Ambient enforcement, with and without access key changes, is defined in rules (8.10) and (8.11). The rule (8.10) allows the enforcement process to go inside an ambient and apply the enforcement. The modification of the ambient protection key is given through rule (8.11). Targeted enforcement is also supported by our enforcement operator, as shown by (8.12). The ambient name permits the delivery of dedicated enforcements on different locations within the process. Let $\Phi = tt$, $\Psi = \langle \mathrm{mov}_m^{k'} \rangle tt$, $\mathrm{P} = \mathrm{mov}_m^{k''}.\mathrm{P}'$, and $\mathrm{Q} = \mathrm{mov}_n^k.\mathrm{Q}'$. The enforcement X for the formula ${}_n^k[\Phi]^i \mid {}_m^{k''}[\Psi]^j$ on the process ${}_m^{k''}[\mathrm{P}]^j \mid_\gamma {}_n^{k'''}[\mathrm{Q}]^i$ is calculated as follows:

$$\frac{{}_n^k[\Phi]^i \mid {}_m^{k''}[\Psi]^j}{{}_m^{k''}[\mathrm{P}]^j \mid_\gamma {}_n^{k'''}[\mathrm{Q}]^i} = \frac{{}_n^k[tt]^i}{{}_n^{k'''}[\mathrm{mov}_n^k.\mathrm{Q}']^i} \mid_\gamma \frac{{}_m^{k''}[\langle \mathrm{mov}_m^{k'} \rangle tt]^j}{{}_m^{k''}[\mathrm{mov}_m^{k''}.\mathrm{P}']^j}$$

The results of the two new quotients obtained are:

$$\frac{{}_n^k[tt]^i}{{}_n^{k'''}[\mathrm{mov}_n^k.\mathrm{Q}']^i} = \mathrm{mov}_n^{k'''}.\mathrm{prot}_n^k . \frac{tt}{\mathrm{mov}_n^k.\mathrm{Q}'} = \mathrm{mov}_n^{k'''}.\mathrm{prot}_n^k .1$$

$$\frac{{}_m^{k''}[\langle \mathrm{mov}_m^{k'} \rangle tt]^j}{{}_m^{k''}[\mathrm{mov}_m^{k''}.P']^j} = \mathrm{mov}_m^{k''} . \blacktriangle \mathrm{mov}_m^{k''} . \frac{tt}{P'} . \blacktriangledown \mathrm{mov}_m^{k'} = \mathrm{mov}_m^{k''} . \blacktriangle \mathrm{mov}_m^{k''} . \blacktriangledown \mathrm{mov}_m^{k'} .$$

Therefore value of the enforcement is: $X = \mathrm{mov}_n^{k'''}.\mathrm{prot}_n^k .1 \mid_\gamma \mathrm{mov}_m^{k''} . \blacktriangle \mathrm{mov}_m^{k''} . \blacktriangledown \mathrm{mov}_m^{k'}$

Theorem 1 (Enforcement Correctness). *Let* $\Phi \in \mathcal{F}$,$\mathsf{P} \in \mathcal{P} \setminus \{0\}$, *and let* $\mathsf{X} = \frac{\Phi}{\mathsf{P}}$. *Then:*

$\mathsf{P} \wr \mathsf{X} \vDash \Phi$

Proof sketch

The proof is done by a structural induction on the logical formula Φ representing the policy. We show that for any process `P` representing a system, the generated enforcement `X` is able to rewrite `P` such that the enforced process $\mathsf{P} \wr \mathsf{X}$ always belongs to the semantics of Φ. To conduct the proof, we verify that the proposition holds for the constants *tt* and *ff* and prove that if the proposition holds for formulas Ψ_1 and Ψ_2 then it holds $\nu\Psi_1$ and $\Psi_1 \mu \Psi_2$, where ν and μ are modalities and operators of the logic.

The proven theorem is crucial for the validation of our approach. It demonstrates that the enforcement will always produce the correct result.

7 Case Study

In this section, we illustrate our technique with the example of the network depicted in Fig. 2. The example demonstrates the use of FPE for specifying and enforcing a security policy on the system. We show that the generated enforcement policy produces the expected changes.

7.1 System Specification

The subject of the case study is a simplified version of a library system as depicted in Fig. 2. In order to make the example easy to comprehend, the number of computer systems has been kept to a minimum: one for a library guest and four for the library. We identified two logical zones: the *Internet* logical zone containing the Guest system and the *Library* logical zone containing the library's own computer systems. The *Library* zone initially contains one computer system for the library's portal server (`Portal`), two for the reservation and fine payments (`Borrowing` and `Fines`) and one for online resources (`Resources`).

Each computer system is represented by a non-blocking process running in a protected ambient. Specific keys are only defined for ambients requiring safeguards: k_l and k_f corresponding to portal and borrowing systems, respectively. All other keys are set to the default value δ, which states that there are no access restrictions. For ease of reading, we have chosen to omit δ from the specification. This means that processes $\mathtt{mov}_n^{\delta}.\mathtt{P}$ and ${}_n^{\delta}[\mathsf{P}]^{i_n}$ will be represented as $\mathtt{mov}_n.\mathtt{P}$ and ${}_n[\mathsf{P}]^{i_n}$, respectively. For the same reason, by abuse of notation, we will denote "$|_\gamma$" by "|", and

suppose that all communications are well defined. Access to the library portal (provided that k_l is known) does not grant access to `Borrowing`, but is required as a preliminary step.
Currently, guests can browse online resources, reserve items from the library and pay fines for late returns. Fine payments are dependant on the borrowing system, as they are linked to the library catalogue. The workflow is straightforward. Guests initiate a session by authenticating to the portal with credentials provided by the library. Once authenticated, they are presented with the choices (browse or borrow) and carry on their intended tasks until they decide to close the session. Browsing online resources does not require special permissions. Consulting the library catalogue and borrowing items, however, involve an additional password. Paying fines does not require special permissions, but can only be done after accessing the borrowing section. The specification for the guest system is as follows:

$$\mathtt{G} = {}_{g}[\mathtt{mov}_{l}^{k_l}.(\mathtt{mov}_{r}.\mathtt{G}_1 + \mathtt{mov}_{b}^{k_b}.(\mathtt{G}_2 + \mathtt{mov}_{f}.\mathtt{G}_3))]^{i_g}$$

$$where\ \mathtt{G}_1, \mathtt{G}_2, \mathtt{G}_3 \neq 0$$

The library allows access to its online resources on the `Resources` web server through the `Portal`. The web server handles reservation requests for items from its `Borrowing` catalogue. Late return fines for borrowed items are processed through `Fines`. The specification for the library system is:

