Introducing Probabilities in Controller Strategies

Jerry den Hartog[1](✉) and Ilaria Matteucci[2]

[1] Technische Universiteit Eindhoven, Eindhoven, The Netherlands
J.d.Hartog@tue.nl
[2] National Research Council, Istituto di Informatica e Telematica (IIT), Pisa, Italy
Ilaria.Matteucci@iit.cnr.it

Abstract. In this paper we propose a basic framework to merge security controllers with probabilistic concepts. This framework provides a first step towards quantitative security achieved by probabilistic controllers. It extends the framework for specification, analysis, and automatic generation of security controllers provided in [21,23] by considering probabilistic aspects of the behaviour of both the target process and the controller. Controllers may actively try to influence the choice of action of the target system or only passively react to actions the target system tried to perform. In a non-probabilistic setting both active and passive controllers can be expressed by the same model. In a probabilistic setting, however, these two types of controllers can differ. We respectively use the notions of generative and reactive processes to capture this distinction and discuss the different behaviours obtaining in the different settings.

Keywords: Security controller · Probability · Reactive · Generative

1 Overview

The importance of securing software and systems needs little argument in our ever more connected and information saturated world. Indeed, the security of systems is one of the main challenges of the last decades in computer science. With ubiquitous information gathering and sharing of huge amounts of valuable sensitive data on the Internet and all its connected devices the importance of security is only growing. Nowadays, each of us is using on a daily bases, devices, such as smart-phones, that exposes most of the functionalities of a networked personal computer. The wide range of users, including many with less (security) skills, adds risks to security. Furthermore, smart-phones are used for multiple purposes, both for work and leisure activities, exposing the users to a wide range of attacks that may *e.g.*, leak private data.

This work has been partially supported by the Italian TENACE PRIN Project (#20103P34XC), ARTEMIS J.U. SESAMO (#295354) and EU FP7 Marie Curie project MEALS (#295261).

J. Garcia-Alfaro et al. (Eds.): DPM/SETOP/QASA 2014, LNCS 8872, pp. 233–249, 2015.
DOI: 10.1007/978-3-319-17016-9_15

Our increasingly complex devices containing ever more (personal) data incorporate software and components with differing levels of trustworthiness, such as different apps running on a smart-phone. Mechanisms to monitor and enforce security policies are thus essential in guaranteeing secrecy, confidentiality, and integrity of our (private) information. Different approaches, *e.g.*, [4,6,7,22,26] therefore, aim to formally define and automatically generate enforcement mechanisms able to guarantee the satisfaction of security policies on systems and devices. In particular, from a formal perspective, starting from Schneider's seminal work [26], a lot of effort has been spent on the characterization the kind of policies can be monitored and enforced at runtime. This leads to the clear identification of basic concepts, such as security policy, target (or monitored system), execution traces, enforcement mechanisms. Initially, this strand of work only considered blocking of the system as a possible countermeasure but it has since then been refined by considering several approaches for the correction of execution traces. The kind of security policies that can be enforced with these models have been extensively studied leading to precise characterization of what can be enforced with a certain class of enforcement mechanisms and under which conditions.

Another research strand related to this work addresses the adoption of formalisms such as automata [7,18] or process algebras [22,23], to better define the interactions between target and enforcement mechanism as well as enforcement capabilities. A main advantage of these approaches is their well studied properties and associated procedure and results. For instance, it is possible to formalize the behaviour of a security automata through process algebra operators. By combining process algebra techniques with concepts from temporal logic, it is possibly to study the problem of automatically synthesizing enforcement mechanisms for (parts of) targets and policies expressed in logic. The framework presented in [23], which is based on process algebras, partial model checking, and satisfiability techniques for logic, finds mechanisms that enforce global security policies on a system by considering the target as only a subcomponent of this system, thus enabling the possibility to project global security policies onto local ones.

In those approaches security is often regarded as a binary concept. Behaviour is either good or bad. Good behaviour is either enforced or not. Thus they focus on what we call *qualitative* enforcement, *i.e.*, finding a "good" enforcement mechanism for a security policy, if possible. However, in many cases perfect enforcement is not possible or *e.g.*, more costly than the value you are trying to protect. For instance, if the target system is an airplane, halting it is just not an option; it may cause failures and deaths (a *safety* violation). In case like this, we have to consider more aspects than 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 may only be *likely* able to enforce a policy or can *cost* less than another one in enforcing a specific security policy, or the *benefits* of enforcing a specific policy may fail to counter-balance the disadvantages. We can also consider the more challenging problem, that we are going to name hereafter *quantitative enforcement*, to find the "best" enforcement mechanism for a security policy and a target, in accordance to some quantitative criteria such as the probability of attack, its cost or precision of enforcement. Quantitative enforcement could be also seen as an optimization problem to find the best among all good enforcement mechanisms. It would, thus, be useful to be able to rank controllers based on different quantitative aspects such as cost.

However, many controllers are incomparable if we do not know the likely behaviour of the target; often some (possibly unlikely) target behaviour can be found for which one controller will be more expensive than the other and vice versa. By taking into account the likely behaviour of a target, such controllers can be compared by looking at the expected cost of the enforcement. This means that we have to consider probabilistic behaviour of the target. Similarly a probabilistic controller strategy will be more powerful than a deterministic strategy and more readily analysed than a purely non-deterministic strategy. Thus also for the controller probabilistic behaviour should be considered.

