On Quantitative Security Policies

Pierpaolo Degano, Gian-Luigi Ferrari, and Gianluca Mezzetti

Dipartimento di Informatica
Università di Pisa, Italy
{degano,giangi,mezzetti}@di.unipi.it

Abstract. We introduce a formal framework to specify and enforce *quantitative* security policies. The framework consists of: (*i*) a stochastic process calculus to express the measurable space of computations in terms of Continuous Time Markov Chains; (*ii*) a stochastic modal logic (a variant of CSL) to represent the bound constraints on execution speed; (*iii*) two enforcement mechanisms of our quantitative security policies: *potential* or *actual*. The potential enforcement computes the probability of policy violations, thus providing a sort of static evaluation of the policy. This supports the user to accept/discard a component when the probability of the security violation is below/above a suitable chosen threshold. The actual enforcement computes the deviation of the execution speed from the acceptable rate. This supports the run-time systems by driving the execution monitor to abort unsafe executions.

Introduction

In the last few years a new trend is emerging that exploits the network for computing in a different manner. Applications are no longer built as monolithic entities, rather they are constructed by plugging together computational facilities and resources offered by (possibly) untrusted providers. Illustrative examples of this approach are the Service Oriented, GRID and CLOUD paradigms. Since applications have not the full control of network facilities, security issues became even more acute. The literature has several proposals that address these problems. They can be roughly divided into dynamic, that monitor executions possibly stopping them when unsecure; and static, that analyse at binding time the published behavioural interfaces to avoid risky executions.

A language based approach supporting the static analysis of security has been developed in [12,11,10,9]. Its main ingredients are: local policies, call-by-contract invocation, type-effect systems, model checking and secure orchestration. However, this approach only takes into account qualitative aspects of behaviour, neglecting quantitative ones, typically the rates at which the different activities are performed. Indeed recently several quantitative models have been put forward, among which PEPA[23], PRISM[25], Stochastic π-calculus [17,30], to cite only a few.

In this paper we extend the approach of [10] to also deal with quantitative aspects. Our starting point is the abstraction of behaviour, called *history* *expressions*. Abstractions are processes of a suitable process calculus, and we associate a *rate* with its actions landing in the world of stochastic process calculi. Our first goal is to give history expressions a quantitative semantics in terms of continuous-time Markov chains (CTMC) so making usable well-known techniques for quantitative analysis [8,26,18]. We use a variation of the stochastic kernels over measurable spaces [15,29] to represent CTMC in the style of [17,16]. To overcome the difficulties with recursion, we restrict history expressions to a disciplined iteration, namely *binary Kleene star*. The first result of this paper is a CTMC semantics of our variant of history expressions, that turn out to be an extension of BPA^*_δ [22,7].

Our second main contribution is sharpening security policies with quantitative constraints. Roughly, quantitative policies are safety properties that enforce bounds on the speed at which actions have to be performed. These policies are first class operators inside history expressions, so that security can be taken into account from the very beginning of application development. To express policies we consider a linear subset of CSL [8,4].

Because of the inherent stochasticity of our programming model, policies are to be controlled in two complementary modalities: *potential* or *actual*. The first one applies to CTMC semantics, hence the check is on the *expected* behaviour — rates represent the *average* speed of actions. Potential analysis then measures the probability of policy violations. This kind of verification can be carried out through a probabilistic model checker [25].

The actual control can only be done dynamically, because in a specific, unlikely computation, the actual speed of an action can greatly deviate from its rate. Security is then enforced during the execution through an *execution monitor* aborting such an unlikely, unsafe computation.

Potential verification enables a user to accept/discard an application when the probability of a security violation is below/above a certain threshold he feels acceptable. Complementary, actual monitoring will stop the unwanted execution, so guaranteeing security.

1 Background

We review the main notions and notations about measure theory and we refer the reader to [3,2] for more details.

Given the *support set* $M \neq \emptyset$, a *σ-algebra* Σ over M is a set of subsets of M, the *measurable sets*, containing $\emptyset$ and closed under complement and countable union. The structure $\mathcal{M} = (M, \Sigma)$ is a *measurable space* and a *measure* over it is a function $\kappa : \Sigma \to \mathbb{R}^+ \cup \{\infty\}$ such that:

1. $\kappa(\emptyset) = 0$
2. Given a countable collection $\{N_i\}_{i \in I}$ of pairwise disjoint sets in Σ then: $\kappa(\cup_{i \in I} N_i) = \sum_{i \in I} \kappa(N_i)$ (*σ-additivity*)

The class of measures on a measurable space $\mathcal{M}$ will be denoted by $\Delta(M, \Sigma)$ or $\Delta(\mathcal{M})$ when the support set and the σ-algebra are clear from the context .

Given a class of sets G, called *generator*, $\sigma(G)$ is the minimal σ-algebra containing G. If G contains all pairwise disjoint sets then G is a *base* of $\sigma(G)$. Note that $\sigma(G)$ always exists since $\wp(G)$ contains G and the intersection of an arbitrary collection of σ-algebras is a σ-algebra.

Given two measurable spaces $(M_1, \Sigma_1), (M_2, \Sigma_2)$ a function $f : M_1 \to M_2$ is *measurable* iff $\forall A \in \Sigma_2 . f^{-1}(A) \in \Sigma_1$. The class $\|M_1 \to M_2\|$ contains the measurable functions between (M_1, Σ_1) and (M_2, Σ_2) omitting Σ_1, Σ_2 when unambiguous. A measurable function is structure-preserving.

Hereafter, whenever using a measurable space of measures $\kappa \in \Delta(M, \Sigma)$, we will consider the σ-algebra generated by the sets $\{\kappa \in \Delta(M, \Sigma) \mid \kappa(S) \geq r\}$ for arbitrary $S \in \Sigma, r > 0,$.

Given an arbitrary set $\Omega \neq \emptyset$, the *sample space*, a σ-algebra on it and a measure $\mathbb{P}$ such that $\mathbb{P}(\Omega) = 1$, we can build a probability space $(\Omega, \Pi, \mathbb{P})$ so interpreting standard probability theory in the measure theoretic context. For instance $\mathbb{P}(A)$ is the "probability of the events in A" with A measurable, i.e. $A \in \Pi$.

The random variables of probability theory can be defined and generalized in measure theory. A random variable X is a measurable function between (Ω, Π) and the measurable space formed by intervals in $\mathbb{R}$. The measure $\mathbb{P}$ governs the probability of $X^{-1}([a, b])$. Given a comparison symbol $\lhd \in \{\leq, \geq, <, >\}$, we abbreviate $\mathbb{P}(X^{-1}(\{x \mid x \lhd z\}))$ with $\mathbb{P}(X \lhd z)$.

A random variable of parameter λ has an exponential distribution if

$$\mathbb{P}(X \leq t) = \begin{cases} 1 - e^{-\lambda t} & \text{if } t \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Only exponentially distributed random variables enjoy the *memoryless property* $\mathbb{P}\left(X > s + t \mid X > t\right) = \mathbb{P}\left(X > s\right)$ for all $s, t \geq 0$; they have *mean* $\frac{1}{\lambda}$; if $\{X_i\}_{i \in I}$ is a finite set of such variables with parameter λ_i then $M = \min\{X_i\}_{i \in I}$ is an exponential random variable with parameter $\sum_{i \in I} \lambda_i$.

We now introduce Continuous-Time Markov chains on a countable state space. We refer to [28,24,3,8] for details.

Definition 1.1. *Let S be a denumerable set, a* Continuous-Time Stochastic Process *is an indexed (by $t \in \mathbb{R}^+$) family of random variables $X(t) : \Omega \to S$ with $(\Omega, B, \mathbb{P})$ probability space.*

The value of the variable X_t can be interpreted as the state of the process at time t. A continuous-time system evolves performing transitions between states. We limit ourself to specific class of processes with the *Markov* property.

Definition 1.2. *A* Continuous-Time Markov Chain *(CTMC) is a stochastic process $X(t)$ with $t \geq 0$ such that for any $s, t \geq 0$ and $i, j, x_u \in S$ the* Markov property *holds :*

$$\mathbb{P}\left(X(t+s) = j \mid X(s) = i, X(u) = x_u, 0 \leq u < s\right) = \mathbb{P}\left(X(t+s) = j \mid X(s) = i\right)$$

The CTMC is said to have homogeneous *transition probabilities if:*

$$\mathbb{P}(X(t+s) = i \mid X(s) = j) = \mathbb{P}(X(t) = i \mid X(0) = j)$$

Hereafter we will only use homogeneous CTMC.