$$\mathtt{L} = {}_{l}^{k_l}[\mathtt{L}_1 \mid {}_{b}^{k_b}[\mathtt{B} \mid {}_{f}[\mathtt{F}]^{i_f}]^{i_b} \mid {}_{r}[\mathtt{R}]^{i_r}]^{i_l}$$

$$where\ \mathtt{L}_1, \mathtt{B}, \mathtt{F}, \mathtt{R} \neq 0$$

The whole system is composed of the library and guest systems `L` and `G`. It is therefore specified by the process `S` below:

$$\begin{aligned} \mathtt{S} &= \mathtt{L} \mid \mathtt{G} \\ &= {}_{l}^{k_l}[\mathtt{L}_1 \mid {}_{b}^{k_b}[\mathtt{B} \mid {}_{f}[\mathtt{F}]^{i_f}]^{i_b} \mid {}_{r}[\mathtt{R}]^{i_r}]^{i_l} \mid {}_{g}[\mathtt{mov}_{l}^{k_l}.(\mathtt{mov}_{r}.\mathtt{G}_1 + \mathtt{mov}_{b}^{k_b}.(\mathtt{G}_2 + \mathtt{mov}_{f}.\mathtt{G}_3))]^{i_g} \end{aligned}$$

To summarize, the signification of processes is as follows:

– `G`: represents the Guest process;

– $\mathtt{G}_1$: symbolizes the process used by the Guest to access online resources;

– $\mathtt{G}_2$: denotes the activities required by the Guest for browsing the catalogue and borrowing items;

– $\mathtt{G}_3$: stands for the activities required by the Guest for paying fines for late returns;

– L: represents the Library process;

– L_1: corresponds to the Library portal access;

– B: represents the Library's catalogue browsing and borrowing services;

– F: denotes to the process associated with fine payment processing;

– R: stands for the online resources web server.

The interfaces share different channels to accommodate communication between ambients: $i_g = \{gl\}, i_l = \{gl, lb, lr\}, i_b = \{lb, bf\}, i_f = \{bf\}$ and $i_r = \{lr\}$. This enables the following:

– guest can communicate with the portal;

– the portal can communicate with the borrowing and resources systems;

– the borrowing and fines systems can communicate;

– no other communication is defined.

The library decides to finally upgrade their ageing catalogue system. The library managers decide to reduce guest access temporarily to online resources only. For a certain period, guests will no longer be allowed to consult the catalogue and borrow items. In order to do that, the Borrowing system's key k_b needs to be changed to a new value k_b'. The new policy for the library system is:

$$\Phi_l = {}^{k_l}_{l}[tt \mid {}^{k_b'}_{b}[tt]^{i_b}]^{i_l}, where\ k_b' \neq k_b$$

The formula Φ_l would be satisfied if the Library process meets several conditions. First, the process must be an ambient named l, protected by the key k_l and communicating across an interface i_l. Second, the ambient must contain a parallelism that involves a non-blocking process and an ambient b. Finally, ambient b must be protected by a new key k_b' and have an interface i_b.

7.2 Security Policy Enforcement

The enforcement required to make the library system compliant with the new policy is given by:

$$X_l = \frac{\Phi_l}{\mathrm{L}}$$

The enforcement process for the library system is calculated as follows:

$$X_l = \frac{\Phi_l}{\mathtt{L}} = \frac{{}^{k_l}_{l}[tt \mid {}^{k_b'}_{b}[tt]^{i_b}]^{i_l}}{{}^{k_l}_{l}[\mathtt{L}_1 \mid {}^{k_b}_{b}[\mathtt{B} \mid {}_{f}[\mathtt{F}]^{i_f}]^{i_b} \mid {}_{r}[\mathtt{R}]^{i_r}]^{i_l}}$$

$$= \mathtt{mov}^{k_l}_{l}.\left(\frac{tt \mid {}^{k_b'}_{b}[tt]^{i_b}}{\mathtt{L}_1 \mid {}^{k_b}_{b}[\mathtt{B} \mid {}_{f}[\mathtt{F}]^{i_f}]^{i_b} \mid {}_{r}[\mathtt{R}]^{i_r}}\right) \qquad (by\ 8.10)$$

$$= \mathtt{mov}^{k_l}_{l}.\left(\frac{tt}{L_1 \mid {}_{r}[\mathtt{R}]^{i_r}} \mid \frac{{}^{k_b'}_{b}[tt]^{i_b}}{{}^{k_b}_{b}[\mathtt{B} \mid {}_{f}[\mathtt{F}]^{i_f}]^{i_b}}\right) \qquad (by\ 8.12)$$

$$= \mathtt{mov}^{k_l}_{l}.\left(1 \mid \mathtt{mov}^{k_b}_{b}.\mathtt{prot}^{k_b'}_{b}.\frac{tt}{\mathtt{B} \mid {}_{f}[\mathtt{F}]^{i_f}}\right) \qquad (by\ 8.1, 8.11)$$

$$= \mathtt{mov}^{k_l}_{l}.\mathtt{mov}^{k_b}_{b}.\mathtt{prot}^{k_b'}_{b}.1 \qquad (by\ 2.17, 8.1)$$

We have computed the necessary enforcement corresponding to the new policies based on the quotient operator table defined in the previous section. It is now time to verify that the enforcements work as expected and the modified systems satisfies the new policy: $\mathtt{L} \setminus \mathtt{X}_l \models \Phi_l$. We proceed by calculating the system specification for the enforced `Library` system by using the reduction relation defined in Table 4:

$$\mathtt{L} \setminus \mathtt{X}_1 = {}^{k_l}_{l}[\mathtt{L}_1 \mid {}_{b}[\mathtt{B} \mid {}_{f}[\mathtt{F}]^{i_f}]^{i_b} \mid {}_{r}[\mathtt{R}]^{i_r}]^{i_l} \setminus \mathtt{mov}^{k_l}_{l}.\mathtt{mov}^{k_b}_{b}.\mathtt{prot}^{k_b'}_{b}.1$$

$$\xrightarrow{\mathtt{mov}^{k_l}_{l}} {}^{k_l}_{l}[\mathtt{L}_1 \mid {}_{b}[\mathtt{B} \mid {}_{f}[\mathtt{F}]^{i_f}]^{i_b} \mid {}_{r}[\mathtt{R}]^{i_r} \setminus \mathtt{mov}^{k_b}_{b}.\mathtt{prot}^{k_b'}_{b}.1]^{i_l} \quad (by\ 4.13)$$

$$\xrightarrow{\mathtt{mov}^{k_b}_{b}} {}^{k_l}_{l}[\mathtt{L}_1 \mid {}_{b}[\mathtt{B} \mid {}_{f}[\mathtt{F}]^{i_f} \setminus \mathtt{prot}^{k_b'}_{b}.1]^{i_b} \mid {}_{r}[\mathtt{R}]^{i_r}]^{i_l} \quad (by\ 4.13)$$

$$\xrightarrow{\mathtt{prot}^{k_b'}_{b}} {}^{k_l}_{l}[\mathtt{L}_1 \mid {}^{k_b'}_{b}[\mathtt{B} \mid {}_{f}[\mathtt{F}]^{i_f}]^{i_b} \mid {}_{r}[\mathtt{R}]^{i_r}]^{i_l} \quad (by\ 4.16)$$