The main goal of this paper is to take a first step towards a formal framework that enables automated generation of probabilistic security controllers by providing a way for specifying the probabilistic behaviour of such controllers. In particular, we consider that both controller and target processes behave in a probabilistic way in order to define probabilistic controlling strategies. In [14], the authors introduce different models for probabilistic processes, including *reactive* and *generative* processes. Starting from these notions, this paper analyses the different combination of these probabilistic models for both the target and the controller process. In particular, we consider four types of controls; *Acceptance*, *Suppression*, and *Insertion* operators. According to the behaviour of the considered target, a controller can apply one of these methods. The application of different rules depends on the probabilistic behaviour of the target and on the nature of both processes, *i.e.*, whether they are reactive or generative. In the end, we obtain a complete view of all the cases. This helps us in the next step of our research stream for the synthesis of the optimal (if any) probabilistic controlling strategy.

The remainder of this paper is organized as follows: The next section provides some basic notions about probabilistic process algebra operators and reactive and generative processes. Section 3 presents semantics definitions of probabilistic controller operators. In this section, we also discuss different resulting behaviours that can be obtained by considering reactive/generative controller processes that control reactive and/or generative target systems. Section 4 compares our approach with respect to related literature. Section 5 draws the conclusion of the paper and presents some further directions.

2 Basic Probabilistic Process Algebra

In order to analyse how probability affects the interaction between a controller and a target, hereafter we consider a minimal language that is still sufficient to address the effect of different types of probabilistic choice on the controller and target process behaviour. This minimal language contains only actions, probabilistic choice, recursion and the controller process which keeps interpretation of the processes simple. For practical use this is clearly too restrictive thus we plan to extend this with *e.g.*, non-deterministic and parallel operators as future work.

We start from a set of actions Act, ranged over by $a, b, c, \ldots$. We extend this set with silent action τ and so called control actions $CAct = \{\ominus a \mid a \in Act\} \cup \{\oplus a.b \mid a, b \in Act\}$. We use ζ to range over this extended action set $EAct = Act \cup \{\tau\} \cup CAct$. A probability ρ is added to obtain probabilistic action $\rho \cdot \zeta$; *i.e.*, $PAct = [0,1] \times EAct$.

For the processes given below, we define transition systems with steps between processes labelled with probabilistic actions. For equivalence of processes we consider probabilistic bisimulation [14] for reactive processes and weak probabilistic bisimulation [3] for generative processes. We assume it is clear how these can be derived from the semantic rules we provide. Below we introduce the syntax of our processes and then in turn explain the different types of processes along with their semantics.

$$P ::= 0 \mid \textstyle\sum_{i \in I} \rho_i \cdot \zeta_i.P_i \mid \sum_{i \in I} \zeta_i \cdot \rho_i.P_i \mid X \mid P \triangleright_K P \qquad (1)$$

A process P can be the empty process 0. This process has no transitions. It can also be a probabilistic choice between actions ζ_i followed by a corresponding process P_i (referred to as the continuation of the process). We omit the $\sum$ symbol if there is only one option and also use $+$ as an alternate notation. We consider both generative and reactive probabilistic choice as explained below.

In a *generative* choice, denoted by $\sum_{i \in I} \rho_i \cdot \zeta_i.P_i$, the sum of all the probabilities of all possible actions ($\sum_{i \in I} \rho_i$) is one. This corresponds to an "active" probabilistic choice; it probabilistically chooses which action to execute next.

In a *reactive* choice, denoted by $\sum_{i \in I} \zeta_i \cdot \rho_i.P_i$, the sum of probabilities per action is one (for the actions present). This type of choice can be seen as passive; it waits for one of the offered actions to be selected and then probabilistically chooses how to continue after that. To emphasise this we place the probability after the action in the notation of this choice. The choice may also be seen as being only partially specified; the relative likelihood of actions is omitted. Control actions $\ominus a$ and $\oplus a.b$ and counted together with the action a. For this we define $\sim$ as the smallest equivalence relation on $EAct$ that contains $\ominus a \sim a$ and $\oplus a.b \sim a$ for all actions a, b. The requirement for a reactive choice thus is that $\sum_{j \in J, \zeta_i \sim \zeta_j} \rho_j$ is one for each $i \in I$.

Both generative choice $\sum_{i \in I} \rho_i \cdot \zeta_i.P_i$ and reactive choice $\sum_{i \in I} \zeta_i \cdot \rho_i.P_i$ can take a step $\xrightarrow{\rho_i \cdot \zeta_i}$ to reach P_i for each $i \in I$. A technical complication in probabilistic transition systems is that when exactly the same transition is obtained from different branches in a probabilistic choice, as for *e.g.*, $\frac{1}{2} \cdot a.0 + \frac{1}{2} \cdot a.0$, their multiplicity becomes important. This can be solved *e.g.*, by counting (turning the transition relation into a multiset) or by turning the transitions into one transitions by adding their probabilities. To not clutter the presentation, we ignore this issue here and implicitly assume equal steps are combined by adding their probabilities. Thus we obtain $\frac{1}{2} \cdot a.0 + \frac{1}{2} \cdot a.0 \xrightarrow{1 \cdot a} 0$.

For the remainder of this paper, we assume that a process will not alternate reactive and generative choices. (This could be relaxed by defining a notion of similarity for such alternating processes. We could then even combine both types of options in one choice if we indicate which options are generative and which reactive, for example by splitting actions in reactive and generative ones. See also Example 1). Formally we thus require a process to be either a reactive or a generative process which are defined by: 0 is a reactive process, a reactive choice $\sum_{i\in I} \zeta_i \cdot \rho_i.P_i$ is a reactive process when P_i is a reactive processes for each $i \in I$, a process variable X is a reactive process if its body is and $P \triangleright_K P'$ is a reactive process when both P and P' are reactive processes. In turn 0 is a generative process, a generative choice $\sum_{i\in I} \rho_i \cdot \zeta_i.P_i$ is a generative process if P_i is a generative process for each $i \in I$, a process variable X is a generative process if its body is and $P \triangleright_K P'$ is a generative processes when P or P' is a generative process and the other is either a generative or a reactive process. (The reason for this typing of the control process will be clarified in the next section).