Because of the Markov property, we denote with a random variable L_j, depending only on the current state j, the *sojourn time*: the amount of time spent in j before performing a new transition. Also we have that:

$$\mathbb{P}\left(L_i > s + v \mid L_i > s\right) = \mathbb{P}\left(L_i > v\right) \qquad \text{memoryless property of } L_i$$

Hence, it turns out that L_i must be *exponentially* distributed. When the process leaves state i, it can reach another state with a certain probability. This probability does not depend on the time spent in i. Hence we will indicate with p_{ij} the probability that the process reach state j from state i.

As a consequence, a CTMC is completely characterized by the parameters λ_i of the exponentially distributed random variables L_i and by p_{ij}. A well-know representation of a CTMC is the *rate-matrix* $R = [r_{ij}]$ with $r_{ij} = p_{ij}\lambda_i$.

This representation suggests another interpretation of the evolution of a CTMC. Since the minimum of a set $\{C_i\}_{i \in I}$ of exponentially distributed random variables with rate c_i is again an exponentially distributed random variable with rate $\sum_{i \in I} c_i$, we can interpret L_i as the minimum $\min\{R_{ij}\}$ of a set of random variables with rate r_{ij}. Hence, a CTMC models a process that, while entering in state i, enables a set of action, whose durations is modelled by the random variables $\{R_{ij}\}_{j \in S}$. These actions are competing (*racing*) for completion. If the action with duration R_{ij} is the faster, then the next state will be j.

Another common and useful characterization of CTMC is through the infinitesimal generator matrix Q, given in terms of the rate matrix:

Definition 1.3. *Let $D = [d_{ij}]$ be a matrix with $d_{ij} = 0$ if $i \neq j$ and $d_{ii} = \sum_{j \in S} r_{ij}$ otherwise. Then the* infinitesimal generator matrix *is $Q = R - D$.*

All previous definitions smoothly extend considering a function $L : S \times S \to A$ labelling transitions between states.

Let $p_{ij}(t) = \mathbb{P}(X(t) = j \mid X(0) = i)$ and $P(t) = [p_{ij}(t)]$ its matrix representation, a *steady distribution* π is a vector such that $\pi = \pi P(t)$ for all $t \geq 0$. The meaning is that if we use π as distribution of $X(0)$ then this will remain the same for all $t > 0$:

$$\mathbb{P}(X(t) = j) = \sum_{i \in S} \mathbb{P}(X(t) = j \mid X(0) = i) \cdot \mathbb{P}(X(0) = i) = \sum_{i \in S} p_{ij}(t)\pi_i = [\pi P(t)]_j = \pi_j$$

It is well know that the infinitesimal generator matrix plays almost the same role (in Continuous-Time) of the probability matrix of a Discrete-Time Markov Chain finding a steady distribution:

Fact. *Given a CTMC let Q be its infinitesimal generator matrix, then π is its steady distribution iff $\pi Q = 0$ and $\sum_{i \in S} \pi_i = 1$*

Such π always exists for a finite CTMC and it is unique if the CTMC is irreducible (if every state can be reached by a sequence of transitions from all other states). A steady distribution is important because a CTMC will reach it "on the long run" as shown by the following theorem:

Fact. *Given a CTMC, if it has a steady distribution* π*, then* $\forall i : \lim_{t\to\infty} p_{ij}(t) = \pi_j$

The notion of steady distribution is very important to analyse reliability and performance of a system modelled as a CTMC. In fact, π_j represents the proportion of time spent in state j on the long run. This information can be used to infer the typical behaviour of the system.

2 Stochastic history expressions

A language-based framework for managing security issues in a distributed contexts has been proposed in [12,11,10,9].

The starting point of these works is a functional programming language supporting remote service invocation and the enforcing of security policies. The execution of a distributed application comprise local and remote computations. Security relevant events generated during the executions of the application are collected in sequences, called *histories*. Security policies express constraints over these histories. Enforcing security can be done statically and dynamically. The dynamic mechanism uses a runtime monitor that blocks executions about to violate a security requirement. The static one uses a formalism, called *history expressions*, to represent all the histories that can be generated. These are then model checked to verify whether the constraints will be always satisfied. Their approach is qualitative only, here we present a first step towards a quantitative extension.

We first extend histories into *timed histories*. A timed history is a possibly empty sequence $((a, t_a), (b, t_b), \dots)$ of occurred events, with t_x duration of event x. Also security policies are extended to express timing constraints on timed histories. The values t_x are unpredictable on a single run of a program but we assume these duration to be exponentially distributed.

Under this assumption, we extend history expressions and obtain *stochastic history expressions* (HEμ). These express in a finite way potentially infinite timed histories and enable us to model check quantitative policies using well-known techniques. Indeed, the semantics of a HEμ process is given in terms of a CTMC, that implicitly describes both the timed histories and the long run behaviour of a program. It is convenient to use a functional representation of CTMC called *Markov kernel* (see Definition 2.4).

2.1 Syntax

The building blocks of HEμ are stochastic events. Given an alphabet of event names $\mathbb{A}$, a stochastic event is a pair $(a, \alpha) \in \mathbb{A} \times \mathbb{Q}^+$, where α is the rate of a, i.e. the parameter of the exponential random variable modelling its duration.

Definition 2.1. *A stochastic history expressions (*HEμ *) $h \in \mathbb{H}$ is a term generated by the following grammar:*

$$
\begin{array}{lllllll}
h_1, h_2 ::= & (a, \alpha) & \textit{(stochastic event)} & | & \delta & \textit{(deadlock)} & | \\
& \psi[e_1] & \textit{(policy framing)} & | & h_1 \cdot h_2 & \textit{(sequentialization)} & | \\
& h_1 + h_2 & \textit{(stochastic choice)} & | & (h_1 {}^* h_2) & \textit{(binary Kleene star)} &
\end{array}
$$

A stochastic event (a, α) performs action a and then successfully terminates. *Deadlock* δ is a non terminated process that cannot perform any action. The *stochastic choice* operator $+$ simultaneously enables two or more actions of processes h_1, h_2. We consider the enabled actions as competing: this means that the system is in a *race condition* and it hangs waiting for the fastest action to occur, while discarding the slower ones. We also consider a disciplined form of iteration: the binary Kleene star. It takes two processes and let them race. If the left process wins then it executes and the race starts again, otherwise the right executes and the iteration is over. We can express infinite behaviour as the *no-exit iteration* $(h_1 {}^* \delta)$ [13] that describes a process continuously doing h_1. For an overview about expressiveness of iteration, recursion, replication and a comparison between unary and binary star in classical process algebras see [21,6]. The sequentialization operator is often present in stochastic process algebras in its restricted variant of action *prefix.* However in [10] it is crucial to define the type-and-effect system that associates programs to history expressions. To manage sequentialization we will need the concept of terminated process, indicated with $\checkmark$, that cannot fire any action, still being different from δ. Indeed intuitively $\checkmark \cdot a = a$ while we will make sure that $\delta \cdot h = \delta$, see [1,5]. We remark that we will deal with $\checkmark$ into the semantic definitions and not in the syntax. In this we follow [1] where termination is treated as a meta-predicate over processes. Similarly to history expressions we attach policies ψ to expressions through the framing construct. We will define formally quantitative policies in Section 3.

We choose to stick on exponential distribution because the resulting mathematical theory enjoys elegant properties, e.g. the way we use to break the race condition. Other distributions can also be accommodated in our framework without much effort, especially because we neglect here an explicit parallel operator.

Note that the stochastic choice operator enables us to cast pure probabilistic branching in a stochastic setting. This behaviour can be simulated using stochastic choice in combination with high rate events such that the time consumed for their completion is negligible in the analysis, while providing the intended probabilistic behaviour.

In the literature there are many stochastic process algebras: PEPA [23], EMPA [14], stochastic π-calculus [30] and the stochastic version of CCS in [16], just to cite a few. Our HEμ algebra differs from these mainly because we offer full sequentialization, quantitative policy framing and the binary Kleene star. As a matter of fact, HEμ with no policy framing turns out to be to be a stochastic extension of BPA$^*_\delta$ [22,7].

