# Quantitative Evaluation of Enforcement Strategies

Vincenzo Ciancia[1], Ilaria Matteucci[1], and Charles Morisset[2]

[1] CNR-IIT Via Moruzzi 1, 56124, Pisa, Italy firstname.lastname@iit.cnr.it
[2] School of Computing Science Newcastle University NE17RU, UK
charles.morisset@ncl.ac.uk

**Abstract.** A *security enforcement mechanism* runs in parallel with a system to check and modify its run-time behaviour, in order to guarantee the satisfaction of a security policy. For a given policy, several enforcement strategies are possible, usually reflecting several trade-offs one has to make to satisfy the policy. We provide a formal framework for the quantification of enforcement strategies, by composing the enforcement mechanism with a process monitoring a user-defined quantity. This is done by extending the notion of *controller process*, that mimics the well-known edit automata, with weights on transitions, valued in a $C$-semiring. We demonstrate with some examples how we leverage the flexibility and generality of $C$-semirings to compare controllers according to multiple dimensions, such as security and/or cost, and we propose different evaluation strategies.

**Keywords:** Process Algebra, Enforcement Mechanism, Quantity Monitoring, Formal Analysis.

## 1 Introduction

Security is often regarded as a binary concept. Behaviour is either good or bad. Good behaviour is either enforced or not. Still, in many cases we do not deal with just two security values. Reality is much more complex and may force us to consider several quantitative aspects that play a role in the design and evaluation of enforcement strategies. For example, a controller can have a higher financial impact than another one in enforcing a specific security policy, or the benefits, in a certain domain, of enforcing a specific policy, may fail to counter-balance the disadvantages in a different domain.

Proper analysis of the situation paves the way to complex decisions in the controlling phase that take several parameters into account. In this work, we deal with such quantitative aspects of security, and especially of the monitoring and enforcement of security policies.

There is a significant bulk of work devoted to the study of enforcement strategies. Several formal models have been defined during the last decade. We recall as foremost examples security automata [29] that are only capable of preventing bad executions, and edit automata [2], capable of inserting, suppressing, and editing their output sequences. Other approaches make use of concurrent languages (such as process algebras), that model both the target and the control system in the same formalism [19,20,15].

One can consider several quantitative dimensions, such as cost, time, risk, or even trust. All these different domains demand for a parametric approach when modelling

targets and controllers. We address this aspect by using $C$-semirings, that are widely adopted as a domain for optimization problems [4]. $C$-semirings account for multi-dimensional valuations, and establish a partial order on values. We chose a variant of the quantitative process algebra GPA [7] as the language to express controllers and targets. This process algebra provides CSP-style synchronization, and actions weighted over a semiring. After providing the necessary background about $C$-semirings and GPA, in Section 2 we define the quantitative evaluation of processes and their traces. The goal of the paper is to provide a quantitative evaluation of enforcement processes to compare different strategies according to several measures. We make a distinction between monitors and controllers: monitors (Section 3) can be used to associate quantities to an existing system without changing its behaviour; controllers (Section 4) are able to modify the behaviour of a target by using *control actions* for suppressing or inserting possible incorrect actions. Our framework is conservative, in the sense that a classical boolean security policy can be monitored using the *boolean $C$-semiring*. Finally, in Section 5 we study the formal grounds of quantitative evaluation and comparisons of controllers. This permits one to compare different strategies in terms of their evaluations with respect to different measures, *e.g.*, security, cost, trust, energy, and so on. Multi-dimensional criteria are indeed possible in the general framework of $C$-semirings.

To sum up, the main contribution of this paper is a parametric framework within which a security designer can model a system, some quantities to monitor and evaluate different controllers with respect to these quantities, thus permitting her to reason about the trade-off among different measures. The use of semirings allows one to combine different domains, while the quantitative process algebra that we use enables one to reason in a compositional manner.

## 2 Quantitative Processes

A quantitative process is a process where each transition is labelled with some quantity. We first introduce the notion of $C$-semiring, and then present an algebra for quantitative processes.

**Definition 1.** *A* $C$-semiring *is a tuple* $\mathbb{K} = (K, \sum, *, \mathbf{0}, \mathbf{1})$ *where* $K$ *is a set, with* $\mathbf{0}$ *and* $\mathbf{1}$ *elements of* $K$. $\sum : \mathcal{P}(K) \to K$ *with* $\sum\{a\} = a$, $\sum \emptyset = \mathbf{0}$, $\sum K = \mathbf{1}$, $\sum(\bigcup K_i) = \sum\{\sum K_i\}$ *for* $K_i \subseteq K, i \geq 0$ *and* $* : K \times K \to K$ *is commutative, associative and distributive property over* $\sum$; $\mathbf{1}$ *is its unit element, and* $\mathbf{0}$ *is its absorbing element.*

We write $k_1 + k_2$ for $\sum\{k_1, k_2\}$, making $+$ a commutative, associative and idempotent operator. Many examples of $C$-semirings can be found in the literature (*e.g.*, [4]), such as: the *Boolean* $C$-semiring $\mathbb{K}_B = \langle\{\mathit{true}, \mathit{false}\}, \vee, \wedge, \mathit{false}, \mathit{true}\rangle$, for logical values and operations; the *Cost* $C$-semiring $\mathbb{K}_C = \langle\mathbb{R}_0^+, min, +, +\infty, 0\rangle$; the *Trust* $C$-semiring $\mathbb{K}_T = \langle[0,1], max, min, 0, 1\rangle$. We also introduce a special *dummy* $C$-semiring $\mathbb{K}_D = \langle\{0\}, \sum, *, 0, 0\rangle$, where $\sum \emptyset = \sum\{0\} = 0$ and $*$ is the traditional arithmetic multiplication. Since any value in this $C$-semiring equals 0, it is easy to see that it satisfies all required properties. We use this $C$-semiring when considering composition of $C$-semirings. We write $\mathbb{K}_1 \equiv \mathbb{K}_2$ when $\mathbb{K}_1$ and $\mathbb{K}_2$ are isomorphic, and the results described hereafter are given modulo isomorphism between semirings.

Every $C$-semiring is endowed with a partial order $\sqsubseteq$, such that $k_1 \sqsubseteq k_2$ if, and only if $k_1 + k_2 = k_2$. This partial order intuitively indicates a notion of preference, such that $k_1 \sqsubseteq k_2$ can be read as $k_2$ is "better" than $k_1$.