A process variable X is also a process and its steps are those of its body. We use guarded equations to specify the body associated to a process variable, *e.g.*, $X = 1 \cdot a.X$ defines X to be a process that produces an infinite sequence of a actions.

In [22,23] the definition of possible security *controller process algebra operator*, denoted by $\triangleright$, has been provided. We also use this operator in our processes, writing $C \triangleright_K F$ for controller process C controlling target process F according to some strategies denoted by K. The parameter K ranges over $\{A, S, I\}$, *i.e.*, *Acceptance* operator, *Suppression* operator, and *Insertion* operator. These operators describe three possible strategies that can be applied in order to control the behaviour of (possibly untrusted) target components by a control program. The intuition behind of the definition of these three operator is the following one: the acceptance operator behaves as a synchronous composition, the suppression operator models communication through synchronizations, and insertion operator works as an asynchronous composition made in an asymmetric way.

The semantics of the controller processes is treated in the next section. We extend the original definition of the operators as it has been provided in [22] to create a probabilistic version of the controller operators. The informal interpretation of Acceptance, Suppression, and Insertion operators is as follows:

Acceptance Operator (A) constrains the controller and the target to perform the same action, in order for it to be observed in the resulting behaviour. Given two processes C and F, the semantics of acceptance operator is equivalent to that of CSP-style parallel composition [16] of C and F, where synchronisation is forced over all actions of the two processes.

Suppression Operator (S) 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 τ action.

Insertion Operator (I) describes the capability of correcting some bad behaviour of the target, by inserting another action in its execution trace. It is worth noting that the target does not perform any action, but rather stays in its current state, as in [7].

Remark 1. The main difference between the controller operators treated here and the ones reported in [22] is that we do not consider *blocking* controllers which prevent the target from performing an action at all. A classical example of a blocking controller is the *truncation* operator which stops the target to perform actions upon attempting a disallowed action. The acceptance controller that we do treat, while disallowing some actions, always allows the process to select one of the allowed actions. In our setting a process is thus either equivalent to 0 or it will always perform some action. To model blocking controllers one could add a notion of failure δ though truncating can also be modelled as suppression of the action followed by no allowed behaviour (0).

Note that a controller only controls actions (see the rules in the next section) not *e.g.*, control actions in $CAct$ of the form $\ominus a$ or $\oplus a.b$. Thus a controller C will typically not control another controller C', *e.g.*, $(C \triangleright_K C') \triangleright_K F$ is usually not meaningful (if C' is a simple acceptance controller it would technically be possible), though C may control a process F already controlled by another controller C' as in $(C \triangleright_K (C' \triangleright_K F))$.

3 Controlling Probabilistic Processes

In this section, we provide the semantics for the controllers in a probabilistic setting. Recall that reactive choice models process behaviour which is guided by the outside such as input to the process; the user selects one of the offered actions. Generative choice models active process behaviour, where the selection of an option is driven by the process itself; for example the output generated by the process. When considering a target process F controlled by a controller C we can consider both types of choice for either the target and the controller. Below we treat each of the four combinations for all the three controller operators giving some intuition on how such processes can be interpreted and thus what type of systems they may be able to model. First we focus on the interpretation of reactive and generative choice in the controller and the target process by considering a probabilistic version of a basic type of controller; the acceptance controller. Next we extend the discussion by introducing probabilistic versions of the other types of controllers: the suppression and insertion controllers.

3.1 Probabilistic Acceptance Controller

The acceptance controller specifies which actions a target system is allowed to take. Any other action along with its continuation is blocked.
The semantics of the probabilistic version of the acceptance operator is given by:

$$\frac{C \xrightarrow{\rho\cdot a} C' \quad F \xrightarrow{\sigma\cdot a} F'}{C \triangleright_A F \xrightarrow{\rho\sigma/n_A\cdot a} C' \triangleright_A F'} \qquad \frac{C \xrightarrow{\rho\cdot\tau} C'}{C \triangleright_A F \xrightarrow{\rho/n_A\cdot\tau} C' \triangleright_A F} \qquad \frac{F \xrightarrow{\sigma\cdot\tau} F'}{C \triangleright_A F \xrightarrow{\sigma/n_A\cdot\tau} C \triangleright_A F'}$$

where normalization factor n_A is 1 if $C \triangleright_A F$ is a reactive process (*i.e.*, both C and F are reactive) and otherwise $n_A = \sum_{C \xrightarrow{\rho\cdot a} C', F \xrightarrow{\sigma\cdot a} F'} \rho\sigma + \sum_{C \xrightarrow{\rho\cdot\tau} C'} \rho + \sum_{F \xrightarrow{\sigma\cdot\tau} F'} \sigma$. This semantics rule states that if F performs action a and this action is allowed by controller C then $C \triangleright_A F$ performs action a. The probability of performing action a is determined by the probabilities for a in C and F and the type of choice (reactive or generative). Both the controller and the target system can independently perform a silent action τ.

For a generative controlled process $C \rhd_A F$, we use normalization to ensure the resulting process is again generative; the total probability sums up to one. The interpretation of this is that if the controller prevents an action selected through a probabilistic choice from occurring, the system will attempt the choice again (with the same probabilities). This eventually results in one of the allowed options being chosen according to the normalized probabilities or in no behaviour at all (equivalent to the 0 process), in case none of the options are allowed. Below we illustrate the interpretation of a controlled process for the four possible combinations of reactive or generative target system and controller.