2.2 Structural equivalence

We first define a structural congruence over the set of processes $\mathbb{H}$.

Definition 2.2. *We define the relation $\equiv$ as the smallest relation over $\mathbb{H}$ such that:*

- *It is an equivalence relation and a congruence with respect to $\cdot, +, \psi, {}^*$.*
- *It respects the following laws:*

$$\begin{aligned} \delta \cdot h_1 &\equiv \delta & h_1 + \delta &\equiv h_1 \\ (h_1 + h_2) \cdot h_3 &\equiv (h_1 \cdot h_3) + (h_2 \cdot h_3) & h_1 + h_2 &\equiv h_2 + h_1 \\ h_1 \cdot (h_2 \cdot h_3) &\equiv (h_1 \cdot h_2) \cdot h_3 & h_1 + (h_2 + h_3) &\equiv (h_1 + h_2) + h_3 \\ h_1 {}^* (h_2 \cdot h_3) &\equiv (h_1 {}^* h_2) \cdot h_3 & & \end{aligned}$$

Note that the law $h_1 \cdot (h_2 + h_3) \equiv (h_1 \cdot h_2) + (h_1 \cdot h_3)$ is missing. Indeed $(a, \alpha) \cdot ((b, \beta) + (c, \gamma)) \not\equiv (a, \alpha) \cdot (b, \beta) + (a, \alpha) \cdot (c, \gamma)$ because b and c are in a race condition within the left process.

We define now some quotient spaces with respect to the structural equivalence that will be used in the semantics:

Definition 2.3. *We define $\mathbb{H}^{\equiv}$ to be the set of $\equiv$-equivalence classes of $\mathbb{H}$ and $[h]^{\equiv}$ the equivalence class of $h \in \mathbb{H}$. Given the minimal σ-algebra $\Xi^{\equiv}$ generated by $\mathbb{H}^{\equiv}$ (i.e. $\Xi^{\equiv} = \sigma(\mathbb{H}^{\equiv})$), the measurable space $\mathcal{H}^{\equiv}$ is $\mathcal{H}^{\equiv} = (\mathbb{H}, \Xi^{\equiv})$. Finally, $\mathbb{H}_{\checkmark}$ indicates $\mathbb{H} \cup \{\checkmark\}$ and $\mathcal{H}^{\equiv}_{\checkmark} = (\mathbb{H}_{\checkmark}, \Xi^{\equiv}_{\checkmark})$ with $\Xi^{\equiv}_{\checkmark}$ the σ-algebra generated by $\mathbb{H}^{\equiv} \cup \{\checkmark\}$.*

2.3 Semantics

We give semantics of HEμ following the approach of [16,17]. We start with a slight variant of the *Markov Kernel* to accommodate termination and iteration. As a matter of fact, Markov Kernel is a labelled version of *Stochastic Kernel* introduced in [15,29].

Definition 2.4. *Given a measurable space $\mathcal{M} = (M, \Sigma)$ and the denumerable set $\mathbb{A}$ of event names, let $\Sigma' = \sigma(\Sigma \cup \{\checkmark\})$ be the smallest σ-algebra over $M' = M \cup \{\checkmark\}$ containing the sets in Σ and the singleton $\{\checkmark\}$. A* Markov kernel *is a triple $(\mathbb{A}, \mathcal{M}, \theta)$ where $\theta : \mathbb{A} \to \|M \to \Delta(M', \Sigma')\|$ is its* transition function. *A* Markov Process *is a quadruple $(\mathbb{A}, \mathcal{M}, \theta, m)$ where $m \in M$ is the initial state.*

To give a compact definition of semantics to HEμ, it is convenient to introduce some auxiliary notations. Recall that it suffices to define a measure on G to obtain its extension on $\sigma(G)$, which follows by σ-additivity.

- The *r-Dirac measure* on N is defined: $\delta^r_N(N') := \begin{cases} r & \text{if } N' = N \\ 0 & \text{otherwise} \end{cases} \quad \forall N' \in G$
- The *null measure* is defined: $\omega(N') := 0 \quad \forall N' \in G$

– Given an alphabet $\mathbb{A}$ we define the function $\omega^{\mathbb{A}} : \mathbb{A} \to \Delta(\mathcal{H}^{\equiv}_{\checkmark})$ such that $\omega^{\mathbb{A}}(x) = \omega$, with ω null measure on $\mathcal{H}^{\equiv}_{\checkmark}$.
– Given $a \in \mathbb{A}$ the function $[{}^{a}{}_{\kappa}] : \mathbb{A} \to \Delta(\mathcal{H}^{\equiv}_{\checkmark})$ is such that

$$[{}^{a}{}_{\kappa}](x) = \begin{cases} \kappa & \text{if } x = a \\ \omega & \text{otherwise} \end{cases}$$

In the following we will use this operator instantiated in $[{}^{a}{}_{\delta^{\alpha}_{\checkmark}}]$, with $\delta^{\alpha}_{\checkmark}$ α-Dirac delta measure on $\checkmark$.
– Given $h \in \mathbb{H}$, any function in the class of h-operator $\odot_h : \Delta(\mathcal{H}^{\equiv}_{\checkmark})^{\mathbb{A}} \to \Delta(\mathcal{H}^{\equiv}_{\checkmark})^{\mathbb{A}}$ is defined as follows:

$$[\odot_h \mu](a)(H) = \sum_{k \in H} \begin{cases} \mu(a)([l]^{\equiv}) & \text{if } \exists l \in \mathbb{H}. k \equiv l \cdot h \\ \mu(a)(\checkmark) & \text{if } k \equiv h, k \not\equiv \delta \\ 0 & \text{otherwise} \end{cases}$$

– The binary operator $\oplus : \Delta(\mathcal{H}^{\equiv}_{\checkmark})^{\mathbb{A}} \times \Delta(\mathcal{H}^{\equiv}_{\checkmark})^{\mathbb{A}} \to \Delta(\mathcal{H}^{\equiv}_{\checkmark})^{\mathbb{A}}$ is defined as follows:

$$(\mu \oplus \mu')(a)(H) = \mu(a)(H) + \mu'(a)(H)$$

The operators are well-defined and enjoy the following properties.

Lemma 2.1 (Properties of the operators).

1. $\odot_h \omega = \omega$
2. $\mu \oplus \mu' = \mu' \oplus \mu$
3. $(\mu \oplus \mu') \oplus \mu'' = \mu \oplus (\mu' \oplus \mu'')$
4. $\mu \oplus \omega^{\mathbb{A}} = \mu$
5. $\odot_h \mu = \odot_{h'} \mu$ *if* $h \equiv h'$
6. $\odot_h(\mu \oplus \mu') = (\odot_h \mu) \oplus (\odot_h \mu')$
7. $\odot_{(h_1 \cdot h_2)} \mu = \odot_{h_2} \odot_{h_1} \mu$

In our semantic context, as in [16], Structural Operational Semantics (SOS) rules are not used to give a pointwise semantics ($P \to Q$), rather they define a function that map processes to rate distributions (the *Markov kernel*).

We now introduce SOS rules to map HEμ processes to functions in $\Delta(\mathcal{H}^{\equiv}_{\checkmark})^{\mathbb{A}}$. Indeed we are defining a relation $\rightsquigarrow \subseteq \mathbb{H} \times [\mathbb{A} \to \Delta(\mathcal{H}^{\equiv}_{\checkmark})]$. If $(h, \mu) \in \rightsquigarrow$, the intended meaning of $\mu(a)(K)$ is the total apparent rate (sum of all rates) of an a-transition from h to a state in K.