Complex $C$-semirings can be obtained by composition. For instance, the cartesian product of $C$-semirings is a $C$-semiring, having elements in the cartesian product of the two sets of values, and operations defined in a point-wise way. Further compositions techniques exist, such as the lexicographic $C$-semiring [27], the expectation semiring [17], etc.

**Definition 2.** *A* composition operator *over $C$-semirings is a function $\odot$ satisfying, for any $C$-semiring $\mathbb{K}$, $\mathbb{K}_D \odot \mathbb{K} \equiv \mathbb{K}$.*

In most cases, the support set of $\mathbb{K}_1 \odot \mathbb{K}_2$ is $K_1 \times K_2$, the additive and multiplicative operators can however change from one composition to another. Given two values $k_1 \in K_1$ and $k_2 \in K_2$ we write $k_1 \odot k_2$ for the corresponding composite value in the support set of $\mathbb{K}_1 \odot \mathbb{K}_2$.

*Example 1.* Given $\mathbb{K}_1 = \langle K_1, \sum_1, *_1, \mathbf{0}_1, \mathbf{1}_1 \rangle$ and $\mathbb{K}_2 = \langle K_2, \sum_2, *_2, \mathbf{0}_2, \mathbf{1}_2 \rangle$, the lexicographic composition operator $\odot_L$ is defined such that $\mathbb{K}_1 \odot_L \mathbb{K}_2 = \langle K_3, \sum_3, *_3, \mathbf{0}_3, \mathbf{1}_3 \rangle$, where $K_3 = K_1 \times K_2$, $\mathbf{0}_3 = (\mathbf{0}_1, \mathbf{0}_2)$, $\mathbf{1}_3 = (\mathbf{1}_1, \mathbf{1}_2)$, and for any $k_1, k_1' \in K_1$ and any $k_2, k_2' \in K_2$:

$$(k_1, k_2) +_3 (k_1', k_2') = \begin{cases} (k_1, k_2) & \text{if } (k_1' \sqsubseteq_1 k_1 \text{ and } k_1 \neq k_1') \text{ or } (k_1 = k_1' \text{ and } k_2' \sqsubseteq_2 k_2) \\ (k_1', k_2') & \text{otherwise.} \end{cases}$$

$$(k_1, k_2) *_3 (k_1', k_2') = (k_1 *_1 k_1', k_2 *_2 k_2')$$

Along this paper we describe the behaviour of controllers and targets in terms of process algebras and semirings. In particular we use the *Generalized Process Algebra* (GPA), introduced in [7], in order to specify quantitative aspects of process transitions.

**Definition 3.** *The set $\mathcal{L}$ of* agents*, or processes, in GPA over a set of finite transition labels Act and a $C$-semiring $\mathbb{K}$ is defined by the grammar*

$$A ::= 0 \mid (a, k).A \mid A + A \mid A \|_S A \mid X$$

*where $a \in Act$, $k \in K$, $S \subseteq Act$, and $X$ belongs to a countable set of* process variables*. We write $\mathsf{GPA}[\mathbb{K}]$ for the set of GPA processes labelled with weights in $\mathbb{K}$.*

Our definition of process diverges from the original formulation of GPA in two main points. Firstly, we use $C$-semirings instead of semirings, since we need to consider infinite sums, quantified over the (infinite) set of traces of a process. In semirings, the additive operation is binary, so they are defined (inductively) only on finite sets. Secondly, we allow for transitions with weight $\mathbf{0}$. We are indeed interested in distinguishing between processes that do not perform a given action and processes that perform it with weight $\mathbf{0}$. For instance, if the weight measures the satisfaction of a security policy, we want to consider processes that actually violate it.

The definition of the syntax adopts typical constructs of process algebras [24,28]. We assume a defining equation $X \triangleq A$ for each variable appearing in a term, and we

**Table 1.** Operational semantics for $GPA$ [7].

$$\frac{}{(a,k).A \xrightarrow{a,k} A} \qquad \frac{A \xrightarrow{a,k} A_1 \quad A' \xrightarrow{a,l} A_1'}{A\|_S A' \xrightarrow{a,k*l} A_1\|_S A_1'} a \in S \qquad \frac{A \xrightarrow{a,k} A_1}{X \xrightarrow{a,k} A_1} X \triangleq A$$

$$\frac{A \xrightarrow{a,k} A_1}{A\|_S A' \xrightarrow{a,k} A_1\|_S A} a \notin S \qquad \frac{A_j \xrightarrow{a,k} A_1}{\sum_{i\in I} A_i \xrightarrow{a,k_\Sigma} A_1} j \in I \qquad \frac{A' \xrightarrow{a,k} A_1'}{A\|_S A' \xrightarrow{a,k} A\|_S A_1'} a \notin S$$

where $k_\Sigma = \sum_{i\in I}(A_i \xrightarrow{a} A_1)$

assume that processes have *guarded recursion*, that is, all the variables in a process are directly under some prefix. We omit the hiding operator from GPA as we will introduce more fine-grained control operators in Section 4. The intuitive meaning of the operators is as follows: 0 describes the termination of a process; $(a,k).A$ performs the action $a$ with a certain *weight* $k$ and then behaves as $A$; $A + A'$ non-deterministically behaves as either $A$ or $A'$; $A\|_S A'$ describes the process in which $A$ and $A'$ proceed concurrently and independently on all actions which are not in $S$. All the action in $S$ are performed if and only if both the process perform the same action in $S$ at the same time. The formal semantics of operators (Table 1) is expressed in terms of *a multiple labelled transition system* ($MLTS$ for short). Similarly to weighted automata [13], MLTSs are labelled transition systems with weights on labels, that we define below. Notice that there is no need to maintain a distinction between GPA processes and states of the corresponding MLTS, as they coincide (which is typical in process calculi).

**Definition 4.** *A multi labelled transition system (MLTS) is a tuple*[3] $(S, Act, \mathbb{K}, \delta)$*, where* $S$ *is the state space which is finite or countably infinite,* $Act$ *is a finite set of transition labels,* $\mathbb{K}$ *is a semiring used for the definition of transition weights, and* $\delta : S \times Act \times S \rightharpoonup \mathbb{K}$ *is the partial transition function.*

*Remark 1.* The definition of an MLTS syntactically resembles that of a *weighted automaton* [13]. Semantically speaking, weighted automata denote (weighted) languages, that is, *properties*, whereas MLTSs are *models*, similarly to the case of classical automata and labelled transition systems.