Reactive target system, reactive controller. Consider the following system: a building with two doors, one leading to the elevator (door a) and one leading to the stairs (door b). The simple system that keeps the doors open can be modelled by reactive process $F_1 = a \cdot 1.F_1 + b \cdot 1.F_1$

Suppose we want to protect the elevator from overload. A strategy to do this could be to place a guard who closes the elevator door with probability $\frac{1}{2}$ whenever someone enters the elevator and opens it again once someone uses the stairs. This controller can be modelled by (reactive) process C_1 that satisfies:

$$C_1 = a \cdot \tfrac{1}{2}.C_1 + a \cdot \tfrac{1}{2}.C_1' + b \cdot 1.C_1 \tag{2}$$
$$C_1' = b \cdot 1.C_1 \tag{3}$$

Using controller C_1 to control process F_1, resulting in process $C_1 \rhd_A F_1$, will here actually result in exactly process C_1 because at each stage all options allowed by C_1 are offered by F_1. Of course this does not have to be the case. Take for example:

$$F_2 = a \cdot \tfrac{1}{2}.F_2 + a \cdot \tfrac{1}{2}.0 + b \cdot \tfrac{1}{3}.F_2 + b \cdot \tfrac{2}{3}.0 \tag{4}$$
$$C_2 = b \cdot \tfrac{1}{2}.C_2 + b \cdot \tfrac{1}{2}.0 \tag{5}$$

The process $C_2 \rhd_A F_2$ will only perform b steps and after every b step there is a five in six chance the process stops; only if both F and C decide to continue will the process be able to take further steps.

As can be seen from examples given above, a reactive target system controlled by a reactive controller acts as one may expect; the enabled actions are those allowed by both the system and the controller and, after selection of one of those controlled actions, the system and controller independently (probabilistically) choose their strategy for which future actions are offered/allowed. Thus a reactive controller may disallow certain steps and probabilistically determines its strategy for future control based on the action that it sees. It, however, does not interfere with the strategy of the target system in choosing its next state for the actions that are allowed. The probabilities for allowed actions are unaffected, the others become zero. The combined process is reactive: the user selects from the enabled actions, *i.e.*, those offered by the target system and allowed by the controller. The controller observes the actions that users execute on the system and decides its control strategy accordingly by disallowing or enabling certain future actions.

Generative target system, reactive controller. If the system we are trying to control is actively making (probabilistic) choices we need to use a generative model to be able to describe this. Consider, for example, users of the building from the example above. These users will actively decide which door to take, for example by randomly selecting one of them. The building and the users together may be modelled by generative process $F_3 = \frac{1}{2} \cdot a.F_3 + \frac{1}{2} \cdot b.F_3$.

Our controller has remained reactive; it only responds to the actions the users of the building execute. Still, because the users drive the choice of actions, the controlled process will still actively choose the actions; it is generative. Consider process $C_1 \triangleright_A F_3$. The users will randomly select a door. If it is the stairs nothing changes. If it is the elevator (a), the controller may react by disabling that door. Then, the next user may try the elevator door but will find it blocked so in the end will have to take the other door (b) with probability 1. The controller will react to this action by re-enabling door a.

The intuitive interpretation of a generative-reactive combination is also clear; the generative target process drives the selection of actions amongst those that are allowed. The reactive controller simply observes which actions the target system executes and updates its strategy.

Generative target system, generative controller. So far we have considered a "passive", reactive controller that can only react to choices being made by updating its strategy for future actions. However, what if we consider a controller that actively tries to influence also the current action. Consider, for example, the guard at the door; the guard knows that taking the stairs is both healthier and cheaper, so would prefer more people taking the stairs. When people arrive at the building the guard tries to convince them that taking the stairs is much better. These actions and preference of the guard (combined with how successful she is in convincing users) can be captured by a generative process, for example; $C_3 = \frac{1}{3} \cdot a.C_3 + \frac{2}{3} \cdot b.C_3$.

Now, if again the users randomly select a door (F_3), the guard trying to convince users to take the stairs will influences the overall probability, resulting in a two third probability of taking the stairs. If, on the other hand, users in a wheelchair arrive who cannot use the stairs; ($F_4 = 1 \cdot a.F_4$) then the preference of the guard will be ignored; the result will be action a with probability 1. These cover the two extreme cases for the users; no preference at all or only having one of the options. In general the preferences of the guard and the user are combined. For instance, if the users have a preference for the stairs already, it will be further strengthened by the preference of the guard.

A generative target and a generative controller combined yield a generative overall system. The target system still drives the choice of the next action among those allowed by the controller but now the controller is also actively involved in this choice. The controller can indicate preferences which will influence the choice of the target system.

Reactive target system, generative controller. If the target system is reactive, it has not specified a preference for any of the actions. A generative controller can by itself determine the likelihood of next action. As an example, we can consider the model for the building (without its users) and an active guard that opens only one door for each user that arrives. As seen above, a generative process can also indicate 'no preference' by giving equal likelihood to each action. When controlled by a generative controller, a reactive target system, indeed, behaves exactly the same as such a generative system without preferences. This combination can thus be seen as a special case of the generative-generative combination.

This intuition for the different controller-target system combinations also translates to the other types of controller automata. However, the other types of controller automata include actions that are inherently generative which posses some restrictions on the processes that can be used. In the next section we discuss how these controllers can be extended to a probabilistic setting as well as the required restrictions.