Definition 2.5. *The relation* $\rightsquigarrow \subseteq \mathbb{H} \times [\mathbb{A} \to \Delta(\mathcal{H}^{\equiv}_{\checkmark})]$ *is the smallest relation satisfying the following rules:*

$$\textit{(ddk)}\frac{}{\delta \rightsquigarrow \omega^{\mathbb{A}}} \qquad \textit{(act)}\frac{}{(a,\alpha) \rightsquigarrow [{}^{a}{}_{\delta^{\alpha}_{\checkmark}}]} \qquad \textit{(cho)}\frac{h_1 \rightsquigarrow \mu_1 \quad h_2 \rightsquigarrow \mu_2}{h_1 + h_2 \rightsquigarrow \mu_1 \oplus \mu_2}$$

$$\textit{(seq)}\frac{h_1 \rightsquigarrow \mu}{h_1 \cdot h_2 \rightsquigarrow \odot_{h_2}(\mu)} \qquad \textit{(cnt)}\frac{h \rightsquigarrow \mu}{\psi[h] \rightsquigarrow \mu} \qquad \textit{(star)}\frac{h_1 \rightsquigarrow \mu_1 \quad h_2 \rightsquigarrow \mu_2}{h_1 {}^{*} h_2 \rightsquigarrow [\odot_{(h_1{}^{*}h_2)}\mu_1] \oplus \mu_2}$$

$$\text{(seq)}\ \dfrac{\text{(cho)}\ \dfrac{\text{(act)}\ \dfrac{\mu_1 = [{}^{\mathrm{a}}\delta^{2}_{\checkmark}]}{(\mathrm{a},2) \rightsquigarrow \mu_1} \qquad \text{(act)}\ \dfrac{\mu_2 = [{}^{\mathrm{b}}\delta^{1.5}_{\checkmark}]}{(\mathrm{b},1.5) \rightsquigarrow \mu_2}}{((\mathrm{a},2) + (\mathrm{b},1.5)) \rightsquigarrow \mu_1 \oplus \mu_2}}{((\mathrm{a},2) + (\mathrm{b},1.5)) \cdot (\mathrm{c},3) \rightsquigarrow \odot_{(\mathrm{c},3)}[\mu_1 \oplus \mu_2]}$$

Table 2.1: SOS derivation of $((\mathrm{a},2) + (\mathrm{b},1.5)) \cdot (\mathrm{c},3)$

We briefly comment on the above rules. The semantics of δ is a function that associates the null measure with any action in $\mathbb{A}$; therefore δ only has transition with rate 0, i.e. it can fire no transition. Instead, (a,α) has an a-transition towards $\checkmark$ with rate α, while the others have rate 0. Sequentialization of h_1, h_2 builds the associated function following a sort of continuation semantics. The quantitative policy ψ is neglected at this semantic level, yet maintaining at a syntactic level, so paving the way to subsequent security check. Our semantics of the choice operator diverges from known approaches (e.g. multi transition systems in PEPA [23], EMPA [14] and stochastic π-calculus [30]) because the non-determinism of SOS rules is substituted by a weighted functional approach, where single possibilities are rated and encoded in a function. This functional approach directly associates the correct rate with racing actions, while others need an additional normalization phase. For instance, an external observer looking at the racing process $(a,\alpha) + (a,\alpha)$ would see action a occurring at rate 2α, while a non-normalized transition system associates with a the rate α. Our SOS correctly associates to this process the function $\mu = [{}^{a}\delta^{\alpha}_{\checkmark}] \oplus [{}^{a}\delta^{\alpha}_{\checkmark}]$ such that $\mu(a)(\checkmark) = 2\alpha$. Back to the operational rules, the semantics of the Kleene star is a composition of $\odot$ and $\oplus$.

The following properties will be useful to build a Markov kernel for HEμ.

Theorem 2.1. *For any $h \in \mathbb{H}$ there exists a unique $\mu \in \Delta(\mathcal{H}^{\equiv}_{\checkmark})$ such that $P \rightsquigarrow \mu$.*

Example 1. Taken $h = ((\mathrm{a},2) + (\mathrm{b},1.5)) \cdot (\mathrm{c},3)$, Table 2.1 shows the derivation of $h \rightsquigarrow \odot_{(\mathrm{c},3)}[\mu_1 \oplus \mu_2]$, that indeed is a function. Then, we can use the classical tabular representation of functions in Table 2.2 to represent the meaning of h. We show entries only for non-zero values or non-null measures for singleton set of process.

We states that our semantics is correct with respect to structural equivalence.

Theorem 2.2. *If $h \equiv h'$ then $h \rightsquigarrow \mu$ and $h' \rightsquigarrow \mu$.*

Now we use the construction given in Definition 2.4 to eventually build the Markov Kernel for HEμ.

Theorem 2.3. *Let $h \in \mathbb{H}, a \in \mathbb{A}, H \in \Xi^{\equiv}, H \in \Xi^{\equiv}, \rho$ be a function such that $\rho(a)(h)(H) = \xi(h)(a)(H)$, where $\xi(h) = \mu$ whenever $h \rightsquigarrow \mu$, and $\mathbb{H}^{\equiv}, \Xi^{\equiv}$ are as in Definition 2.3. Then $(\mathbb{A}, \mathcal{H}^{\equiv}, \rho)$ is the* Markov kernel *associated with the HEμ.*

$$\mu_1 = \begin{array}{|c|c|}\hline a & \begin{array}{|c|c|}\hline [\checkmark]^{\equiv} & 2 \\ \hline \vdots & \vdots \\ \hline\end{array} \\ \hline \vdots & \vdots \\ \hline\end{array} \quad \mu_2 = \begin{array}{|c|c|}\hline b & \begin{array}{|c|c|}\hline [\checkmark]^{\equiv} & 1.5 \\ \hline \vdots & \vdots \\ \hline\end{array} \\ \hline \vdots & \vdots \\ \hline\end{array} \quad \mu_1 \oplus \mu_2 = \begin{array}{|c|c|}\hline a & \begin{array}{|c|c|}\hline [\checkmark]^{\equiv} & 2 \\ \hline \vdots & \vdots \\ \hline\end{array} \\ \hline b & \begin{array}{|c|c|}\hline [\checkmark]^{\equiv} & 1.5 \\ \hline \vdots & \vdots \\ \hline\end{array} \\ \hline \vdots & \vdots \\ \hline\end{array} \quad \odot_{(c,3)}[\mu_1 \oplus \mu_2] = \begin{array}{|c|c|}\hline a & \begin{array}{|c|c|}\hline [(c,3)]^{\equiv} & 2 \\ \hline \vdots & \vdots \\ \hline\end{array} \\ \hline b & \begin{array}{|c|c|}\hline [(c,3)]^{\equiv} & 1.5 \\ \hline \vdots & \vdots \\ \hline\end{array} \\ \hline \vdots & \vdots \\ \hline\end{array}$$

The function $\odot_{(c,3)}[\mu_1 \oplus \mu_2]$ given in the rightmost box reads: we perform action a with rate 2 or action b with rate 1.5; in both cases we reach state $(c, 3)$.

Table 2.2: Tabular representation of semantics

Eventually we associate a Markov Process with a HEμ h as follows:

Definition 2.6. *$P[\![h]\!]$ is the Markov Process $(\mathbb{A}, \mathcal{H}^{\equiv}, \xi, h)$.*

2.4 Rate Bisimulation of HE$\boldsymbol{\mu}$

The notion of behavioural equivalence is of primary importance because it highlights what really matters focusing objects at the right distance, and allows one to substitute equivalent processes preserving the overall behaviour. Additionally, some model checking techniques exploits behavioural equivalence to reduce the state space dimension and this is particularly important for our purpose verifications.

Structural equivalence is too weak. For instance consider the expressions $(a, 2\alpha)$ and $(a, \alpha) + (a, \alpha)$. They represents two process doing action a with the same apparent rate. Hence from an external point of view their behaviour is identical, but clearly $(a, 2\alpha) \not\equiv (a, \alpha) + (a, \alpha)$.

In this work we will use a specific type of bisimulation, globally recognized as the finest equivalence notion [31]. Rate bisimulation used here appears in [16] as a generalisation of *rate aware bisimulation* [19] and probabilistic bisimulation by *Larsen and Skou* [27].

Definition 2.7 (Rate bisimulation of HEμ). *A* rate bisimulation *is an equivalence relation $\mathfrak{R} \subseteq \mathbb{H}^{\equiv} \times \mathbb{H}^{\equiv}$ such that if $(h_1, h_2) \in \mathfrak{R}$ then for all $a \in \mathbb{A}$ and for all measurable subset H that are $\mathfrak{R}$-closed in $\Xi^{\equiv}_{\checkmark}$:*

$$\rho(a)(h_1)(H) = \rho(a)(h_2)(H)$$

Two histories h, h' are rate bisimilar *(written $h \sim h'$) iff there exist a rate bisimulation $\mathfrak{R}$ such that $(h, h') \in \mathfrak{R}$, thus $\sim$ (*bisimilarity*) is the union of all rate-bisimulations.*

As expected, rate bisimilarity is compatible with structural congruence and with the SOS semantics; also it is preserved under all the operator of HEμ, namely it is a congruence.