The policy satisfaction relation for Φ_l can then be written as:

$$\mathtt{L} \upharpoonright \mathtt{X}_1 \vDash \Phi_l \quad \Leftrightarrow \quad {}^{k_l}_{l}[\mathtt{L}_1 \mid {}^{k_b'}_{b}[\mathtt{B} \mid {}_{f}[\mathtt{F}]^{i_f}]^{i_b} \mid {}_{r}[\mathtt{R}]^{i_r}]^{i_l} \vDash {}^{k_l}_{l}[tt \mid {}^{k_b'}_{b}[tt]^{i_b}]^{i_l}$$

$$\Leftrightarrow \quad \mathtt{L}_1 \mid {}^{k_b'}_{b}[\mathtt{B} \mid {}^{k_f}_{f}[\mathtt{F}]^{i_f}]^{i_b} \mid {}_{r}[\mathtt{R}]^{i_r} \vDash tt \mid {}^{k_b'}_{b}[tt]^{i_b} \quad (by\ 6.7)$$

We can verify that indeed $\mathtt{L} \upharpoonright \mathtt{X}_l \vDash \Phi_l$, since $\mathtt{L}_1 \mid {}_{r}[\mathtt{R}]^{i_r} \vDash tt$ (by 6.1) and ${}^{k_b'}_{b}[\mathtt{B} \mid {}_{f}[\mathtt{F}]^{i_f}]^{i_b} \vDash {}^{k_b'}_{b}[tt]^{i_b}$ (by 6.7 and 6.1). Therefore, the library system has been successfully enforced to satisfy the new policy.

8 Software Implementation

The main purpose of developing PEA (short for "Policy Enforcement Application") was to mechanize the method developed in this paper and to demonstrate its applicability. Given a computer network system, no matter how complex, the application allows us to build the network topology, generate the network specification using the calculus, specify policies with the logic and calculate required enforcements based on the quotient operator formulas.

The application was developed in Java [25] using Eclipse [26], with Swing libraries [27] being used for the GUI development. The architecture of our application contains different roles, a GUI and three modules that implement the elements of FPE. One module translates topology information into an interprocess algebra-based specification. The second module is used for defining system security policies with logical formulas. Enforcement calculations are performed by the third module. Since we used Eclipse, it is easy to extend the application by adding new modules. The addition of libraries of other elements (services, other network hardware, etc.) is also straightforward. The internationalization feature of Java allows interface components (menu items, labels, etc.) to be defined in your language of choice. Currently, there are English and French versions of the application. The software is available on GitHub under GPL license.

The interface is intuitive and easy to use. Fig. 3 displays the network topology described in our case study, along with the associated network specification. The system's topology is built by simply dragging predefined elements (computers, routers, firewalls, etc.) into the main window of the GUI. Associated details such as ambient names and keys, or processes, can be added as properties of the elements via a context-aware box. Once the components have been inserted, a dedicated button can be used to connect them by pointing to the ends of the link.

The network specification for the system built is displayed at the bottom of the main window. Any change to the topology or process details is automatically applied to the specification. All

applicable process reductions are also taken into account and are transparently performed in the background.

Systems built with the aid of the application can be modified, copied, exported to XML files and imported for re-utilization. Links between components can be added and removed through a simple right-click. Parent-child ambient relations can be changed via drag-and-drop. Ambient names, keys, interfaces, and process details are easily defined, as illustrated in Fig. 4 and Fig. 5. PEA also allows specifying security policies by using a GUI-style method. The policy is displayed both in logic formulas format and as an XML style structure showing the components, as in Fig. 6.

The main window displays the topology and system specification. If the policy window, showing the logic formulas, is also open, one can calculate the necessary enforcement to be applied to the system. The quotient button permits the automated generation of an adequate enforcement, as demonstrated in Fig. 7.

The application is the first step in our effort to obtain automatic enforcements for the implementation of desired security policies. It can be employed for various network simulations: system design, policy verification, disaster recovery scenarios, etc.

9 Conclusion

In this paper, we have developed a new formal framework for computer system specification and security policy enforcement. The FPE framework consists of three main components. The first is a new formalism for specifying computer systems that captures in an effective and elegant way the system's behaviour and topology. We define a new calculus that draws from the agility of the mobile ambients and adds several original and powerful concepts such as access control for domains (access keys, protection changes) and communication interfaces. It facilitates the specification of a computer system's current state and its evolution. The second component is a new dedicated logic for defining security policies. The logic formulas allow expressing current and desired security policies. The semantics of the logic links policy satisfaction to compliant processes. Again, the system's evolution can be followed, in this case from the point of view of applicable constraints. The third original component is the quotient operator that allows an automatic calculation of required enforcements. Given a process P and a security policy Φ, we calculate $X = \frac{\Phi}{P}$ as a first step. If X = 1, then the process P satisfies the policy Φ. Otherwise,

the enforcement can be applied to update the process and make it policy compliant. The formal foundation of all components ensures that the enforcement produces a secured system, free of incomplete specifications, arbitrary interpretations or faulty implementation of policies.
Note that our approach is different from previous works as it allows both the verification of policy compliance and the application of corrective actions. The automated aspect of the methodology further enhances its value. A software implementation demonstrates its effectiveness and proves its applicability to practical problems. The versatility of FPE is supported by the numerous potential uses. It could be a free and lightweight tool employed by educators to simulate basic or complex network topologies, or to build and test security policies. A detailed specification that includes all potentially vulnerable systems is a worthwhile effort. Intrusion prevention systems of firewalls could take advantage of it by blocking exploits of known vulnerabilities through access key changes for the affected components. This should be done in conjunction with patch management systems, which could signal when access can be re-enabled. Cyber attacks could be modelled to determine whether critical systems can be reached, and under which conditions. They can be further analyzed to decide the best detection and prevention strategy. The approach can also be beneficial for business continuity planning while designing alternate configurations or disaster recovery scenarios. Such cases imply system specifications adapted to the potential crisis (natural or human-triggered disasters, hardware failure, wars, etc.). Our methodology could be used to validate whether the expectations (i.e. security policy) can be satisfied by the available computer systems (i.e. part of the new system specification). Further extensions of the calculus syntax, in particular, would increase the range of applications. In its current state, the access key incorporates all conditions necessary to enter an ambient. In practice, they correspond to more than just system credentials or access control lists. A finer granularity would allow a more precise control over process movement. A new action could model direct communications between a sender and a receiver that are not neighbours (encrypted channels, quantum computing, etc). Other actions that do not relate specifically to mobility would enrich specifications to better reflect the complex set of computer system behaviours.