In run-time enforcement, one should not tell processes apart by their internal choices, but rather by their execution traces. In this light, we define quantitative evaluation of predicates over agents. In the following, let $A$ be an agent and consider the corresponding MLTS with set of states $S$.

**Definition 5.** *Given states* $s_1, s_2 \in S$*, we say that* $t \in (Act \times K)^*$ *is a* path *from* $s_1$ *to* $s_2$*, and write* $s_1 \overset{t}{\Rightarrow} s_2$*, if either* $t$ *is empty, and* $s_1 = s_2$*, or* $t = (a,k).t'$*, and there is*

[3] In [7], an *initialization* function is also taken into account, assigning an initial quantitative valuation to each state. In the current paper we do not need it, thus we simplify the presentation.

$s' \in S$ *such that* $s_1 \overset{(a,k)}{\rightarrow} s'$*, and* $t'$ *is a path from* $s'$ *to* $s_2$*. We let* $\mathcal{T}(s)$ *be the set of paths from* $s$ *to some other state, and we write* $\mathcal{T}(A)$ *to denote the set of paths of a process* $A$*.*

**Definition 6.** *Given a path* $t = (a_1, k_1) \cdots (a_n, k_n)$*, the* label *of* $t$ *is given by* $l(t) = a_1 \cdots a_n \in Act^*$*, and its* run weight *by* $|t| = k_1 * \ldots * k_n \in K$*.*

The *valuation* of a process intuitively corresponds to the best possible quantity of this process.

**Definition 7.** *Given a process* $A$*, the valuation of* $A$ *s is given by* $[\![A]\!]$*, such that*

$$[\![A]\!] = \sum_{\{t \in \mathcal{T}(A)\}} |t|$$

Finally, without any loss of generality, we consider any unlabelled process as labelled with the *dummy* $C$-semiring $\mathbb{K}_D$.

## 3 Quantitative monitor operators

Definition 7 assumes that the considered process is already labelled with some quantities. However, a system, hereafter named *target*, does not always come with the quantities we are interested in evaluating, and might even be not labelled at all. Hence, in the most general case, the security designer must provide a *labelling function* $\lambda : \mathsf{GPA}[\mathbb{K}_1] \rightarrow \mathsf{GPA}[\mathbb{K}_2]$, such that given any process $A$ labelled in $\mathbb{K}_1$, $\lambda(A)$ represents the process $A$ labelled with a quantity in $\mathbb{K}_2$. A simple example is the function $\lambda_v$, which assign any transition with the value $v \in \mathbb{K}_2$, thus erasing any previous quantity.

In practice, the responsibility of measuring a particular aspect is often delegated to a *monitor*, which probes the system and indicates the weight of each operation. In terms of security, a monitor is usually passive, i.e., it does not effectively modify the behaviour of the considered target. This means that it does not prevent violation of a security policy. On the other hand, a *controller* is able to modify the behaviour of a target in order to guarantee security requirements. A security monitor and a security controller are often merged into a single entity, responsible both for deciding whether an action would violate the policy and what corrective action should be taken if necessary. We propose here to make an explicit distinction between these two processes and to extend the monitoring to measures other than security. In this section we investigate quantitative monitors. Controllers are detailed in Section 4.

Intuitively, a monitor measures a quantity not already present in the monitored target. Since the target might be already equipped with some quantities, coming for instance from another monitor, we need to *merge* the quantities from the monitor with those of the target. As said in Section 2, there is not a single way to compose $C$-semirings together, and therefore, the merge requires a composition operator $\odot$.

**Definition 8.** *Given a process* $A$ *labelled with* $\mathbb{K}$*, a process* $A'$ *labelled with* $\mathbb{L}$ *and a composition operator* $\odot$*, we write* $A \odot A'$ *for the merged process, semantically defined as:*

$$\frac{A \overset{a,k}{\rightarrow} A_1 \quad A' \overset{a,l}{\rightarrow} A'_1}{A \odot A' \overset{a,k \odot l}{\rightarrow} A_1 \odot A'_1}$$

A merged process can only move on when both of its components can move on with the same action.

We are now able to define a monitor, which is a process that can be composed with a target without affecting its behaviour.

**Definition 9.** *Given a composition operator $\odot$ and a process $A$, a process $M$ is a monitor for $A$ if and only if $\{l(t) \mid t \in \mathcal{T}(A \odot M)\} = \{l(t) \mid t \in \mathcal{T}(A)\}$.*

Given any process $A$ labelled with $\mathbb{K}_1$, any monitor $M$ for $A$ labelled with $\mathbb{K}_2$ and any composition operator $\odot$, we can define the labelling function $\lambda : \mathsf{GPA}[\mathbb{K}_1] \to \mathsf{GPA}[\mathbb{K}_1 \odot \mathbb{K}_2]$ as $\lambda(A) = A \odot M$. It is worth observing that given any unlabelled process $A$, i.e., a process in $\mathsf{GPA}[\mathbb{K}_\mathsf{D}]$, and a monitor $M$ labelled in $\mathbb{K}_2$, we have $A \odot M \in \mathsf{GPA}[\mathbb{K}_2]$, since from Definition 2, we have $\mathbb{K}_D \odot \mathbb{K}_2 \equiv \mathbb{K}_2$. In other words, unlabelled systems can be monitored in an equivalent way than systems already labelled.

*Example 2.* Let us consider the simple case of a process $A$ with no existing weight and with an alphabet $\Sigma = \{a, b\}$. We want to define an energy monitor using the semiring $\mathbb{K}_C$, such that the action $a$ consumes 3 units, and the action $b$ consumes $2n$ units, where $n$ is the number of times $b$ has been performed (i.e., $b$ has an increasing energy cost). Hence, for $n > 0$, we define the monitor:

$$M_n = (a, 3).M_n + (b, 2n).M_{n+1}$$

For instance, the process $A = a.b.b.a.b$ can be monitored with[4]

$$A \odot M_1 = (a, 3).(b, 2).(b, 4).(a, 3).(b, 6)$$

The valuation of the monitored process corresponds to the total energy consumed, i.e., $[\![A \odot M_1]\!] = 18$. Similarly, the monitored process of $B = a + b$ is $B \odot M_1 = (a, 3) + (b, 2)$, and its valuation $[\![B \odot M_1]\!] = 2$, since the valuation returns the best possible quantity. Finer-grained approaches can be used to get the valuation of a process, as discussed in Section 7.