Theorem 2.4.

- *If $h \rightsquigarrow \mu$ and $h' \rightsquigarrow \mu$ then $h \sim h'$.*
- *If $h \equiv h'$ then $h \sim h'$.*
- *The relation $\sim$ is a congruence.*

3 Stochastic security policies

In this section we will introduce our notion of quantitative securities policy as constraints on the timed behaviour of processes along with two complementary ways to check them. Our first manner is *potential*: the check is made on the CTMC associated with a HEμ process h. This is because, if $h \rightsquigarrow \mu$, the function μ records the rates of the actions in h, and because these rates express the probability that the actions have to fire within a certain time interval.

Since probabilities are computed on the long run, in principle we would like to perform a security check on an always running system. The infinite behaviour of such system is indeed represented by the long run behaviour of the associated CTMC. Then we assume the semantics of the process h under analysis be an irreducible CTMC with a unique steady state. The potential check of security policies is therefore performed on steady states.

However, the duration of actions in an unlikely execution may greatly differ from the speed expresses by their rates. An *actual* check is therefore performed on the execution history $(a, t_a)(b, t_b)\dots$ by a monitor, which can abort the execution, when about to violate the security policy in hand. For this we assume that timed histories encompass all security relevant events. We shall formalise the above intuition below.

In the rest of this section, we assume as given a CTMC $\mathbf{O}$. Recall that we let $s_i \in S$ be the set of states of $\mathbf{O}$, $R = [r_{ij}]$ be its rate matrix, $Q = [q_{ij}]$ be its infinitesimal generator matrix, π be its steady state vector and $D = [d_{ij}]$ be the matrix in Definition 1.3, where, for notational convenience we let $D(i) = d_{ii}$. In addition $I = [a, b]$ is an interval in $\mathbb{R}^+$, when $\inf I = a \leq b = \sup I$, with possibly $b = \infty$.

3.1 Abstracting executions

A CTMC implicitly represents a set of timed histories associated with the *paths* we define below.

Definition 3.1 (Paths). *A* path $\sigma \in$ **Path** *over* $\mathbf{O}$ *is an infinite sequence* $\sigma = s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} s_2 \dots \xrightarrow{t_i} s_{i+1} \dots$ *with* $\forall i \in \mathbb{N}, s_i \in S$ *and* $t_i \in \mathbb{R}^+$ *such that* $r_{i,i+1} > 0$. *Given a labelling function* $L : S \times S \to \mathbb{A}$, *the path* σ *is associated with the timed history* $(a_0, t_0), (a_1, t_1), \dots$ *with* $a_i = L(s_i, s_{i+1})$. *We write* $\sigma[i]$ *for* s_i, $\delta(\sigma, i)$ *for* t_i *and* $\sigma@t$ *for the state* σ *at time* t.

We construct the following σ-algebra over paths in order to measure the probability of sets of timed histories.

Definition 3.2. *Let $p = (s_0, I_0, \ldots, I_{n-1}, s_n) \in P_I$ be a sequence of "intervals of paths", i.e. states and intervals, and let $C(p)$ be the* cylinder set*: consisting of all paths $\sigma \in \mathbf{Path}$ such that $\forall i \leq n.\sigma[i] = s_i$ and $\forall i < n.\delta(\sigma, i) \in I_i$. Final let $\Sigma_{\mathbf{Path}}$ be the σ-algebra generated by the base of cylinder sets $\{C(p)\}_{p \in P_I}$.*

Assume as given a cylinder set $C(p)$ and let τ be the initial probability distribution over states of the CTMC $\mathbf{O}$. We measure the probability of all paths $\sigma \in C(s_0, I_0, \ldots, I_{n-1}, s_n)$ assuming to be in state s_0 with probability $\tau(s_0)$. The sojourn time in state s_i is an exponentially distributed random variable with parameter $D(s_k) = \sum_{k \in S} r_{ik}$. The probability of leaving s_k in the interval I_k is

$$\int_I D(s_k) \cdot e^{-D(s_k) \cdot t}\, dt = e^{-D(s_k) \cdot \inf(I_k)} - e^{-D(s_k) \cdot \sup(b_k)} \quad \text{with } e^{-D(s_k) \cdot \infty} = 0$$

and the probability of choosing as next state s_{k+1} is $p_{k,k+1} = \frac{r_{k,k+1}}{D(s_k)}$.

Finally, the probability to follow a path in $C(s_0, I_0, \ldots, I_{k-1}, s_k)$ is inductively defined on the length of cylinder as:

$$\mathsf{P}_\tau\left(C\left(s_0\right)\right) = \tau(s_0)$$
$$\mathsf{P}_\tau\left(C\left(\ldots, s_k, I_k, s_{k+1}\right)\right) = \frac{r_{k,k+1}}{D(s_k)} \cdot \left(e^{-D(s_k) \cdot a_k} - e^{-D(s_k) \cdot b_k}\right) \cdot \mathsf{P}_\tau\left(C\left(\ldots, s_k\right)\right)$$

Let π be the steady state distribution of $\mathbf{O}$. The value of the probability $\mathsf{P}_\pi(C(p))$ is the portion of time spent by following the paths in the cylinder set p on the long run. In the following we will use P_s to denote P_τ, with $\tau(s) = 1$.

3.2 Actual and potential checks of quantitative policies

Here we define our stochastic security policies through a variant of Continuous Stochastic Logic (CSL) [20,8], that extends CTL. Our logic, called CSL_S, comprises path formulas and state formulas. Path formulas denote measurable unions of cylinder sets, while state formulas are propositions, the atoms of which constraint the given measure P_τ.

Definition 3.3. *State and path formulas are defined by:*

- *State formulas: $\upsilon, \upsilon' ::= tt \mid \curvearrowright^a \mid \neg\upsilon \mid \upsilon \wedge \upsilon' \mid \upsilon \vee \upsilon' \mid \mathcal{C}_{\leq c}(\iota)$*
- *Path formulas: $\iota, \iota' ::= X^I \upsilon \mid \upsilon U^I \upsilon'$*

The semantics of formulas is defined below over the given CTMC $\mathbf{O}$. Path formulas are evaluated over paths and state formulas are evaluated over states. Informally, $\mathcal{C}_{\leq c}(\iota)$ states that, on the long run, the portion of time spent doing any of the paths denoted by ι is bound by p. For simplicity we only use a $\leq$ bound, but of course we could add at no cost any other symbol of comparison, e.g. $>, \geq$. Moreover, since we focus on transitions rather than on states, in CSL_S we "label" states with their outgoing transition using a class of predicates $\curvearrowright^a, a \in \mathbb{A}$.

The operator *next* $X^I \upsilon$ describes paths that start with a transition leading to a state where υ holds, and with duration in the interval I. The *until* operator $\upsilon U^I \upsilon'$ describes paths along states where υ does not hold until a transition leads to a state where υ' holds after a time in the interval I.

Definition 3.4. *The semantics of state formulas is evaluated over states $s \in S$ of* $\mathbf{O}$*; below let* $Prb(s, \iota) = \mathsf{P}_s(\{\sigma \mid \sigma \models \iota\})$*:*

$$\begin{array}{llll} s \models \mathrm{tt} & \textit{always true} & s \models \upsilon \wedge \upsilon' & \textit{iff } s \models \upsilon \textit{ and } s \models \upsilon' \\ s \models \neg\upsilon & \textit{iff } s \not\models \upsilon & s \models \mathcal{C}_{\leq c}(\iota) & \textit{iff } \pi_s \times Prb(s,\iota) \lhd p \\ s \models \curvearrowright^a & \textit{iff a transition from } s \textit{ labelled } a \textit{ exists} & & \end{array}$$

Path formulas are evaluated over the paths of $\mathbf{O}$*:*

$$\begin{array}{ll} \sigma \models X^I \upsilon & \textit{iff } \sigma[1] \textit{ is defined } \wedge \sigma[1] \models \upsilon \wedge \delta(\sigma, 0) \in I \\ \sigma \models \upsilon U^I \upsilon' & \textit{iff } \exists t \in I.\sigma@t \models \upsilon' \wedge (\forall t' \in [0,t).\sigma@t' \models \upsilon) \end{array}$$

As proved in [20], the set $\{\sigma \mid \sigma \models \iota\}$ turns out to be measurable, hence ι *denotes* a measurable set of paths (this also implies that the definition of Prb is correct).