Acknowledgement

This research is supported by a research grant from the Natural Science and Engineering Council of Canada, NSERC.

Appendix

Theorem 1 (Enforcement Correctness). *Let* $\Phi \in \mathcal{F}$,$\mathsf{P} \in \mathcal{P} \setminus \{0\}$, *and let* $\mathsf{X} = \frac{\Phi}{\mathsf{P}}$. *Then:*

$\mathsf{P} \wr \mathsf{X} \models \Phi$

Proof. The proof is done by a structural induction on the logical formula Φ representing the policy.

☑ $\Phi = tt$:

$\mathsf{X} = \frac{tt}{\mathsf{P}} = 1$ (*by* 8.1)

$\mathsf{P} \wr 1 \equiv \mathsf{P}$ (*by* 2.18)

$\mathsf{P} \models tt$ (by 6.1)

☑ $\Phi = ff$:

$\mathsf{X} = \frac{f\, f}{\mathsf{P}} = 0$ (*by* 8.2)

$\mathsf{P} \wr 0 \equiv 0$ (*by* 2.14)

$0 \models ff$ (*by* 6.2)

☑ $\Phi = \langle a \rangle \Psi$:

✓ $\mathtt{P} = \mathtt{a}.\mathtt{Q}$

$\mathsf{X} = \frac{\langle \mathtt{a} \rangle \Psi}{\mathtt{a}.\mathsf{Q}} = \bullet\, \mathtt{a}.\frac{\Psi}{\mathsf{Q}}$ (*by* 8.3)

$\mathtt{a}.\mathsf{Q} \wr \bullet\, \mathtt{a}.\frac{\Psi}{\mathsf{Q}} \xrightarrow{\bullet_{\mathtt{a}}} \mathtt{a}.(\mathsf{Q} \wr \frac{\Psi}{\mathsf{Q}})$ (*by* 4.10)

$\mathsf{Q} \wr \frac{\Psi}{\mathsf{Q}} \models \Psi$ (*by hypothesis*)

$\mathtt{a}.(\mathsf{Q} \wr \frac{\Psi}{\mathsf{Q}}) \models \langle \mathtt{a} \rangle \Psi$ (*by* 6.4)

✓ $\mathsf{P} = \mathtt{b}.\mathsf{Q}, \mathtt{b} \neq \mathtt{a}$

$\mathsf{X} = \frac{\langle \mathtt{a} \rangle \Psi}{\mathtt{b}.\mathsf{Q}} = \blacktriangle_{\mathtt{b}}.\frac{\Psi}{\mathsf{Q}}.\blacktriangledown_{\mathtt{a}}$ (*by* 8.4)

$\mathtt{b}.\mathsf{Q} \wr \blacktriangle_{\mathtt{b}}.\frac{\Psi}{\mathsf{Q}}.\blacktriangledown_{\mathtt{a}} \xrightarrow{\blacktriangle_{\mathtt{b}}} \mathsf{Q} \wr \frac{\Psi}{\mathsf{Q}}.\blacktriangledown_{\mathtt{a}}$ (*by* 4.11)

$$Q \upharpoonright \frac{\Psi}{Q}.\blacktriangledown a \equiv (Q \upharpoonright \frac{\Psi}{Q}) \upharpoonright \blacktriangledown a \qquad (by\ 2.22)$$

$$(Q \upharpoonright \frac{\Psi}{Q}) \upharpoonright \blacktriangledown a \xrightarrow{\blacktriangledown a} a.(Q \upharpoonright \frac{\Psi}{Q}) \qquad (by\ 4.12)$$

$$a.(Q \upharpoonright \frac{\Psi}{Q}) \models \langle a \rangle \Psi \qquad (by\ 6.4)$$

☑ $\Phi = \neg\langle a \rangle \Psi$:

✓ $P = a.Q$

$$X = \frac{\neg\langle a \rangle \Psi}{a.Q} = \blacktriangle a.\frac{\Psi}{Q} \quad (by\ 8.5)$$

$$a.Q \upharpoonright \blacktriangle a.\frac{\Psi}{Q} \xrightarrow{\blacktriangle a} Q \upharpoonright \frac{\Psi}{Q} \qquad (by\ 4.11)$$

$$Q \upharpoonright \frac{\Psi}{Q} \models \Psi \qquad (by\ hypothesis)$$

$$Q \upharpoonright \frac{\Psi}{Q} \not\models \langle a \rangle \Psi \quad (by\ 6.4)$$

$$Q \upharpoonright \frac{\Psi}{Q} \models \neg\langle a \rangle \Psi \qquad (by\ 6.3)$$

✓ $P = b.Q, b \neq a$

$$X = \frac{\neg\langle a \rangle \Psi}{b.Q} = \bullet b.\frac{\Psi}{Q} \quad (by\ 8.5)$$

$$b.Q \upharpoonright \bullet b.\frac{\Psi}{Q} \xrightarrow{\bullet b} b.(Q \upharpoonright \frac{\Psi}{Q}) \ (by\ 4.10)$$

$$Q \upharpoonright \frac{\Psi}{Q} \models \Psi \qquad (by\ hypothesis)$$

$$b.(Q \upharpoonright \frac{\Psi}{Q}) \models \langle b \rangle \Psi \qquad (by\ 6.4)$$

$$b \neq a \Rightarrow b.(Q \upharpoonright \frac{\Psi}{Q}) \not\models \langle a \rangle \Psi \qquad (by\ 6.4)$$

$$b.(Q \upharpoonright \frac{\Psi}{Q}) \models \neg\langle a \rangle \Psi \qquad (by\ 6.3)$$