3.2 Probabilistic Suppression, Insertion, and Edit Controllers

The Suppression, Insertion, and Edit controllers enable additional control strategies. Suppression controllers are able to hide target actions to an external observer; Insertion controllers aim at correcting wrong execution traces by inserting actions before not allowed action; Edit controllers combine both the functionalities in such a way to be able to hide actions and correct execution traces.

In order to extend the definitions above with probabilistic versions of these controllers, we have to consider again the reactive versus the generative choice. Recall that a reactive process passively reacts to external actions, only probabilistically choosing the continuation after an action. It can also be interpreted as an incompletely specified choice. A generative process, on the other hand, actively makes a probabilistic choice between actions to execute.

According to the example of probabilistic acceptance controller, both reactive and generative choices are useful for modelling different types of target systems and controllers. However, some types of actions are inherently generative. The silent τ action, for example, is used to model a change in the system state which is not (directly) observable by the user. Such an action can obviously not be "chosen by the user". The system has to actively choose to execute it.

We may need to deal with inherently generative actions in a reactive setting. The Suppression controller, for example, may cause an action to be replaced by silent action τ. In [1], the syntax of the hiding operation, which also replaces actions by τ, is extended with an additional (probability) label to allow its use in a reactive setting. The label p determines the generative probability of τ if a reactive action is hidden and is ignored if a generative action is hidden. This thus uses the 'not fully specified choice' interpretation of reactive choice and provides the added specification in the syntax in case it is needed. Here we follow a similar approach; we require that probabilities for inherently generative actions are fully specified. A choice involving such actions is interpreted as always first making an active choice between either executing one of the generative actions or, if present, offering the choice between the reactive options.

Example 1. Suppose we add an additional label to the syntax; *e.g.*, let $^{\frac{1}{2}}a$ denote a reactive action a that will be selected with probability $\frac{1}{2}$ when suppressed. If a reactive process with additional label $^{\frac{1}{2}}a \cdot 1.p_1 + b \cdot 1.p_2 + c \cdot 1.p_3$ is controlled by a reactive controller that suppresses action a and allows actions b and c the result is a choice of the form: $\frac{1}{2} \cdot \tau.p_1 + \frac{1}{2} \cdot (b \cdot 1.p_2 + c \cdot 1.p_3)$. Note that this is a choice between processes rather than between actions or continuations after actions; as such it is not a (generative or reactive) process as defined in this paper but rather a stratified choice [14].

We actually do not need to add additional labels to our syntax to model this; a generative choice has exactly the same expressiveness as a reactive choice with additional labels. The labels are determined by the relative probabilities of the first action.

Example 2. The labelled reactive process of the previous example is expressed by generative process $a \cdot \frac{1}{2}\tau.p_1 + b \cdot \frac{1}{4}.p_2 + c \cdot \frac{1}{4}.p_3$. When this process is controlled by a reactive controller that suppresses action a and allows actions b and c, the result is generative process: $\frac{1}{2} \cdot \tau.p_1 + \frac{1}{4} \cdot b.p_2 + \frac{1}{4} \cdot c.p_3$.

We can informally write this as a stratified choice $\frac{1}{2} \cdot \tau.p_1 + \frac{1}{2} \cdot (\frac{1}{2} \cdot b.p_2 + \frac{1}{2} \cdot c.p_3)$ which, by reversing the interpretation from generative probability back to 'additional labels' on the actions b and c, results in the choice $\frac{1}{2} \cdot \tau.p_1 + \frac{1}{2} \cdot (^{\frac{1}{2}}b \cdot 1.p_2 + {}^{\frac{1}{2}}c \cdot 1.p_3)$. Note that this choice is exactly the same as the result in the previous example. The 'additional labels' for b and c are included but these are ignored when not needed.

The example above illustrates that generative processes can be used rather than labelled reactive ones without changing expressiveness or the semantics. Also, as already seen in the previous section, a uniform generative choice, *i.e.*, one which assigns equal probability to all initial actions, behaves like a reactive choice. Thus, having to give a probability to all actions, rather than to only those that may be suppressed, is not a real restriction; we can ignore/omit the probabilities if they are equal and the actions are not suppressed. Finally, by using the interpretation above we can restrict ourselves to generative processes rather than needing to consider stratified choices.

Probabilistic Suppression Controller. The probabilistic suppression controller operator is an extension of the acceptance operator. We require that at least one of the controller or the target is generative. The probabilistic suppression operator inherits the three rules given for the acceptance operator and, additionally, has the following rule:

$$\frac{C \xrightarrow{\rho \cdot \ominus a} C' \quad F \xrightarrow{\sigma \cdot a} F'}{C \triangleright_S F \xrightarrow{\rho\sigma/n_S \cdot \tau} C' \triangleright_S F'}$$

with $n_S = n_A + \sum_{C \xrightarrow{\rho \cdot \ominus a} C', F \xrightarrow{\sigma \cdot a} F'} \rho\sigma$. The inherited semantics rules allow F, as with the acceptance operator, to perform the action $a \neq \tau$ if also performed by the controller, *i.e.*, if F performs action a and C performs the same action a, then $C \triangleright_S F$ performs action a. The inherited rules allow both the controller C and the target F to independently perform τ actions. The new rule states that the controller is able to hide a possible incorrect action to an external observer by relabelling a to τ. The normalization factor n_S is updated to include this final possibility to ensure probabilities still add up to 1.

As the action is hidden from outside, a choice involving it clearly has to be made by the system itself. Hence, it needs to be generative. As such the rule does not work in the case both controller and target are reactive. In particular, the rule then does not properly work in the following two cases:

– when the target performs internal action τ;
– when the controller suppresses more that one action.

In these cases a choice has to be made between actions while their relative probabilities are not known. The following example illustrates this point. (For an example of use of suppression in a generative controller see controler C_S in Sect. 3.3.)