We briefly comment on the definition for $\mathcal{C}_{\leq c}(\iota)$, recalling that ι represents a set of paths, π_s the portion of time spent in state s on the long run, while $Prb(s, \iota)$ is the probability, once in s, of doing a path belonging to ι. Their product gives the portion of time spent doing a path denoted by ι on the long-run.

We do not present here a procedure for operationally verifying a state formula. We refer the interested reader to [8], that gives a fixpoint characterization of Prb.

We define now quantitative policy ψ as a CSL_S formula of the form

$$\psi = \mathcal{C}_{\leq c}(\iota) \text{ with operator } \mathcal{C} \text{ no longer occurring in } \iota$$

A policy of this form endows a path formula ι, denoting a measurable set of undesired paths, wrapped up by the operator $\mathcal{C}_{\leq c}$. Obviously c bounds the probability of all paths denoted by ι.

Summing up, with the above definitions we can explain how actual and potential checks works. The actual check requires an execution monitor that watches the computation and aborts it whenever the generated timed history is about to fall in the set described by ι. This kind of monitoring causes a performance degradation because it should be always enabled.

The potential control is done by checking the semantics $P[\![h]\!]$ of a process h, the model, against the policies to be obeyed. We say that h respects all policies occurring in it if and only if for all sub-expressions of the form $\psi_i[h_i].\psi_i[h_i] \models \psi$. Needless to say, the generation of the CTMC and of its steady state can be easily done by following the semantic definitions of Section 2.3 and by using standard packages for numerical computations. The verification is then completed by a suitable combination of the algorithms in [8] with standard model checking tools [25].

We now suggest a complementary usage of the two different ways of verifying policies put forward above. The result of a potential check can be interpreted as a bound over the probability of a monitor intervention. Indeed, suppose that $\mathcal{C}_{\leq c}(\iota)$ is verified true. The system will then execute an offending run, belonging to ι, in a percentage of its time smaller than c. A user can consciously decide

Event	Description	Event	Description
CIC,1	Customer insert card	RO,2	Ask for offers remotely
POT,3	Print offer on the ticket	RTM,2	Request MAN transaction
RTI,1.5	Request Internet transaction	BIT,1	Opening Internet transaction
BMT,1	Opening MAN transaction	DT,2	Executing transaction operations
ET,1	Closing transaction		

Table 4.1: Events and their description.

to activate the run-time monitor, based on this information, as well as on the risks that a possible violation may cause. If the risk is acceptable, the user can instead deactivate the monitor, so freeing the system from the induced overhead. Additionally, as potential analysis bounds the time spent in unsafe computations, one can evaluate the performance and reliability of his system, by bounding the time lost in computations that will be aborted. Finally, by decreasing the value of the parameter c, we can determine the minimum value for $\mathcal{C}_{\leq c}$ to be true — and give hints to the designer on which parts of the system need security-improving refinements.

4 A working example

A shop in Milan, called *Vestiti*, is part of dress brand chain. The shop database is mirrored in two servers: one in Milan and one in Rome. The shop is connected to both: to the one in Milan through a private MAN and to the one in Rome through Internet. The shop can communicate with the server in Milan with low latency, but sometimes it could happen that the one in Rome performs better.

When an item is sold, the shop updates a single remote database, as they are autonomously mirrored. The manager of *Vestiti* requires the payment system to ask, at the moment of a payment, both servers and then to choose the fastest to answer. Then a cash transaction occurs: the client inserts her fidelity card; the payment system asks for offers reserved to that client; and a remainder of the offer is printed at the top of the receipt. Then the system asks both servers for a new transaction to update the database with the items sold. This race is won by the first server that answers.

Assuming as given the set of actions and rates in Table 4.1, we formalize the above as follows:

$$h_{Vestiti} = \Big((\mathrm{CIC},1)\cdot(\mathrm{RO},2)\cdot(\mathrm{POT},3)\cdot\psi\Big[\big((\mathrm{RTM},2)\cdot(\mathrm{BMT},1)+(\mathrm{RTI},1.5)\cdot(\mathrm{BIT},1)\big)\cdot(\mathrm{DT},2)\cdot(\mathrm{ET},1)\Big]\Big)^{*}\delta$$

However, the CEO of the dress brand is scared by using an Internet connection, that he considers much more unreliable than their own MAN. To stay on the safe side, the CEO asks the manager of *Vestiti* to enforce a security policy, so to abort internet transaction lasting more than three seconds.

The policy ψ expressing the requirement of the manager is formally rendered by the CSL_S expression:

$$\psi = \mathcal{C}_{\leq 0.01}\left(\neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[3,\infty]}\ \curvearrowright^{CIC}\right)$$

This policy states that we require a system not to spend more than 1% portion of time doing an Internet transaction longer than 3 seconds. In other words, only 1% of computational time will be spent by a run that the security monitor will abort.

After some easy calculations, involving the computation of the steady state distribution π of the CTMC associated with $h_{Vestiti}$, we obtain that

$$\psi[\dots] \models \mathcal{C}_{\leq 0.01}(\neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[3,\infty]}\ \curvearrowright^{CIC}) \text{ iff}$$
$$\pi_{S4} \times Prb(\psi[\dots], \neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[3,\infty]}\ \curvearrowright^{CIC}) \leq 0.01$$

Now

$$Prb(\psi[\dots], \neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[3,\infty]}\ \curvearrowright^{CIC}) = 0.12 \text{ and } \pi_{S4} = 0.06$$

the policy is respected, because $0.06 \times 0.12 = 0.0072 \leq 0.01$.

Then, our analysis shows that the manager of *Vestiti* did a good job: the payment system of his shop always uses the fastest server available at the moment. The static check guarantees that system is quite reliable even without a security monitor switched on, because there is a low probability of violating the policy, i.e. 0.0072%. If the manager still feels unsecure and activates a security monitor, we can estimate that in a period of one hour, approximately less that 30 seconds are lost serving a payment that will result in a security exception.

Conclusions

In this paper we addressed the problem of expressing and enforcing non-functional security policies on programs. In particular we focused on quantitative security policies which constraint program behaviour over time. Our approach is based on the stochastic process algebra HEμ to abstract programs behaviour. The calculus endows the binary Kleene star iteration operator and a full-fledged sequentialization operator. The semantics of HEμ has been given in terms of CTMC using the approach of [17,16]. Security policies are expressed as formulae of CSL_S predicates over CTMCs. We plan to integrate our quantitative security policies in the language-based security framework of [12,11,10,9]. In this approach programs are typed as functions with a side effect that abstractly describes the possible run-time executions of the program. Security policies are properties over effects and model-checking techniques are used to control statically whether or not the program satisfied the security policies on demands. We plan to exploit HEμ to represent quantitative effects of programs.

References

1. L. Aceto and M. Hennessy. Termination, deadlock and divergence. In *Mathematical Foundations of Programming Semantics*, pages 301–318. Springer, 1989.
2. R.B. Ash and C. Doléans-Dade. *Probability and measure theory.* Academic Press, 2000.