☑ $\Phi = {}^{k'}_{n}[\Phi']^{i} \mid \Psi$ and $P = {}^{k}_{n}[P']^{i} \mid_{\gamma} Q$:

$$\mathsf{X} = \frac{{}^{k'}_{n}[\Phi']^{i} \mid \Psi}{{}^{k}_{n}[\mathsf{P}']^{i} \mid_{\gamma} Q} = \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}} \mid_{\gamma} \frac{\Psi}{\mathsf{Q}} \text{ (by 8.12)}$$

$$\begin{aligned} \mathsf{P} \upharpoonright \mathsf{X} &\equiv ({}^{k}_{n}[\mathsf{P}']^{i} \upharpoonright (\frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}} \mid_{\gamma} \frac{\Psi}{\mathsf{Q}})) \mid_{\gamma} (\mathsf{Q} \upharpoonright (\frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}} \mid_{\gamma} \frac{\Psi}{\mathsf{Q}})) \\ &\equiv (((({}^{k}_{n}[\mathsf{P}']^{i} \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}}) \upharpoonright \frac{\Psi}{\mathsf{Q}}) + (({}^{k}_{n}[P']^{i} \upharpoonright \frac{\Psi}{\mathsf{Q}}) \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}})) \mid_{\gamma} \end{aligned} \quad (by\ 2.31)$$

$$(((\mathsf{Q} \upharpoonright \frac{\Psi}{\mathsf{Q}}) \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}}) + ((\mathsf{Q} \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}}) \upharpoonright \frac{\Psi}{\mathsf{Q}})) \quad (by\ 2.28)$$

$${}^{k}_{n}[\mathsf{P}']^{i} \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}} \upharpoonright {}^{k'}_{n}[\Phi']^{i} \text{ (by induction)}$$

By the quotient operator and the reduction relation, we have:

$$\frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}} = \mathtt{mov}^{k}_{n}.\mathsf{R} \text{ and } {}^{k}_{n}[\mathsf{P}']^{i} \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}} \rightarrow^{*} {}^{k'}_{n}[\mathsf{S}]^{i} \quad \mathsf{R}, \mathsf{S} \in \mathcal{P} \quad \textbf{(9.1)}$$

Since domains are uniquely identified, we have:

$$\frac{\Psi}{\mathsf{Q}} \neq \mathtt{mov}^{-}_{n}.\mathsf{T} \quad \mathsf{T} \in \mathcal{P} \qquad \textbf{(9.2)}$$

$$({}^{k}_{n}[\mathsf{P}']^{i} \upharpoonright \frac{{}^{k'}_{n}[\Phi]^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}}) \upharpoonright \frac{\Psi}{\mathsf{Q}} = {}^{k'}_{n}[\mathsf{S}]^{i} \upharpoonright \frac{\Psi}{\mathsf{Q}} \text{ (9.2.1)}$$

By (9.2), there is no possible reduction for (9.2.1). We then have:

$$({}^{k'}_{n}[\mathsf{S}]^{i} \upharpoonright \frac{\Psi}{\mathsf{Q}})_{\Downarrow} \text{ (9.2.2)}$$

By (9.2.1) and(9.2.2) we have:

$$({}^{k}_{n}[\mathsf{P}']^{i} \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}}) \upharpoonright \frac{\Psi}{\mathsf{Q}} \equiv {}^{k}_{n}[P']^{i} \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}} \textbf{ (9.3)}$$

$$({}^{k}_{n}[\mathsf{P}']^{i} \upharpoonright \frac{\Psi}{\mathsf{Q}})_{\Downarrow} \quad (by\ 9.2)$$

$${}^{k}_{n}[\mathsf{P}']^{i} \upharpoonright \frac{\Psi}{\mathsf{Q}} \equiv {}^{k}_{n}[\mathsf{P}']^{i} \qquad (by\ 2.32)$$

$$({}^{k}_{n}[\mathsf{P}']^{i} \upharpoonright \frac{\Psi}{\mathsf{Q}}) \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}} \equiv {}^{k}_{n}[\mathsf{P}']^{i} \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}} \textbf{ (9.4)}$$

Since domains are uniquely identified, we have:

$$Q \neq {}^{k}_{n}[U]^{i} \quad U \in \mathcal{P}$$

By the quotient operator, (9.1) and (9.5), we have:

$$(Q \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[P']^{i}})_{\Downarrow} \quad (by\ 2.32)$$

$$(Q \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[P']^{i}}) \upharpoonright \frac{\Psi}{Q} \equiv Q \upharpoonright \frac{\Psi}{Q} \qquad (\mathbf{9.5})$$

Since domains are uniquely identified, we have:

$$Q \upharpoonright \frac{\Psi}{Q} \neq {}^{k}_{n}[V]^{i} \quad V \in \mathcal{P}$$

$$((Q \upharpoonright \frac{\Psi}{Q}) \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[P']^{i}})_{\Downarrow} \qquad (by\ 9.1, 2.32)$$

$$(Q \upharpoonright \frac{\Psi}{Q}) \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[P']^{i}} \equiv Q \upharpoonright \frac{\Psi}{Q} \qquad (\mathbf{9.6})$$

$$P \upharpoonright X \rightarrow ({}^{k}_{n}[P']^{i} \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[P']^{i}})|_{\gamma} (Q \upharpoonright \frac{\Psi}{Q}) \vDash {}^{k'}_{n}[\Phi']^{i} \mid \Psi \qquad (by\ 9.3\ -\text{-}\ 9.6, 6.5, 6.7)$$

☑ $\Phi = \Phi' \vee \Psi$:

$$X = \frac{\Phi' \vee \Psi}{P} = \frac{\Phi'}{P} + \frac{\Psi}{P} \qquad (by\ 8.8)$$

$$P \upharpoonright (\frac{\Phi'}{P} + \frac{\Psi}{P}) \equiv (P \upharpoonright \frac{\Phi'}{P}) + (P \upharpoonright \frac{\Psi}{P}) \qquad (by\ 2.29)$$

$$P \upharpoonright \frac{\Phi'}{P} \vDash \Phi' \quad \text{and} \quad P \upharpoonright \frac{\Psi}{P} \vDash \Psi \qquad (by\ hypothesis)$$

$$P \upharpoonright \frac{\Phi'}{P} \vDash \Phi' \vee \Psi \quad \text{and} \quad P \upharpoonright \frac{\Psi}{P} \vDash \Phi' \vee \Psi \qquad (by\ 6.6)$$

$$(P \upharpoonright \frac{\Phi'}{P}) + (P \upharpoonright \frac{\Psi}{P}) \vDash \Phi' \vee \Psi \quad (by\ 6.6)$$

☑ $\Phi = {}^{k}_{n}[\Psi]^{i}$:

✓ $P = {}^{k}_{n}[Q]^{i}$

$$X = \frac{{}^{k}_{n}[\Psi]^{i}}{{}^{k}_{n}[Q]^{i}} = \mathtt{mov}^{k}_{n}.\frac{\Psi}{Q} \ (by\ 8.10)$$

$${}^{k}_{n}[\mathsf{Q}]^{i} \upharpoonright \mathtt{mov}^{k}_{n}.\frac{\Psi}{\mathsf{Q}} \xrightarrow{\mathtt{mov}^{k}_{n}} {}^{k}_{n}[\mathsf{Q} \upharpoonright \frac{\Psi}{\mathsf{Q}}]^{i} \quad (\textit{by } 4.13)$$

$$\mathsf{Q} \upharpoonright \frac{\Psi}{\mathsf{Q}} \vDash \Psi \qquad (\textit{by hypothesis})$$

$${}^{k}_{n}[\mathsf{Q} \upharpoonright \frac{\Psi}{\mathsf{Q}}]^{i} \vDash {}^{k}_{n}[\Psi]^{i} \qquad (\textit{by } 6.7)$$

✓ $\mathsf{P} = {}^{k'}_{n}[\mathsf{Q}]^{i}, k' \neq k$

$$\mathsf{X} = \frac{{}^{k}_{n}[\Psi]^{i}}{{}^{k'}_{n}[\mathsf{Q}]^{i}} = \mathtt{mov}^{k'}_{n}.\mathtt{prot}^{k}_{n}.\frac{\Psi}{\mathsf{Q}} \qquad (\textit{by } 8.11)$$

$${}^{k'}_{n}[\mathsf{Q}]^{i} \upharpoonright \mathtt{mov}^{k'}_{n}.\mathtt{prot}^{k}_{n}.\frac{\Psi}{\mathsf{Q}} \xrightarrow{\mathtt{mov}^{k'}_{n}} {}^{k'}_{n}[\mathsf{Q} \upharpoonright \mathtt{prot}^{k}_{n}.\frac{\Psi}{\mathsf{Q}}]^{i} \quad (\textit{by } 4.13)$$

$${}^{k'}_{n}[\mathsf{Q} \upharpoonright \mathtt{prot}^{k}_{n}.\frac{\Psi}{\mathsf{Q}}]^{i} \xrightarrow{\mathtt{prot}^{k}_{n}} {}^{k}_{n}[\mathsf{Q} \upharpoonright \frac{\Psi}{\mathsf{Q}}]^{i} \qquad (\textit{by } 4.16)$$

$$\mathsf{Q} \upharpoonright \frac{\Psi}{\mathsf{Q}} \vDash \Psi \qquad (\textit{by hypothesis})$$

$${}^{k'}_{n}[\mathsf{Q} \upharpoonright \frac{\Psi}{\mathsf{Q}}]^{i} \vDash {}^{k}_{n}[\Psi]^{i} \qquad (\textit{by } 6.7)$$

References

[1] Cardelli, L., Gordon, A.D.: Mobile ambients. In: Foundations of Software Science and Computation Structures: First International Conference, FOSSACS '98, Springer-Verlag, Berlin Germany (1998)

[2] Pene, L., Adi, K.: Calculus for distributed firewall specification and verification. In: Proceedings of the 5th International Conference on Software Methodologies. SoMeT'06, IOS Press (2006) 301–315

[3] Ferrari, G., Moggi, E., Pugliese, R.: Guardians for ambient-based monitoring. F-WAN: Foundations of Wide Area Network Computing **66** (2002)

[4] Cardelli, L., Gordon, A.: Anytime, anywhere, modal logics for mobile ambients. In: In Proc. of POPL 2000. (2000) 365:377

[5] Cardelli, L., Gordon, A.D.: Ambient logic. Mathematical Structures in Computer Science (2006)

[6] Hirschkoff, D., Lozes, E., Sangiorgi, D.: On the expressiveness of the ambient logic. In: Logical Methods in Computer Science. (2006)

[7] Baier, C., Katoen, J.: Principles of model checking. MIT Press (2008)
[8] Walker, D.: A type system for expressive security policies. In: Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. POPL '00, New York, NY, USA, ACM (2000) 254–267
[9] Gorla, D., Pugliese, R.: Enforcing security policies via types (2003)
[10] Necula, G.C.: Proof-carrying code, ACM Press (1997) 106–119
[11] Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. **3**(1) (2000) 30–50
[12] Hamlen, K.W., Morrisett, G., Schneider, F.B.: Computability classes for enforcement mechanisms. ACM Trans. Program. Lang. Syst. **28**(1) (2006) 175–205
[13] Hamlen, K.: Security Policy Enforcement by Automated Program-rewriting. PhD thesis, Cornell University, Ithaca, NY, USA (2006) AAI3227141.
[14] Langar, M., Mejri, M., Adi, K.: Formal enforcement of security policies on concurrent systems. Journal of Symbolic Computation **3** (2011) 997–1016
[15] Sui, G., Mejri, M.: Security enforcement by rewriting: An algebraic approach. In: Foundations and Practice of Security, Springer International Publishing (2015) 311–321
[16] Bauer, L., Ligatti, J., Walker, D.: A language and system for composing security policies. Technical report, Princeton University (2004)
[17] Bauer, L.: Composing security policies with polymer. In: In Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation, ACM (2005) 305–314
[18] Clarkson, M.R., Schneider, F.B.: Hyperproperties. In: Proceedings of the 2008 21st IEEE Computer Security Foundations Symposium. CSF '08, Washington, DC, USA, IEEE Computer Society (2008) 51–65
[19] Basin, D., Jugé, V., Klaedtke, F., Zălinescu, E.: Enforceable security policies revisited. ACM Trans. Inf. Syst. Secur. **16**(1) (2013) 3:1–3:26
[20] Khoury, R., Tawbi, N.: Equivalence-preserving corrective enforcement of security properties. Int. J. Inf. Comput. Secur. **7**(2/3/4) (2015) 113–139
[21] Shields, M.W.: Implicit system specification and the interface equation. Comput. J. **32**(5) (1989) 399–412

[22] Khédri, R.: Concurrence, bisimulations et quation d'interface : une approche relationnelle. PhD thesis, Universit Laval (1998)

[23] Parrow, J.: Submodule construction as equation solving in ccs. Theoretical Computer Science **29**(1) (1989) 175–202

[24] M.T.Norris: The role of formal methods in system design. British Telecom Technical Journal (1985)

[25] Gosling, J., Joy, B., Steele, G.L.: The Java Language Specification. 1st edn. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1996)

[26] Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework 2.0. 2nd edn. Addison-Wesley Professional (2009)