*Example 3.* Let us now consider a security policy $P$ defined as a predicate on traces. For instance, in the previous example, consider a policy stating that the action $a$ cannot be performed before $b$. Using the boolean semiring $\mathbb{K}_B$, we define the following monitor:

$$M_P = (b, true).M'_P + (a, false).M''_P$$
$$M'_P = (b, true).M'_P + (a, true).M'_P$$
$$M''_P = (b, false).M''_P + (a, false).M''_P$$

Roughly speaking, $M_P$ observes the first action: if the target executes $b$, then any following action is secure, otherwise any following action is not secure. In order to monitor both security and energy, we consider the lexicographic composition $\odot_L$ defined in Example 1. In other words, a secure trace is always better than a non-secure one, and two traces equally (non)secure are compared based on their energy. Hence, given the process $A = a + b.a$, its monitored version is given by $M_P \odot (A \odot M_1) = (a, false, 3) + (b, true, 2).(a, true, 3)$, and we have $[\![M_P \odot_L (A \odot M_1)]\!] = (true, 5)$.

[4] In the following, the terminating process 0 is omitted when obvious from context.

Bauer et al. established in [2] than it is possible to define an enforcement mechanism for any *safety property*, i.e., any property such that an incorrect trace cannot be extended into a correct one. Therefore, we are able to state the following proposition.

**Proposition 1** *Given any safety property $P$ and any process $A$, there exists a monitor $M_P$ such that for any trace $t \in \mathcal{T}(A \odot M_P)$, $|t| = true$ if and only if $P(l(t))$.*

*Proof.* (sketch) From [2,20], we know that if $P$ is a safety property, there exists a process that works as a truncation automaton[5] $(Q, \delta, q_0)$ for enforcing $P$. We therefore define the monitor over $Q \cup \{q_\star\}$, such that, for any state $q \in Q$ and for any action $a$, $q \overset{a,true}{\rightarrow} q'$ if, and only if $q' = \delta(q, a)$, $q \overset{a,false}{\rightarrow} q_\star$ if $\delta(q, a)$ is not defined, and $q_\star \overset{a,false}{\rightarrow} q_\star$.

Clearly, finer-grained approaches can be used to monitor a security policy. For instance, in the previous example, we could consider that executing a single $a$ before a $b$ is somehow "better" than executing $a$ many times. In that case, we could use the cost semiring $\mathbb{K}_C$, and define the monitor as:

$$M_P = (b, 0).M'_P + (a, 1).M''_P \quad M'_P = (b, 0).M_P + (a, 0).M'_P$$
$$M''_P = (b, 0).M'_P + (a, 1).M''_P$$

Note that a monitor is only one possible way to build a labelling function $\lambda$. Although monitors are expressive enough for the examples we consider in this paper, more complex labelling functions may also be of interest.

*Remark 2.* By construction, the valuation of the empty process is equal to $\mathbf{0}$, since the sum of the empty set is equal to $\mathbf{0}$. For instance, when considering the quantity for a security policy, the valuation of the empty process is $false$. Although such a value makes sense for liveness properties (where an action *must* happen in order for the property to hold), it might be counter-intuitive for properties that hold on the empty trace. In order to avoid such cases, given any process $A$, it is always possible to consider the process $A_\iota$, such that $A_\iota \overset{\tau,\iota}{\rightarrow} A$, where $\iota$ represents an initialization value. This approach is intuitively similar to that of Buchholz and Kemper [7], who explicitly introduce an initialization function.

## 4 Quantitative control operators

The role of the monitor is to *detect* a policy violation, and not to *prevent* a *target* system, hereafter denoted by $F$, from doing so. For this reason it can be used, for instance, for directly evaluating a security policy $P$ as a value on each transition of a *target* process (see Example 3).

A controller $E$, just like a monitor $M$, follows target actions step by step. The difference is that $M$ observes target actions, labelling them with *true* when they obey to the policy $P$ or *false* when they attempt to violate $P$. On the contrary, the controller can decide not only to accept but also to change target traces. The resulting process is the *controlled process* $E \triangleright F$, following the semantics given in Table 2.

[5] Due to the lack of space, we do not recall the definition of the truncation automaton here, and we refer to [2] for a complete definition.

**Table 2.** MLTS rules for quantitative control operators.

$$\frac{E \xrightarrow{a,k} E' \quad F \xrightarrow{a,k'} F'}{E \triangleright F \xrightarrow{a,k*k'} E' \triangleright F'}\ (\textsc{Accept}) \qquad \frac{E \xrightarrow{\boxminus a,k} E' \quad F \xrightarrow{a,k'} F'}{E \triangleright F \xrightarrow{\tau,k*k'} E' \triangleright F'}\ (\textsc{Sup}) \qquad \frac{E \xrightarrow{\boxplus a.b,k} E' \quad F \xrightarrow{a,k'} F'}{E \triangleright F \xrightarrow{b,k} E' \triangleright F}\ (\textsc{Ins})$$

Intuitively speaking, each rule corresponds to a different controlling behaviour. The alphabets of $E$, $F$, and of the resulting process $E \triangleright F$ are different, as $E$ may perform *control actions* that regulate the actions of the target $F$, and moreover the resulting process $E \triangleright F$ may perform internal actions, denoted by $\tau$, as a consequence of suppression. From now on, we will let $Act$ be the alphabet of (the GPA describing) $F$. The alphabet of $E$ consists of symbols of the form $a$, $\boxplus a.b$, $\boxminus a$ for $a, b \in Act$, denoting respectively the actions of *acceptance*, *suppression*, and *insertion*; the alphabet of $E \triangleright F$ is $Act \cup \{\tau\}$.

The *acceptance rule* (ACCEPT) constrains the controller and the target to perform the same action, in order for it to be observed in the resulting behaviour; the observed weight is the product of those of the controller and the target. Given two processes $A$ and $B$, the semantics of truncation is equivalent to that of CSP-style parallel composition of $A$ and $B$, where synchronisation is forced over all actions of the two processes.

The *suppression rule* (SUP) allows the controller to hide actions of the target. The target *wants to* perform the action, but the action is not performed by the controlled entity and the observed result is a $\tau$ action, with the weight calculate as the product of the suppressing and the target action.

Finally, the *insertion rule* (INS) describes the capability of correcting some bad behaviour of the target, by inserting another action in its execution trace. The weight of insertion is only the weight provided by the controller; this accounts for the fact that the target does not perform any action, but rather stays in its current state, as in [2].