Example 3. Consider again the building example used above. We consider a basic system that allows one entry but through either door. A guard in front of the doors hides the choice of the user:

$$F = a \cdot 1.0 + b \cdot 1.0 \tag{6}$$

$$C = \ominus a \cdot 1.0 + \ominus b \cdot 1.0 \tag{7}$$

For this simple system the above rule would give for $C \triangleright_S F$:

$$C \triangleright_S F = \tau \cdot 1.0 + \tau \cdot 1.0 = \tau \cdot .2.0 \tag{8}$$

The total probability is more than 1. We cannot scale this probability without knowing the relative probability to use between the two options.

Probabilistic Insertion Controller. Sometimes actions are not forbidden per se but they need to be prepared for, *e.g.*, you are allowed to enter the building but only if a security guard is present in the building. In this case a controller may want to insert an action (send in guard) before allowing your action (enter building). The insertion controller operation is used to express this control strategy. The special action $\oplus a.b$ is used to express that the controller inserts action b before allowing action a. Also insertion operator inherits the rules of acceptance operators. In addition to those rules, it has the following key semantic rule:

$$\frac{E \overset{\rho\cdot\oplus a.b}{\longrightarrow} E' \quad F \overset{\sigma\cdot a}{\longrightarrow} F'}{E \triangleright_I F \overset{\rho\sigma/n_I\cdot b}{\longrightarrow} E' \triangleright_I F}$$

(Note that if F can chose among multiple a transitions the resulting steps of the controlled process are the same. As mentioned we implicitly assume these steps are combined by adding their probabilities together.) The normalization factor is adapted as before to also consider $\oplus a.b$ action; $n_I = n_A + \sum_{C \overset{\rho\cdot\oplus a.b}{\longrightarrow} C', F \overset{\sigma\cdot a}{\longrightarrow} F'} \rho\sigma$. In this rule we assume that the users/controlled systems can change their mind;

after inserting the action b, F again gets to make its choice for which action to execute. It is thus very well possible that action a in the end does not actually happen (*e.g.*, you do not enter the building after the guard has been sent in.)

Insertion of an action has, like the τ action, an inherent generative nature; it is the system that decides to insert the action. This rule thus is not valid when both the controller and target are both reactive processes and the target is able to perform the same action that the controller has inserted.

Example 4. Coming back to the building example with a target that leaves both doors open. Consider a reactive controller that always lets the user open the door to the stairs (b) but if the user tries to take the elevator (a), it blocks her and first opens the door for the stairs in order to convince her to take the stairs instead of the elevator.

$$F = b \cdot 1.0 + a \cdot 1.0 \tag{9}$$

$$C = b \cdot 1.0 + \oplus a.b \cdot 1.0 \tag{10}$$

Hence, $C \triangleright_I F$ always leaves the stairs door open and hermetically closes the stairs one. The resulting probabilistic process is as follows:

$$C \triangleright_I F = b \cdot 1.0 + b \cdot 1.0 = b \cdot 2.0 \tag{11}$$

The total probability is more than 1 and we miss the information needed to normalize.

Probabilistic Edit Controller operator. It is possible, as is done in the quantitative case, to combine the functionalities of suppression and insertion in a single probabilistic controller, a *probabilistic Edit controller.* The Edit controller thus inherits the two semantic rules above along with their restrictions on processes that can be used.

The non-probabilistic Edit controller operators apply suppression and insertion rules in a mutual exclusive way, *i.e.*, the set of action that can be suppressed is disjoint from the one of actions that are prevented to be made by inserting something before. This assumption has been made in order to be able to deterministically apply one rule or the other. For the probabilistic controller we do not need this assumption; we can use the relative probabilities of controlling actions $\ominus a$ and $\oplus a$. For instance consider in the building with users scenario that a check should be made upstairs before using the elevator. A guard lets people take the stairs (b) but if they try to take the elevator (a) he either simply forbids this or sends someone upstairs to make the check: $C = b \cdot 1.C + \ominus a \cdot \frac{3}{4}.C + \oplus a.b \cdot \frac{1}{4}.C'$ If a user tries to take the elevator then with probability $\frac{3}{4}$ they will be forbidden and with probability $\frac{1}{4}$ someone will be sent up the stairs first (with C' presumably upon confirmation that it is safe allowing use of the elevator.)

3.3 Comparing Probabilistic Operators

Let us consider again the example of the building with two doors, one for the elevator (door a) and one for the stairs (door b). We have already seen the example of the guard that tries to persuade users to take the stairs expressed with the probabilistic Acceptance operator: $C_A = \frac{1}{3} \cdot a.C_A + \frac{2}{3} \cdot b.C_A$ When people arrive at the building the guard tries to convince them that taking the stairs is much better. Note that simple locking the elevator door $C'_A = 1 \cdot b.C'_A$ is not an option as the whole system would block when a user unable to use the stairs arrives.

Guard C_A only tries to persuade the user. We can also consider a guard that wants to force users to not use the elevator. One possible strategy, expressed using the Suppression operator is $C_S = \ominus a \cdot 1.C_S + b \cdot 1.C_S$; the guard allows anyone to take the stairs but anyone trying to take the elevator is sent away. This can be combined with persuasion as in $C'_S = \frac{1}{3} \cdot \ominus a.C'_S + \frac{2}{3} \cdot b.C'_S$; the guard tries to convince users to take the stairs. Those that choose the elevator anyway are sent away.