3. K.B. Athreya and S.N. Lahiri. *Measure theory and probability theory.* Springer-Verlag New York Inc, 2006.
4. A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model-checking continuous-time Markov chains. *ACM Transactions on Computational Logic (TOCL)*, 1(1):170, 2000.
5. J.C.M. Baeten. Process algebra with explicit termination. Technical report, 2000.
6. JCM Baeten and F. Corradini. Regular expressions in process algebra. In *Logic in Computer Science, 2005. LICS 2005. Proceedings. 20th Annual IEEE Symposium on*, pages 12–19, 2005.
7. J.C.M. Baeten and W.P. Weijland. *Process algebra.* Cambridge University Press Cambridge, 1990.
8. C. Baier, B. Haverkort, H. Hermanns, and J.P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on software engineering*, 29(6):524–541, 2003.
9. M. Bartoletti, P. Degano, and G.L. Ferrari. Types and effects for secure service orchestration. In *CSFW*, pages 57–69. IEEE Computer Society, 2006.
10. M. Bartoletti, P. Degano, and G.L. Ferrari. Planning and verifying service composition. *Journal of Computer Security*, 17(5):799–837, 2009.
11. M. Bartoletti, P. Degano, G.L. Ferrari, and R. Zunino. Semantics-based design for secure web services. *IEEE Trans. Software Eng.*, 34(1):33–49, 2008.
12. M. Bartoletti, P. Degano, G.L. Ferrari, and R. Zunino. Local policies for resource usage analysis. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 31(6):23, 2009.
13. JA Bergstra, A. Ponse, and S.A. Smolka. *Handbook of process algebra.* Elsevier Science Ltd, 2001.
14. M. Bernardo and R. Gorrieri. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 202(1-2):1–54, 1998.
15. R. Blute, J. Desharnais, A. Edalat, and P. Panangaden. Bisimulation for labelled Markov processes. 1997.
16. Luca Cardelli and Radu Mardare. The measurable space of stochastic processes. In *QEST*, pages 171–180. IEEE Computer Society, 2010.
17. Luca Cardelli and Radu Mardare. Stochastic pi-calculus revisited. Unpublished, 2010.
18. G. Clark, S. Gilmore, and J. Hillston. Specifying performance measures for PEPA. *Formal Methods for Real-Time and Probabilistic Systems*, pages 211–227.
19. R. De Nicola, D. Latella, M. Loreti, and M. Massink. Rate-Based Transition Systems for Stochastic Process Calculi. *Automata, Languages and Programming*, 2009.
20. J. Desharnais and P. Panangaden. Continuous stochastic logic characterizes bisimulation of continuous-time Markov processes. *Journal of Logic and Algebraic Programming*, 56(1-2), 2003.
21. W. Fokkink. Axiomatizations for the perpetual loop in process algebra. *Automata, Languages and Programming*, pages 571–581.
22. W. Fokkink and H. Zantema. Basic process algebra with iteration: Completeness of its equational axioms. *The Computer Journal*, 37(4):259, 1994.
23. J. Hillston. *A compositional approach to performance modelling.* Cambridge Univ Pr, 1996.
24. J.G. Kemeny, J.L. Snell, and A.W. Knapp. *Denumerable markov chains.* Springer, 1976.
25. M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. *Computer Performance Evaluation: Modelling Techniques and Tools*, 2002.
26. M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. *Formal Methods for Performance Evaluation*, pages 220–270, 2007.
27. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.
28. J.R. Norris. *Markov chains.* Cambridge Univ Pr, 1998.
29. P. Panangaden. *Labelled Markov Processes.* Imperial College Press, 2009.
30. C. Priami. Stochastic π-calculus. *The Computer Journal*, 38(7):578, 1995.
31. D. Sangiorgi. On the origins of bisimulation and coinduction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 31(4):1–41, 2009.

A Appendix

We collect in this appendix the proofs of our results and a detailed representation of our working example.

A.1 Proofs

Proof of Theorem (2.1). By structural induction:

- case $h = \delta$: Immediate by rule (ddk).
- case $h = (a, \alpha)$: Immediate by rule (act).
- case $h = h_1 \cdot h_2$: Immediate by rule (seq) and by the inductive hypothesis on h_1.
- case $h = h_1 + h_2$: Immediate by rule (cho) and by the inductive hypothesis on h_1, h_2.
- case $h = h_1 {}^* h_2$: Immediate by rule (star) and by the inductive hypothesis on h_1, h_2.
- case $h = \psi[h_1]$: Immediate by rule (cnt) and by the inductive hypothesis on h_1.

□

Proof of Theorem (2.3). It suffices to show that for each $h \in \mathbb{H}$ and $a \in \mathbb{A}$, $\rho(a)(h)$ is a measure in $\Delta(\mathcal{K}^{\equiv}_{\checkmark})^{\mathbb{A}}$. The proof is straightforward by structural induction on terms. □

Proof of Lemma (2.1).
The only non trivial cases are 5, 6 and 7

5 $\odot_h \mu = \odot_{h'} \mu$ if $h \equiv h'$

$$[\odot_h \mu](a)(K) \stackrel{\text{def}}{=}$$

$$\stackrel{\text{def}}{=} \sum_{k \in K} \begin{cases} \mu(a)([l]^{\equiv}) & \text{if } \exists l \in \mathbb{H}. k \equiv l \cdot h \\ \mu(a)(\checkmark) & \text{if } k \equiv h, k \not\equiv \delta \\ 0 & \text{otherwise} \end{cases} =$$

$$= \sum_{k \in K} \begin{cases} \mu(a)([l]^{\equiv}) & \text{if } \exists l \in \mathbb{H}. k \equiv l \cdot h' \\ \mu(a)(\checkmark) & \text{if } k \equiv h', k \not\equiv \delta \\ 0 & \text{otherwise} \end{cases} \stackrel{\text{def}}{=}$$

$$\stackrel{\text{def}}{=} [\odot_{h'} \mu](a)(K)$$

Since From from definitions and the fact that $\equiv$ is a congruence follows:
$k \equiv l \cdot h \Leftrightarrow k \equiv l \cdot h'$ (since $l \cdot h \equiv l \cdot h'$)
$k \equiv h \Leftrightarrow k \equiv h'$

6 $\odot_h(\mu \oplus \mu') = (\odot_h \mu) \oplus (\odot_h \mu')$

$$\odot_h\ (\mu \oplus \mu')(a)(K) \overset{\text{def}}{=}$$

$$\overset{\text{def}}{=} \sum_{k\in K} \begin{cases} (\mu \oplus \mu')(a)([l]^{\equiv}) & \text{if } \exists l \in \mathbb{H}. k \equiv l \cdot h \\ (\mu \oplus \mu')(a)(\checkmark) & \text{if } k \equiv h, k \not\equiv \delta \\ 0 & \text{otherwise} \end{cases} \overset{\text{def}}{=}$$

$$\overset{\text{def}}{=} \sum_{k\in K} \begin{cases} \mu(a)(\bar{l}) + \mu'(a)(\bar{l}) & \text{if } \exists l \in \mathbb{H}. k \equiv l \cdot h \\ \mu(a)(\checkmark) + \mu'(a)(\checkmark) & \text{if } k \equiv h, k \not\equiv \delta \\ 0 \quad \text{otherwise} \end{cases} =$$

$$= \sum_{k\in K} \begin{cases} \mu(a)([l]^{\equiv}) & \text{if } \exists l \in \mathbb{H}. k \equiv l \cdot h \\ \mu(a)(\checkmark) & \text{if } k \equiv h, k \not\equiv \delta \\ 0 & \text{otherwise} \end{cases} + \sum_{k\in K} \begin{cases} \mu'(a)([l]^{\equiv}) & \text{if } \exists l \in \mathbb{H}. k \equiv l \cdot h \\ \mu'(a)(\checkmark) & \text{if } k \equiv h, k \not\equiv \delta \\ 0 & \text{otherwise} \end{cases} \overset{\text{def}}{=}$$

$$\overset{\text{def}}{=} ((\odot_h \mu) \oplus (\odot_h \mu'))(a)(K)$$

7 $\odot_{(h_1 \cdot h_2)} \mu = \odot_{h_2} \odot_{h_1} \mu$
Let

$$\gamma(a)(K) = [\odot_{h_1}\mu](a)(K) \overset{\text{def}}{=} \sum_{k\in K} \begin{cases} \mu(a)([l]^{\equiv}) & \text{if } \exists l \in \mathbb{H}. k \equiv l \cdot h_1 \\ \mu(a)(\checkmark) & \text{if } k \equiv h_1, k \not\equiv \delta \\ 0 & \text{otherwise} \end{cases}$$

then

$$[\odot_{h_2}\gamma](a)(K) \overset{\text{def}}{=} \sum_{k\in K} \begin{cases} \gamma(a)([l]^{\equiv}) & \text{if } \exists l \in \mathbb{H}. k \equiv l \cdot h_2 \\ \gamma(a)(\checkmark) & \text{if } k \equiv h_2, k \not\equiv \delta \\ 0 & \text{otherwise} \end{cases}$$

while

$$[\odot_{(h_1 \cdot h_2)}\mu](a)(K) \overset{\text{def}}{=} \sum_{k\in K} \begin{cases} \mu(a)([l]^{\equiv}) & \text{if } \exists l \in \mathbb{H}. k \equiv l \cdot (h_1 \cdot h_2) \\ \mu(a)(\checkmark) & \text{if } k \equiv (h_1 \cdot h_2), k \not\equiv \delta \\ 0 & \text{otherwise} \end{cases}$$