*Example 4.* Consider the policy of Example 3, where any trace should start with at least one action $b$. Omitting the weights for now, we can define two controllers: $E_2$ suppresses any action $a$ as long as an action $b$ has not been performed; $E_3$ inserts an action $b$ if $a$ is submitted first. Both controllers use an auxiliary controller $E_1$, that accepts any action:

$$E_1 = a.E_1 + b.E_1 \qquad E_2 = \boxminus a.E_2 + b.E_1 \qquad E_3 = \boxplus a.b.E_1 + b.E_1$$

Given the sequential target $F = a.b$, we have

$$E_2 \triangleright a.b \xrightarrow{\tau} E_2 \triangleright b \xrightarrow{b} E_1 \triangleright 0 \qquad E_3 \triangleright a.b \xrightarrow{b} E_1 \triangleright a.b \xrightarrow{a} E_1 \triangleright b \xrightarrow{b} 0$$

The process $E_3 \triangleright F$ has one more transition than $E_2 \triangleright F$, because the insertion does not change $F$.

For the sake of simplicity, we do not consider here *blocking* controllers, which prevent the target to perform any action. A classical example of blocking controller is the *truncating* controller, forbidding the target to perform any action. In the previous example, a

truncating controller could be defined by $E_4 = b.E_1$, meaning that the controlled target $E_3 \triangleright a.b$ would be simply blocked. The focus of this paper is indeed to quantify the behaviour of a controlled process, which requires that the controlling strategy has an observable effect. In practice, truncating can be achieved by a continuous suppression of any action of the target, for instance, omitting again any weight, the previous controller $E_4$ has the same observable effect (i.e., ignoring $\tau$ actions) than the following controller $E_5$:

$$E_5 = \boxminus a.E_6 + b.E_1 \qquad E_6 = \boxminus a.E_6 + \boxminus b.E_6$$

Formally, in the rest of the paper, we only consider controllers obeying the following definition.

**Definition 10.** *Given any target $F$, a controller $E$ is said to be* non-blocking*, if and only if*

$$\exists a \in Act\ F \xrightarrow{a} F' \Rightarrow \exists b \in Act \cup \{\tau\}\ E \triangleright F \xrightarrow{b} E' \triangleright F''$$

*where $E'$ is also a non-blocking controller. Note that we do not impose $F' = F''$, since $E$ might not let $F$ perform its intended action, as it is the case with the insertion rule.*

### 4.1 Soundness and Transparency

Classical properties of a controller are *soundness* and *transparency*: intuitively, a controller is sound if, and only if, any output trace is correct, i.e., it has weight $true$ in our framework, and transparent if, and only if, any correct trace of the target is not modified by the controller.

**Definition 11.** *Given a property $P$, a process $A \in \mathsf{GPA}[\mathbb{K}]$ satisfies $P$ if, and only if:*

$$\forall t \in \mathcal{T}(A \odot_c M_P)\ |t| = (v, true), \quad \textit{for some } v \in \mathbb{K}$$

*A controller $E$ is sound for $P$ if, and only if, given any target $F$, $E \triangleright F$ satisfies $P$. In addition, $E$ is said to be transparent for $P$ if, and only if, for any target $F$ satisfying $P$, we have $\mathcal{T}(F) = \mathcal{T}(E \triangleright F)$.*

The previous controllers $E_2, E_3$ and $E_5$ are sound and transparent, while $E_1$ is transparent, but not sound, and $E_6$ is sound, but not transparent. It is worth observing that given a property $P$, if $E$ is sound for $P$, then, assuming that $E$ and $F$ are unlabelled, we trivially have $[\![(E \triangleright F) \odot M_P]\!] = true$, but the reverse does not necessarily hold. Indeed, the valuation of a controlled process returns the best possible evaluation, so as long as one trace in $(E \triangleright F) \odot M_P$ has weight $true$, the whole process also evaluates to $true$. In other words, in the context of security, the valuation of a process can be seen as: if $[\![(E \triangleright F) \odot M_P]\!] = false$, then there is no possibility for the controlled process to satisfy $P$.

### 4.2 Controller Quantities

In the previous example, we monitor the security policy on the controlled target, rather than controlling the monitored target. This latter approach would indeed maintain the weight of some of the bad actions, and so the value of each trace would not match the satisfaction of the policy. Furthermore, in order to be consistent with the semantical rules, if the controlled target is labelled with some weight, the controller needs to be labelled with compatible weights. In other words, given a target monitored by $M_P$, the controller must be labelled in $\mathbb{K}_B$. For instance, $\lambda_{true}(E)$ associates each transition of $E$ with $true$, and since it is the neutral value of the multiplication, the weight of the target is that of the controlled target.

Clearly, these approaches are not exclusive, and it might be valuable to monitor both how many times the target tried to violate the policy and whether the controlled target violates the policy. Hence, using the simple cartesian composition $\odot_C$, where each operation is defined in a point-wise way, given an unlabelled target $F$ and controller $E$ and a security monitor $M_P$, a trace $t$ belonging to the process $(\lambda_{true}(E) \triangleright (F \odot M_P)) \odot_C M_P$ has a weight $(b_1, b_2)$, where $b_1$ indicates whether the target tried to violate the policy, and $b_2$ whether this trace actually satisfies the policy.

In some cases, it might be desirable to monitor both the controller and the target independently. For instance, the controlling actions can be associated with a notion of cost [11].

*Example 5.* The monitor $M_n$ defined in Example 2 can be extended to also monitor controlling actions, in such a way that accepting an action costs 0, suppressing an action costs 1 and inserting an action costs 2:

$$M_c = (x, 0).M_c + (\boxminus y, 1).M_c + (\boxplus w.v, 2).M_c$$

for $x, y, w, v \in Act$. The global energy consumed both by a controller $E$ and a target $F$ can be obtained with $(E \odot M_c) \triangleright (F \odot M_1)$. For instance, given the process $A = a.b.a$ and the controller $E_2$ of Example 4, we have:

$$(E_2 \odot M_c) \triangleright (a.b.a \odot M_1) \overset{\tau,1+3}{\rightarrow} (E_2 \odot M_c) \triangleright (b.a \odot M_1) \overset{b,0+2}{\rightarrow}$$
$$(E_1 \odot M_c) \triangleright (a \odot M_2) \overset{a,0+3}{\rightarrow} (E_1 \odot M_c) \triangleright (0 \odot M_2)$$

which raises a total energy consumed of 9. In order to measure only the energy consumed by the controlled target, taking also into account controlling actions, one should instead consider the process $((E \odot M_c) \triangleright (\lambda_0(F)))\|_{\{a,b\}} M_1$.