We can extend any base strategy $(C_I^{(0)})$ with a guard that will always allow users to take the stairs but will first send them back a number of times when they try to use the elevator: $C_I^{(i)} = \oplus a.\tau \cdot 1.C_I^{(i-1)} + b \cdot 1.C_I$ Only users determined to take the elevator (*e.g.*, because they cannot take the stairs) will reach the base strategy.

Adding probability to the system description as well as the controller operators improves the ability to compare different controller strategies and to select the one with the best (expected) results. To illustrate this we add another feature to our example as a first glance to our future work. Users can only take the elevator when an operator is present. Action c expresses deploying the operator for that day. A controller could choose to insert action c at some point. We want to allow as many users as possible to enter but deploying the operator incurs some cost. The result (traces) of the system can thus be quantified by considering these costs and benefits. While formalizing cost and benefit is part of future work we consider a basic example in order to show this idea.

Both C'_S above and C_c below which inserts an operator as soon as needed achieve the security property that the elevator is not used without an operator.

$$C_c = \oplus a.c \cdot 1.C'_c + b \cdot 1.C_c \tag{12}$$

$$C'_c = a \cdot 1.C'_c + b \cdot 1.C'_c \tag{13}$$

If n users arrive without preference for a door, $F^{(n)} = \frac{1}{2} \cdot a.F^{(n-1)} + \frac{1}{2} \cdot b.F^{(n-1)}$, $F^{(0)} = 0$ the guard C'_S will convince two thirds of the users to take the stairs. One third is sent away. Guard C_c on the other hand will allow all users to enter but with high probability $(1 - \frac{1}{2}^n)$ will incur the cost of c. Thus if $(1 - \frac{1}{2}^n) \cdot \mathit{cost}(c)$ is less than $\frac{1}{3} \cdot \mathit{benefit}(a)$ the later strategy is superior in this scenario. If also users in a wheelchair arrive who cannot use the stairs; $(F = 1 \cdot a.F')$ then C'_S will not be able to convince them to take the stairs; such a scenario thus more quickly favors strategy C_c.

Besides allowing comparison of operators that guarantee a deterministic property, we can also consider probabilistic requirements, *e.g.*, a limit on the probability that the elevator is used. To sum up; adding probability to controller operator introduces a new parameter according to which it is possible to refine and to optimize controlling strategies in order to satisfy the system requirement as well as possible, and, sometimes in the *best* way.

4 Related Work

Security and, more specifically, its evaluation, are important topics that have been receiving a lot of attention in recent years. However, controlling systems and formally measuring and evaluating (quantitative) security aspects of the system remains a challenging subject which seems to have received limited study compared to its importance.

Caravagna *et al.* consider in [10] 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 [5] the case where some actions are uncontrollable (*i.e.*, cannot be stopped), and define what policies can then be enforced, by modelling 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 [25], and balance the risk of error against the cost of contacting the real mechanism to get the actual decision.

In a recent approach [20], the authors deal with probabilistic cost enforcement based on input/output automata to model complex and interactive systems. Associating to each execution trace a probability and a cost measure, it is possible to evaluate the expected cost of the monitor and of the monitored systems. Even though Input/output automata are similar to our reactive processes, our work goes beyond [20] by also considering generative processes as both controller and target. Furthermore, it uses a process algebraic specification of the controllers which we hope will allow for the automatic generation of controller automata as in [23]. Also, in [12], a notion of cost to compare correct enforcement mechanisms (defined as state machines) with different strategies.

Easwaran *et al.* in [13] aim at finding an optimal control strategy in the context of software monitoring. A system is represented as a Directed Acyclic Graph and rewards and penalties with correcting actions are taken into account. Dynamic programming is then used to find the optimal solution. Similarly, an encoding of access control mechanisms using Markov Decision Process is proposed in [24], where the optimal policy can be derived by solving the corresponding optimisation problem. Markov chains are used in PRISM [17] in order to evaluate and validate systems in a quantitative and probabilistic way. From a different perspective, Bielova and Massacci propose in [8] 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 [9], Buchholz and Kemper propose a *generalized process algebra* in which each operators has been extended with the notion of cost modelled through a semiring. In [11], the authors present a framework for quantitative evaluation of controller strategies. This framework is based on process algebra to model the behaviour of the process and on semirings to model different measures. Probability is not considered as an evaluation metric since it cannot be easily modelled through a semiring.

5 Conclusion and Future Work

In this paper we take a first step towards automated generation of optimal quantitative controller strategies. This step consists of classifying probabilistic controller strategies for probabilistic systems. The classification of the strategy considers reactive and generative processes for both the controller and target processes. In particular, focusing on a simplified process algebra with only probabilistic choice and a controller operator as its main components, we consider three possible corrective actions (acceptance, suppression, and insertion) that can be performed by the controller process. We provide a definition of probabilistic controlling strategies based on these actions and we analyse the different behaviours according to the reactive and generative nature of both controller and target processes.

The use of a probabilistic language allows modelling a wide range of strategies applied to different situations. Also many strategies which are incomparable in a possibilistic analysis can be compared probabilistically. When we want to compare controllers across multiple systems or we do not know the (exact) probabilities for certain choices (as will often be the case in realistic complex systems) having a notion of non-determinism in conjunction with probability would be useful.

To have a language powerful enough for specifying a realistic, complex system we aim to extend the basic process language considered here with more advanced controllers, non-deterministic choice and parallelism. We will then also need to extend our classification by considering different models for the combination of nondeterminism and probability, *e.g.* [2,15,19,27].

Another extension of this work that we plan to investigate is the combination with the notion of cost. In particular, we aim to be able to choice the best control strategy finding a trade-off between maintaining a security property and the cost of attack.