[27] Fowler, A.: A swing architecture overview. SUN/Oracle (2004)

Fig. 1. Our approach

Fig. 2. Case study - Library network

Fig. 3. `PEA`: network specification

Fig. 4. `PEA`: ambient modification

Fig. 5. `PEA`: process modification

Fig. 6. `PEA`: library security policy

Fig. 7. `PEA`: library enforcement process

Table 1. Security Enforcement Calculus Syntax

n	$\in$	$\mathcal{N}$	domain name
k	$\in$	$\mathcal{K}$	security key
a	$\in$	$\mathcal{A}$	process action
i	$\subseteq$	$\mathcal{C}$	communication interface
P,Q	::=		processes
		0	deadlock
	|	1	successful termination
	|	M Act	action
	|	P.Q	sequence
	|	$P \mid_\gamma Q$	parallel composition
	|	$P \uparrow Q$	enforcement
	|	P + Q	choice
	|	!P	replication
	|	$^{k}_{n}[P]^{i}$	ambient
M	::=		action modalities
		•	preserve
	|	▲	remove
	|	▼	insert
Act	::=		process capabilities
		$\mathtt{mov}^{k}_{n}$	movement
	|	$\mathtt{prot}^{k}_{n}$	protection
	|	a	other actions

Table 2. Structural Congruence

$P \equiv P$	(2.1)
$P \equiv Q \Rightarrow Q \equiv P$	(2.2)
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	(2.3)
$P \equiv Q \Rightarrow R.P \equiv R.Q$	(2.4)
$P \equiv Q \Rightarrow P.R \equiv Q.R$	(2.5)
$P \equiv Q \Rightarrow P \mid_{\gamma} R \equiv Q \mid_{\gamma} R$	(2.6)
$(P \mid_{\gamma} Q) \mid_{\gamma} R \equiv P \mid_{\gamma} (Q \mid_{\gamma} R)$	(2.7)
$P \equiv Q \Rightarrow P + R \equiv Q + R$	(2.8)
$P \equiv Q \Rightarrow !P \equiv !Q$	(2.9)
$P \equiv Q \Rightarrow R \uparrow P \equiv R \uparrow Q$	(2.10)
$P \equiv Q \Rightarrow P \uparrow R \equiv Q \uparrow R$	(2.11)
$P \equiv Q \Rightarrow {}^{k}_{n}[P]^{i} \equiv {}^{k}_{n}[Q]^{i}$	(2.12)
$0.P \equiv 0$	(2.13)
$P \uparrow 0 \equiv 0$	(2.14)
$P + 0 \equiv P$	(2.15)
$1.P \equiv P \equiv P.1$	(2.16)
$P \mid_{\gamma} 1 \equiv P$	(2.17)
$P \uparrow 1 \equiv P \equiv 1 \uparrow P$	(2.18)
$P + 1 \equiv P$	(2.19)
$P.(Q + R) \equiv P.Q + P.R$	(2.20)
$(P + Q).R \equiv P.R + Q.R$	(2.21)
$P \uparrow Q.R \equiv (P \uparrow Q) \uparrow R$	(2.22)
$P \mid_{\gamma} Q \equiv Q \mid_{\gamma} P$	(2.23)
$P \mid_{\gamma} (Q + R) \equiv (P \mid_{\gamma} Q) + (P \mid_{\gamma} R)$	(2.24)
$P + P \equiv P$	(2.25)
$P + Q \equiv Q + P$	(2.26)
$!P \equiv P \mid_{\gamma} !P$	(2.27)
$P \uparrow (Q \mid_{\gamma} R) \equiv (P \uparrow Q) \uparrow R + (P \uparrow R) \uparrow Q$	(2.28)
$P \uparrow (Q + R) \equiv (P \uparrow Q) + (P \uparrow R)$	(2.29)
$(P + Q) \uparrow R \equiv (P \uparrow R) + (Q \uparrow R)$	(2.30)
$(P \mid_{\gamma} Q) \uparrow R \equiv (P \uparrow R) \mid_{\gamma} (Q \uparrow R)$	(2.31)

Table 3. Termination

$1\downarrow$	(3.1)
$\frac{P\downarrow}{(P+Q)\downarrow}$	(3.2)
$\frac{Q\downarrow}{(P+Q)\downarrow}$	(3.3)
$\frac{P\downarrow, Q\downarrow}{(P.Q)\downarrow}$	(3.4)
$\frac{P\downarrow, Q\downarrow}{(P\mid_{\gamma} Q)\downarrow}$	(3.5)
$\frac{P\downarrow}{{}^{k}_{n}[P]^{i}\downarrow}$	(3.6)

Table 4. Reduction Relation

$$\frac{P' \equiv P, P \xrightarrow{a} Q, Q \equiv Q'}{P' \xrightarrow{a} Q'} \quad (4.1)$$

$$\frac{}{a \xrightarrow{a} 1} \quad (4.9)$$

$$\frac{P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'} \quad (4.2)$$

$$\frac{}{a.P \uparrow \bullet a.Q \xrightarrow{\bullet a} a.(P \uparrow Q)} \quad (4.10)$$

$$\frac{P \xrightarrow{a} P'}{P.Q \xrightarrow{a} P'.Q} \quad (4.3)$$

$$\frac{}{a.P \uparrow \blacktriangle a.Q \xrightarrow{\blacktriangle a} P \uparrow Q} \quad (4.11)$$

$$\frac{P \xrightarrow{a} P'}{P \mid_{\gamma} Q \xrightarrow{a} P' \mid_{\gamma} Q} \quad (4.4)$$

$$\frac{}{P \uparrow \blacktriangledown a.Q \xrightarrow{\blacktriangledown a} a.(P \uparrow Q)} \quad (4.12)$$

$$\frac{Q \xrightarrow{a} Q'}{P \uparrow Q \xrightarrow{a} P \uparrow Q'} \quad (4.5)$$

$$\frac{k' \geq k}{{}^{k}_{n}[P]^{i} \ddagger mov^{k'}_{n}.Q \xrightarrow{mov^{k'}_{n}} {}^{k}_{n}[P \ddagger Q]^{i}} \quad (4.13)$$

$$\frac{P \xrightarrow{a} P', Q \downarrow}{P \uparrow Q \xrightarrow{a} P'} \quad (4.6)$$

$$\frac{k' \geq k}{P \ddagger {}^{k}_{n}[mov^{k'}_{n}.Q \mid_{\gamma} R]^{i} \xrightarrow{mov^{k'}_{n}} P \ddagger Q \mid {}^{k}_{n}[R]^{i}} \quad (4.14)$$