Note we can use directly the parallel composition with $M_1$, since the controlled target is already labelled with energy quantities. In this case, for $A = a.b.a$ and $E_2$, the total energy consumed is only 6, since the energy of the first action $a$ is not taken into account.

### 4.3 Evaluation Strategy

In order to evaluate a given controller against a given target, different labelling functions can be used, and as shown above, the actual order can have an impact on the global

valuation. For the sake of clarity, we introduce the notion of *matching operator* $\bowtie$, which has the following form:

$$E \bowtie F = \lambda_T(\lambda_E(E) \triangleright \lambda_F(F))$$

where $\lambda_E$ labels the controller, $\lambda_F$ labels the target and $\lambda_T$ labels the controlled target.

*Example 6.* The different evaluation strategies defined here can be summarised as:

$$E \bowtie_D F = \lambda_{true}(E) \triangleright (F \odot M_P) \quad E \bowtie_P F = (E \triangleright F) \odot M_P$$
$$E \bowtie_C F = (E \odot M_c) \triangleright (F \odot M_1) \quad E \bowtie_G F = M_P \odot_L (E \bowtie_C F)$$

where $\bowtie_D$ detects policy violations, even if they are corrected by the controller, $\bowtie_P$ monitors the satisfaction of the policy by the controlled target, $\bowtie_C$ monitors the energy of both the controller and the target, and $\bowtie_G$ defines a lexicographic measure of the cost and the satisfaction of the policy.

## 5 Ordering controller strategies

In this section, we present a way to compare different controller strategies. Firstly, since we can always get the valuation of two controllers for a given target, we can easily compare them accordingly. We then generalize this ordering to *any* target.

Hence, we provide a classification for any considered semiring. This means that we classify both sound and not sound controllers. This leaves to the user the choice of the measure that has to be used for classifying controller strategies.

**Definition 12.** *Given a target $F$ and a matching operator $\bowtie$, a controller $E_2$ is* better *than a controller $E_1$ with respect to $F$, and in this case, we write $E_1 \sqsubseteq_{\bowtie,F} E_2$, if and only if $[\![E_1 \bowtie F]\!] \sqsubseteq [\![E_2 \bowtie F]\!]$.*

This definition does not directly depend on the semiring used to quantify the controlled target, and it is therefore possible to use the same definition to say that a controller is better than another one with respect to a security monitor, a cost monitor or any other measure.

Since it might be that a controller is better than another one for a specific target, and that the converse holds for some other target, we introduce a stricter ordering, where the comparison is performed over all possible targets.

**Definition 13.** *Given two controllers $E_1$ and $E_2$ and a matching operator $\bowtie$, we say that $E_2$ is* always better *than $E_1$, and in this case we write $E_1 \sqsubseteq_{\bowtie} E_2$ if and only if $E_1 \sqsubseteq_{\bowtie,F} E_2$, for any target $F$. In addition, if $E_1 \sqsubseteq_{\bowtie} E_2$ and there exists at least one target $F$ such that $[\![E_1 \bowtie F]\!] \neq [\![E_2 \bowtie F]\!]$, we say that $E_2$ is* strictly *better than $E_1$, and write $E_1 \sqsubset_{\bowtie} E_2$.*

Since each individual trace can be represented as a target, it implies that the valuation of $E_1$ should be lower for every possible trace. Hence, this definition identifies the cases where a controller strategy is always better than another one.

*Example 7.* Let us extend the example described in Section 3, such that we have now three actions $\{a, b, c\}$, and a policy $P$ stating that any trace should start with at least one action $b$. Now, consider the four following controllers:

$$E_1 = a.E_1 + b.E_1 + c.E_1 \qquad E_2 = \boxminus a.E_2 + b.E_1 + c.E_2$$
$$E_3 = \boxminus a.E_3 + b.E_1 + \boxminus c.E_3 \qquad E_4 = \boxplus a.b.E_1 + b.E_1 + \boxplus c.b.E_1$$

Intuitively, $E_1$ accepts all actions, $E_2$ suppresses all initial actions $a$, but accepts action $c$, $E_3$ suppresses both actions $a$ and $c$, and $E_4$ inserts a $b$ before any initial $a$ or $c$. As soon as an action $b$ is performed, all processes are equivalent to $E_1$, and accept all actions.

Since $E_3$ and $E_4$ are sound, we have $[\![E_3 \bowtie_P F]\!] = [\![E_4 \bowtie_P F]\!] = true$, for any target $F$. In addition, given any target $F$ such that $[\![E_2 \bowtie_P F]\!] = false$, we also have $[\![E_1 \bowtie_P F]\!] = false$. Since there are also targets $F$ such that $[\![E_2 \bowtie_P F]\!] = true$ and $[\![E_1 \bowtie_P F]\!] = false$, we have

$$E_1 \sqsubset_{\bowtie_P} E_2 \sqsubset_{\bowtie_P} E_3 \equiv_{\bowtie_P} E_4$$

where $\equiv_{\bowtie}$ is the equivalence relation induced by the partial order $\sqsubseteq_{\bowtie}$. In other words, $E_3$ and $E_4$ are maximal, and $E_1$ is strictly worse than $E_2$.

However, it is worth observing that $E_1$ is not the worst possible controller. Indeed, $E_1$ leaves unchanged the correct traces of $F$, meaning that there exists some targets $F$ such that $[\![E_1 \bowtie_P F]\!] = true$. The worst controller always outputs incorrect traces, even when the target is correct. For instance, we can define the controller $E_0 = \boxplus b.a.E_1 + a.E_0 + c.E_0$, which satisfies $[\![E_0 \bowtie_P F]\!] = false$, for any target $F$, and therefore $E_0 \sqsubset_{\bowtie_P} E_1$.

In some cases, controllers can be incomparable. In the previous example, the controller that only suppresses bad actions $a$ is incomparable with the one that only suppresses bad actions $c$. Furthermore, the choice of the controlling operators can have an impact on the overall evaluation.

We believe this example to be representative of the contribution of our framework. Indeed, a traditional, qualitative analysis would label $E_3$ and $E_4$ as sound on the one hand, and $E_0$, $E_1$ and $E_2$ as equally incorrect on the other hand. The quantitative analysis provides us instead with an ordering over controllers. Of course, it is always desirable to use a sound controller, but when such an option is not available, it is useful to know which one is the next best. For instance, if one cannot implement the controlling strategy $E_3$ because the action $c$ is uncontrollable [1], i.e., cannot be suppressed or protected, then a security designer may prefer to choose $E_2$ over $E_1$, and certainly over $E_0$.