References

1. Aldini, A., Gorrieri, R.: Security analysis of a probabilistic non-repudiation protocol. In: Hermanns, H., Segala, R. (eds.) PROBMIV 2002, PAPM-PROBMIV 2002, and PAPM 2002. LNCS, vol. 2399, pp. 17–36. Springer, Heidelberg (2002)
2. Andova, S.: Process algebra with probabilistic choice. In: Katoen, J.-P. (ed.) AMAST-ARTS 1999, ARTS 1999, and AMAST-WS 1999. LNCS, vol. 1601, pp. 111–129. Springer, Heidelberg (1999)
3. Baier, C., Hermanns, H.: Weak bisimulation for fully probabilistic processes. In: Grumberg, O. (ed.) Computer Aided Verification. LNCS, pp. 119–130. Springer, Heidelberg (1997)

4. Bartoletti, M., Degano, P., Ferrari, G.L.: Policy framings for access control. In: Proceedings of the 2005 Workshop on Issues in the Theory of Security, pp. 5–11. ACM (2005)
5. Basin, D., Jugé, V., Klaedtke, F., Zălinescu, E.: Enforceable security policies revisited. In: Degano, P., Guttman, J.D. (eds.) Principles of Security and Trust. LNCS, vol. 7215, pp. 309–328. Springer, Heidelberg (2012)
6. Bauer, L., Ligatti, J., Walker, D.: More enforceable security policies. In: Cervesato, I. (ed.) Foundations of Computer Security: proceedings of the FLoC 2002 workshop on Foundations of Computer Security, pp. 95–104. DIKU Technical Report (2002)
7. Bauer, L., Ligatti, J., Walker, D.: Edit automata: enforcement mechanisms for run-time security policies. Int. J. Inf. Secur. **4**(1–2), 2–16 (2005)
8. Bielova, N., Massacci, F.: Predictability of enforcement. In: Erlingsson, Ú., Wieringa, R., Zannone, N. (eds.) ESSoS 2011. LNCS, vol. 6542, pp. 73–86. Springer, Heidelberg (2011)
9. Buchholz, P., Kemper, P.: Quantifying the dynamic behavior of process algebras. In: de Alfaro, L., Gilmore, S. (eds.) Proceedings of the Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification. LNCS, vol. 2165, pp. 184–199. Springer, Heidelberg (2001)
10. Caravagna, G., Costa, G., Pardini, G.: Lazy security controllers. In: Jøsang, A., Samarati, P., Petrocchi, M. (eds.) STM 2012. LNCS, vol. 7783, pp. 33–48. Springer, Heidelberg (2013)
11. Ciancia, V., Martinelli, F., Ilaria, M., Morisset, C.: Quantitative evaluation of enforcement strategies: position paper. In: Danger, J.-L., Debbabi, M., Marion, J.-Y., Garcia-Alfaro, J., Heywood, N.Z. (eds.) FPS 2013. LNCS, vol. 8352, pp. 178–186. Springer, Heidelberg (2014)
12. Drábik, P., Martinelli, F., Morisset, C.: Cost-aware runtime enforcement of security policies. In: Jøsang, A., Samarati, P., Petrocchi, M. (eds.) STM 2012. LNCS, vol. 7783, pp. 1–16. Springer, Heidelberg (2013)
13. Easwaran, A., Kannan, S., Lee, I.: Optimal control of software ensuring safety and functionality. Technical report MS-CIS-05-20, University of Pennsylvania (2005)
14. Glabbeek, R.V., Smolka, S., Steffen, B.: Reactive, generative and stratified models of probabilistic processes. Inform. Comput. **121**, 130–141 (1990)
15. den Hartog, J.I., de Vink, E.P.: Mixing up nondeterminism and probability: a preliminary report. Electr. Notes Theor. Comput. Sci. **22**, 88–110 (1999)
16. Hoare, C.: Communicating Sequential Processes, vol. 178. Prentice-hall, Englewood Cliffs (1985)
17. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011)
18. Ligatti, J., Bauer, L., Walker, D.W.: Enforcing non-safety security policies with program monitors. In: di Vimercati, S.C., Syverson, P.F., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 355–373. Springer, Heidelberg (2005)
19. Lowe, G.: Representing nondeterminism and probabilistic behaviour in reactive processes. Technical report PRG-TR-11-93, Oxforf University Computing Laboratory (1993)
20. Mallios, Y., Bauer, L., Kaynar, D., Martinelli, F., Morisset, C.: Probabilistic cost enforcement of security policies. In: Accorsi, R., Ranise, S. (eds.) STM 2013. LNCS, vol. 8203, pp. 144–159. Springer, Heidelberg (2013)
21. Martinelli, F.: Analysis of security protocols as open systems. Theor. Comput. Sci. **290**(1), 1057–1106 (2003)

22. Martinelli, F., Matteucci, I.: Through modeling to synthesis of security automata. Electr. Notes Theor. Comput. Sci. **179**, 31–46 (2007)
23. Martinelli, F., Matteucci, I.: A framework for automatic generation of security controller. Softw. Test. Verif. Reliab. **22**(8), 563–582 (2012)
24. Martinelli, F., Morisset, C.: Quantitative access control with partially-observable markov decision processes. In: Proceedings of CODASPY 2012, pp. 169–180. ACM (2012)
25. Molloy, I., Dickens, L., Morisset, C., Cheng, P.C., Lobo, J., Russo, A.: Risk-based security decisions under uncertainty. In: Proceedings of CODASPY 2012, pp. 157–168. ACM (2012)
26. Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. **3**(1), 30–50 (2000)
27. Segala, R.: Modeling and verification of randomized distributed real-time systems. Ph.D. thesis, Massachusetts Institute of Technology (1995)