$$\frac{P \xrightarrow{a} P', Q \xrightarrow{b} Q'}{P \mid_{\gamma} Q \xrightarrow{\gamma(a,b)} P' \mid_{\gamma} Q'} \quad (4.7)$$

$$\frac{k'' \geq k, k'' \geq k', i \cap j \neq \emptyset}{{}^{k}_{n}[P]^{i} \ddagger {}^{k'}_{m}[mov^{k''}_{n}.Q \mid_{\gamma} R]^{j} \xrightarrow{mov^{k''}_{n}} {}^{k}_{n}[P \ddagger Q]^{i} \ddagger {}^{k'}_{m}[R]^{j}} \quad (4.15)$$

$$\frac{P \xrightarrow{a} P'}{{}^{k}_{n}[P]^{i} \xrightarrow{a} {}^{k}_{n}[P']^{i}} \quad (4.8)$$

$$\frac{}{{}^{k}_{n}[P \uparrow prot^{k'}_{n}.Q]^{i} \xrightarrow{prot^{k'}_{n}} {}^{k'}_{n}[P \uparrow Q]^{i}} \quad (4.16)$$

where $\ddagger \in \{\mid_{\gamma}, \uparrow\}$

Table 5. Syntax of the logic

Φ, Ψ	::=		
	tt		True
|	$\neg \Phi$		Negation
|	$\langle a \rangle \Phi, a \in \mathcal{A}$		Capability
|	$\Phi \mid \Psi$		ParallelComposition
|	$\Phi \vee \Psi$		Disjunction
|	${}^{k}_{n}[\Phi]^{i}$		Protectedlocation

Table 6. Semantics of the logic

$[\![tt]\!]$	=	$\mathcal{P} \setminus \{0\}$	(6.1)
$[\![ff]\!]$	=	0	(6.2)
$[\![\neg \Phi]\!]$	=	$\mathcal{P} \setminus [\![\Phi]\!]$	(6.3)
$[\![\langle a \rangle \Phi]\!]$	=	$\{a.P \in \mathcal{P} : P \in [\![\Phi]\!]\}$	(6.4)
$[\![\Phi \mid \Psi]\!]$	=	$\{P \mid Q \in \mathcal{P} : P \in [\![\Phi]\!] \wedge Q \in [\![\Psi]\!]\}$	(6.5)
$[\![\Phi \vee \Psi]\!]$	=	$[\![\Phi]\!] \cup [\![\Psi]\!]$	(6.6)
$[\![{}^{k}_{n}[\Phi]^{i}]\!]$	=	$\{{}^{k}_{n}[P]^{i} \in \mathcal{P} : P \in [\![\Phi]\!]\}$	(6.7)

Table 7. The elimination of the $\neg\Phi$ form

$\neg tt$	$\rightarrow$	ff	(7.1)
$\neg ff$	$\rightarrow$	tt	(7.2)
$\neg\neg \Phi$	$\rightarrow$	Φ	(7.3)
$\neg(\langle a \rangle \Phi)$	$\rightarrow$	$(\neg\langle a \rangle) tt \vee \langle a \rangle \neg \Phi$	(7.4)
$\neg(\Phi \mid \Psi)$	$\rightarrow$	$(tt \mid \neg \Psi) \vee (\neg \Phi \mid tt) \vee (\neg \Phi \mid \neg \Psi)$	(7.5)
$\neg(\Phi \vee \Psi)$	$\rightarrow$	$\neg \Phi \wedge \neg \Psi$	(7.6)
$\neg({}^{k}_{n}[\Phi]^{i})$	$\rightarrow$	${}^{k}_{n}[\neg \Phi]^{i}$	(7.7)

Table 8. Quotient Operator

$\overline{\overline{/}}: \mathcal{F} \times (\mathcal{P} \setminus \{0\}) \to \mathcal{P}$

$$\frac{tt}{\mathsf{P}} = 1 \qquad (8.1)$$

$$\frac{f\,f}{\mathsf{P}} = 0 \qquad (8.2)$$

$$\frac{\langle \mathtt{a} \rangle \Phi}{\mathtt{a}.\mathsf{P}} = \bullet \mathtt{a}.\frac{\Phi}{\mathsf{P}} \qquad (8.3)$$

$$\frac{\langle \mathtt{a} \rangle \Phi}{\mathtt{b}.\mathsf{P}} = \blacktriangle b.\frac{\Phi}{P}.\blacktriangledown a \qquad a \neq b \qquad (8.4)$$

$$\frac{\neg \langle \mathtt{a} \rangle \Phi}{\mathtt{a}.\mathsf{P}} = \blacktriangle \mathtt{a}.\frac{\Phi}{\mathsf{P}} \qquad (8.5)$$

$$\frac{\neg \langle \mathtt{a} \rangle \Phi}{\mathtt{b}.\mathsf{P}} = \bullet \mathtt{b}.\frac{\Phi}{\mathsf{P}} \qquad \mathtt{a} \neq \mathtt{b} \qquad (8.6)$$

$$\frac{\langle \mathtt{a} \rangle \Phi}{1} = \frac{\Phi}{1}.\blacktriangledown \mathtt{a} \qquad (8.7)$$

$$\frac{\Phi \vee \Psi}{\mathsf{P}} = \frac{\Phi}{\mathsf{P}} + \frac{\Psi}{\mathsf{P}} \qquad (8.8)$$

$$\frac{\Phi}{\mathsf{P} + \mathsf{Q}} = \frac{\Phi}{\mathsf{P}} + \frac{\Phi}{\mathsf{Q}} \qquad (8.9)$$

$$\frac{{}^{k}_{n}[\Phi]^{i}}{{}^{k}_{n}[\mathsf{P}]^{i}} = mov^{k}_{n}.\frac{\Phi}{\mathsf{P}} \qquad (8.10)$$

$$\frac{{}^{k'}_{n}[\Phi]^{i}}{{}^{k}_{n}[\mathsf{P}]^{i}} = \mathtt{mov}^{k}_{n}.\mathtt{prot}^{k'}_{n}.\frac{\Phi}{\mathsf{P}} \qquad k \neq k' \qquad (8.11)$$

$$\frac{{}^{k'}_{n}[\Phi]^{i} \mid \Psi}{{}^{k}_{n}[\mathsf{P}]^{i} \mid_{\gamma} \mathsf{Q}} = \frac{{}^{k'}_{n}[\Phi]^{i}}{{}^{k}_{n}[\mathsf{P}]^{i}} \mid_{\gamma} \frac{\Psi}{\mathsf{Q}} \qquad (8.12)$$