The controllers $E_3$ and $E_4$ are equivalent with respect to $\bowtie_P$, since they are both sound, and, if policy satisfaction is the only criterion, a security designer might choose either. However, other dimensions can easily be included within our framework, with the intuition that the more accurate is the quantification of the controlled system, the more informed is the security designer to choose a particular controller.

*Example 8.* In order to compare the previous controllers $E_3$ and $E_4$, let us consider the matching operator $\bowtie_G$ with the extended cost monitor $M_c$ of Example 5:

$$(a,0).M_c + (b,0).M_c + (c,0).M_c + (\boxminus a,1).M_c + (\boxminus c,1).M_c + (\boxplus a.b,2).M_c + (\boxplus c.b,2).M_c$$

First, it is worth observing that since we use the lexicographic ordering, the relations $E_0 \sqsubset_{\bowtie_G} E_1 \sqsubset_{\bowtie_G} E_2 \sqsubset_{\bowtie_G} E_3$ and $E_0 \sqsubset_{\bowtie_G} E_1 \sqsubset_{\bowtie_G} E_2 \sqsubset_{\bowtie_G} E_4$ still hold. However, $E_3$ and $E_4$ are no longer equivalent, and as a matter of fact, they become incomparable. Indeed, consider the target $F_1 = a$: we have $[\![E_3 \bowtie_G F_1]\!] = (true, 1)$ and $[\![E_4 \bowtie_G F_1]\!] = (true, 2)$, meaning that $E_4 \sqsubset_{\bowtie_G, F_1} E_3$. On the other hand, given the target $F_2 = a.a.a$, we have $[\![E_3 \bowtie_G F_1]\!] = (true, 3)$ and $[\![E_4 \bowtie_G F_1]\!] = (true, 2)$, meaning that $E_3 \sqsubset_{\bowtie_G, F_2} E_4$, and therefore that $E_3$ and $E_4$ are incomparable.

The previous example illustrates that, in general, there might not be a strictly best strategy. In some cases, it might be possible to define an *optimal* strategy, which is best *in average*. For instance, in the previous example, if each trace can be associated with a probability, then the *expected cost* of each controller can be computed, thus providing a unique best controller.

## 6 Related Work

The problem of finding an optimal control strategy is considered by Easwaran et al. in [14] in the context of software monitoring, where the system is represented as a Directed Acyclic Graph, and where rewards and penalties with correcting actions are taken into account, thus using dynamic programming to find the optimal solution. Similarly, an encoding of access control mechanisms using Markov Decision Process is proposed in [23], where the optimal policy can be derived by solving the corresponding optimisation problem. Markov chains are used in PRISM [16] in order to evaluate and validate systems in a quantitative and probabilistic way. From a different perspective, Bielova and Massacci propose in [3] a notion of distance among traces, thus expressing that if a trace is not secure, it should be edited to a secure trace close to the non-secure one, thus characterizing enforcement strategies by the distance from the original trace they create. In a recent approach [11], a notion of cost similar to the one in this paper is used to compare correct enforcement mechanisms (defined as state machines) with different strategies. In this work, we focus on using a generalised notion of weight expressed as an element of a semiring, following some intuitive leads given in [22] in order to move from qualitative to quantitative enforcement. Semirings have been used by Bistarelli et al. in the context of access control [6] and trust systems [5]. Here we use them in the context of enforcement mechanism defined using a process algebra, following the approach by Buchholz and Kemper [7]. We also consider in this paper the possibility for a controller not to be correct, i.e., to allow for some violations of the policy. Such a possibility is quantified over traces in [12] for non-safety policy, where a controller cannot be both correct and fully transparent. Caravagna et al. consider in [8] the notion of lazy controllers, which only control the security of a system at some points in time, and based on a probabilistic modelling of the system, quantify the expected risk. Basin et al. consider in [1] the case where some actions are uncontrollable (i.e., cannot be stopped), and define what policies can then be enforced, by modeling a controller as a

Deterministic Turing Machine. In the context of access control, Molloy et al. use a machine learning approach to predict the decision for a given request [26], and balance the risk of error against the cost of contacting the real mechanism to get the actual decision.

Non-binary measures of security have also been considered for access control systems, for instance by Cheng et al. [9], who consider that the level of security should correspond to a fuzzy domain rather than a strict separation between what is secure and what is not. Similarly, Zhang et al. define with the BARAC model [30] a notion of benefit for each access, with the underlying idea that allowing an access comes with a benefit for the system (while we take in this paper, a somewhat dual approach, by considering that denying an access might come with a cost for the system). The "value" of an access or an action can be for instance calculated using market-based techniques [25].

## 7 Discussion - Future work

Our framework allows a security designer to consider a security policy as yet another quantity to measure. Instead of a binary classification between sound and unsound controllers, we provide a finer-grained ordering, distinguishing between different degrees of soundness.

Other dimensions can be accommodated. A first point worth discussing is whether the valuation of processes should return the best-case scenario, as done in our current framework, or should instead return the worst-case scenario, thus following a rather traditional, pessimistic approach to security. If the $C$-semiring is equipped with a subtraction operation, i.e., an inverse to the addition operation, then such a valuation can be calculated as the subtraction of all the weights of the traces. For instance, in the boolean semiring, subtraction corresponds to conjunction, and it is easy to see that the valuation of a process would be true only if *all* traces evaluate to true. Similarly, subtraction in the cost semiring corresponds to $max$; in this case the valuation would return the highest possible cost. However, not all semirings have a subtraction operation. We will investigate this problem in future work.

As discussed at the end of Section 5, some controllers are incomparable. In some cases, it might be desirable to try to compare them nonetheless. A first step is to consider a single target when comparing two controllers, although, as said above, only the best-case scenarios are compared, which might not be fine-grained enough. One could however associate a weight to each target, and compose this weight with the valuation of each controller for each considered target. For instance, if we can define the probability of each target, and the valuation returns the energy spent by the controlled process, then it is possible to obtain the expected amount of energy spent by each controller. The expectation semiring [17] can be used in this process.

It is noteworthy that, given any two processes $A$ and $B$, we have $[\![A]\!] \sqsubseteq [\![A+B]\!]$, for any measured quantity. Indeed, the best case of $A+B$ is necessarily better than the best case of $A$. In practice, given two controllers $E_1$ and $E_2$, $[\![E_1 \triangleright F]\!] \sqsubseteq [\![(E_1+E_2) \triangleright F]\!]$, for any target $F$. In other words, adding non-deterministic choice to the controller itself always improves security. Clearly, this characteristic is mostly of theoretical importance, but it raises the interesting question whether, given some quantities, there exists a deterministic maximal controller or not. For instance, given any safety property, we can build an optimal deterministic controller if we only monitor security. However, if we add a

notion of cost as in Example 8, we have two incomparable deterministic controllers, which are strictly worse than their non-deterministic composition. It would therefore be interesting to study the class of quantities that are enforceable by a deterministic enforceable controller against those for which only a non-deterministic controller is optimal, in a way loosely similar to the difference between the classes $\mathsf{P}$ and $\mathsf{NP}$ in computational complexity. Finally, our notion of a security policy is just a set of finite traces that a process is allowed to use. This set could be specified by e.g. automata or logics. Predicates, and formulas specifying them, could also be quantitative by themselves, e.g. employing logics with valuations in a $C$-semiring (see e.g. [18,10]). In this paper, we do not yet investigate this aspect of the framework; this is left as future work. In particular, we plan to use quantitative evaluation of security policies, specified by logic formulas, in order to extend previous work on automated verification and synthesis of (qualitative) controllers [21].

## References


1. D. Basin, V. Jugé, F. Klaedtke, and E. Zălinescu. Enforceable security policies revisited. In *Proceedings of POST*, volume 7215 of *LNCS*, pages 309–328. Springer-Verlag, 2012.
2. L. Bauer, J. Ligatti, and D. Walker. Edit automata: Enforcement mechanisms for run-time security policies. *International Journal of Information Security*, 4(1–2), 2005.
3. N. Bielova and F. Massacci. Predictability of enforcement. In *Proceedings of the International Symposium on Engineering Secure Software and Systems 2011*, volume 6542, pages 73–86. Springer, 2011.
4. S. Bistarelli. *Semirings for Soft Constraint Solving and Programming*, volume 2962 of *LNCS*. Springer, 2004.
5. S. Bistarelli, S. N. Foley, B. O'Sullivan, and F. Santini. Semiring-based frameworks for trust propagation in small-world networks and coalition formation criteria. *Security and Communication Networks*, 3(6):595–610, 2010.
6. S. Bistarelli, F. Martinelli, and F. Santini. A semiring-based framework for the deduction/abduction reasoning in access control with weighted credentials. *Computers & Mathematics with Applications*, 64(4):447–462, 2012.
7. P. Buchholz and P. Kemper. Quantifying the dynamic behavior of process algebras. In *Proceedings of the Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification*, PAPM-PROBMIV '01, pages 184–199. Springer-Verlag, 2001.
8. G. Caravagna, G. Costa, and G. Pardini. Lazy security controllers. In *Security and Trust Management*, STM'12, 2012. to appear.
9. P.-C. Cheng, P. Rohatgi, C. Keser, P. A. Karger, G. M. Wagner, and A. S. Reninger. Fuzzy multi-level security: An experiment on quantified risk-adaptive access control. In *Proceedings of the 2007 IEEE S&P*, pages 222–230. IEEE Computer Society, 2007.
10. V. Ciancia and G. L. Ferrari. Co-algebraic models for quantitative spatial logics. *ENTCS*, 190(3):43–58, 2007.
11. P. Drábik, F. Martinelli, and C. Morisset. Cost-aware runtime enforcement of security policies. In *Security and Trust Management*, STM'12, 2012. to appear.
12. P. Drábik, F. Martinelli, and C. Morisset. A quantitative approach for inexact enforcement of security policies. In *Proceedings of the 15th international conference on Information Security*, ISC'12, pages 306–321. Springer-Verlag, 2012.
13. M. Droste and P. Gastin. Weighted automata and weighted logics. In *In Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005*, pages 513–525. Springer-Verlag, 2005.

14. A. Easwaran, S. Kannan, and I. Lee. Optimal control of software ensuring safety and functionality. Technical Report MS-CIS-05-20, University of Pennsylvania, 2005.
15. R. Gay, H. Mantel, and B. Sprick. Service automata. In *Proceedings of the 8th International Workshop on Formal Aspects of Security and Trust (FAST)*, LNCS 7140, pages 148–163. Springer, 2012.
16. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
17. Z. Li and J. Eisner. First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *EMNLP*, pages 40–51, 2009.
18. A. Lluch-Lafuente and U. Montanari. Quantitative mu-calculus and ctl defined over constraint semirings. *TCS*, 346(1):135–160, 2005.
19. F. Martinelli and I. Matteucci. Partial model checking, process algebra operators and satisfiability procedures for (automatically) enforcing security properties. Technical report, IIT-CNR, 2005.
20. F. Martinelli and I. Matteucci. Through modeling to synthesis of security automata. *ENTCS*, 179, 2007.
21. F. Martinelli and I. Matteucci. A framework for automatic generation of security controller. *Softw. Test. Verif. Reliab.*, 22(8):563–582, Dec. 2012.
22. F. Martinelli, I. Matteucci, and C. Morisset. From qualitative to quantitative enforcement of security policy. In *Proceedings of MMM-ACNS'12*, pages 22–35. Springer-Verlag, 2012.
23. F. Martinelli and C. Morisset. Quantitative access control with partially-observable markov decision processes. In *Proceedings of CODASPY '12*, pages 169–180. ACM, 2012.
24. R. Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.
25. I. Molloy, P.-C. Cheng, and P. Rohatgi. Trading in risk: using markets to improve access control. In *Proceedings of the 2008 workshop on New security paradigms*, NSPW '08, pages 107–125. ACM, 2008.
26. I. Molloy, L. Dickens, C. Morisset, P.-C. Cheng, J. Lobo, and A. Russo. Risk-based security decisions under uncertainty. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*, CODASPY '12, pages 157–168. ACM, 2012.
27. B. Roark, R. Sproat, and I. Shafran. Lexicographic semirings for exact automata encoding of sequence models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers - Volume 2*, HLT '11, pages 1–5, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.
28. P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and A. Roscoe. *Modelling and Analysis of Security Protocols*. Addison-Wesley Publishing Co., 2000.
29. F. B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, 2000.
30. L. Zhang, A. Brodsky, and S. Jajodia. Toward Information Sharing: Benefit And Risk Access Control (BARAC). *Proceedings of POLICY'06*, pages 45–53, 2006.