Reasoning by case (on k) considering singleton set $K = \{k\}$ we get the result.
(a) case $k \equiv h^{'} \cdot h_2$ for some h'
  i. case $h' \equiv h_1$
  $h' \equiv h_1 \Rightarrow k \equiv h_1 \cdot h_2 \Rightarrow (k \equiv l \cdot h_2 \Rightarrow l \equiv h_1)$
  $[\odot_{h_2}\gamma](a)(k) = \gamma(a)(h_1) = \mu(a)(\checkmark)$

  $[\odot_{h_1 \cdot h_2}\mu](a)(k) = \mu(a)(\checkmark)$
  ii. case $h' \not\equiv h_1$

A. case $h' \equiv l' \cdot h_1$ for some l'
$[\odot_{h_2}\gamma](a)(k) = \gamma(a)(l' \cdot h_1) = \mu(a)(l')$

$h' \equiv l' \cdot h_1 \Rightarrow k \equiv (l' \cdot h_1) \cdot h_2 \equiv l' \cdot (h_1 \cdot h_2)$
$[\odot_{h_1 \cdot h_2}\mu](a)(k) = \mu(a)(l')$

B. case $h' \not\equiv l' \cdot h_1$ for all l'
$[\odot_{h_2}\gamma](a)(k) = \gamma(a)(h') = 0$

$\exists l. k \equiv l \cdot (h_1 \cdot h_2) \Rightarrow k \equiv (l \cdot h_1) \cdot h_2 \Rightarrow h' \equiv l \cdot h_1 \quad \bot$
$k \equiv h_1 \cdot h_2 \Rightarrow h' \equiv h_1 \quad \bot$
$[\odot_{h_1 \cdot h_2}\mu](a)(k) = 0$

(b) case $k \not\equiv h \cdot h_2$ for all h

$$[\odot_{h_2}\gamma](a)(k) = \begin{cases} \gamma(a)(\checkmark) & \text{if } k \equiv h_2 \\ 0 & \text{otherwise} \end{cases} = 0$$

$[\odot_{h_1 \cdot h_2}\mu](a)(k) = 0$

□

Proof of Theorem (2.2).
By induction on rules of $\equiv$ showing that property preserves from assumptions (of existing equivalences) to conclusion (of new equivalences).

- case $h_1 \equiv h_1$: Immediate.
- case $h_1 \equiv h_2$ implies $h_2 \equiv h_1$:
  by inductive hypothesis $h_1 \rightsquigarrow \mu, h_2 \rightsquigarrow \mu$.
- case $h_1 \equiv h_2$ and $h_2 \equiv h_3$ implies $h_1 \equiv h_3$:
  by inductive hypothesis $h_1 \rightsquigarrow \mu, h_2 \rightsquigarrow \mu, h_3 \rightsquigarrow \mu$.
- case $h_1 \equiv h_1'$ implies $\psi[h_1] \equiv \psi[h_1']$:
  thesis immediate by inductive hypothesis and definitions.
- case $h_1 \equiv h_1', h_2 \equiv h_2'$ implies $h_1 \, {}^* \, h_2 \equiv h_1' \, {}^* \, h_2'$:
  thesis follows by definitions and inductive hypothesis showing that $\odot_{(h_1 {}^* h_2)}\mu_1 \oplus \mu_2 = \odot_{(h_1' {}^* h_2')}\mu_1 \oplus \mu_2$ by properties of operators.
- case $h_1 \equiv h_1', h_2 \equiv h_2'$ implies $h_1 \cdot h_2 \equiv h_1' \cdot h_2'$:
  thesis follows by definitions and inductive hypothesis showing that $\odot_{h_2}\mu_1 = \odot_{h_2'}\mu_1$ by properties of operators.
- case $h_1 \equiv h_1', h_2 \equiv h_2'$ implies $h_1 + h_2 \equiv h_1' + h_2'$:
  thesis follows by definitions and inductive hypothesis showing that $\mu_1 \oplus \mu_2 = \mu_1 \oplus \mu_2$.
- case $\delta \cdot h_1 \equiv \delta$:
  knowing that $\delta \rightsquigarrow \omega^{\mathbb{A}}$, thesis follows observing that $\odot_{h_1}\omega^{\mathbb{A}} = \omega^{\mathbb{A}}$.
- case $h_1 + \delta \equiv h_1$:
  knowing that $\delta \rightsquigarrow \omega^{\mathbb{A}}$ and $h_1 \rightsquigarrow \mu_1$ thesis follows by property that shows $\omega^{\mathbb{A}} \oplus \mu_1 = \mu_1$.
- case $h_1 + h_2 \equiv h_2 + h_1$:
  knowing that $h_1 \rightsquigarrow \mu_1, h_2 \rightsquigarrow \mu_2$, result follow by commutativity of operator $\oplus$.

- case $(h_1 + h_2) \cdot h_3 \equiv h_1 \cdot h_3 + h_2 \cdot h_3$:
  knowing that $h_1 \rightsquigarrow \mu_1, h_2 \rightsquigarrow \mu_2, h_3 \rightsquigarrow \mu_3$ and applying definitions we get $(h_1 + h_2) \cdot h_3 \rightsquigarrow \odot_{h_3}(\mu_1 \oplus \mu_2)$ and $h_1 \cdot h_3 + h_2 \cdot h_3 \rightsquigarrow \odot_{h_3}\mu_1 \oplus \odot_{h_3}\mu_2$. Thesis $\odot_{h_3}(\mu_1 \oplus \mu_2) = \odot_{h_3}\mu_1 \oplus \odot_{h_3}\mu_2$ follows by properties of operators.
- case $h_1 \cdot (h_2 \cdot h_3) \equiv (h_1 \cdot h_2) \cdot h_3$:
  knowing that $h_1 \rightsquigarrow \mu_1, h_2 \rightsquigarrow \mu_2, h_3 \rightsquigarrow \mu_3$ and applying definitions we get $h_1 \cdot (h_2 \cdot h_3) \rightsquigarrow \odot_{h_2 \cdot h_3}\mu_1$ and $(h_1 \cdot h_2) \cdot h_3 \rightsquigarrow \odot_{h_1} \odot_{h_2} \mu_1$. Thesis $\odot_{h_2 \cdot h_3}\mu_1 = \odot_{h_1} \odot_{h_2} \mu_1$ follows by properties of operator.
- case $h_1 + (h_2 + h_3) \equiv (h_1 + h_2) + h_3$:
  knowing that $h_1 \rightsquigarrow \mu_1, h_2 \rightsquigarrow \mu_2, h_3 \rightsquigarrow \mu_3$, result follow by associativity of operator $\oplus$.
- case $h_1 {}^* (h_2 \cdot h_3) \equiv (h_1 {}^* h_2) \cdot h_3$:
  knowing that $h_1 \rightsquigarrow \mu_1, h_2 \rightsquigarrow \mu_2, h_3 \rightsquigarrow \mu_3$ and applying definitions we get $h_1 {}^* (h_2 \cdot h_3) \rightsquigarrow \odot_{(h_1 {}^* (h_2 \cdot h_3))}\mu_1 \oplus \odot_{h_3}\mu_2$ and $(h_1 {}^* h_2) \cdot h_3 \rightsquigarrow \odot_{h_3}(\odot_{(h_1 {}^* h_2)}\mu_1 \oplus \mu_2)$. Thesis follows by $\odot_{(h_1 {}^* (h_2 \cdot h_3))}\mu_1 \oplus \odot_{h_3}\mu_2 = \odot_{(h_1 {}^* h_2) \cdot h_3}\mu_1 \oplus \odot_{h_3}\mu_2 = \odot_{h_3} \odot_{(h_1 {}^* h_2)} \mu_1 \oplus \odot_{h_3}\mu_2 = \odot_{h_3}(\odot_{(h_1 {}^* h_2)}\mu_1 \oplus \mu_2)$ and properties of operators.

□

Lemma A.1. *$\forall h, h', l, l'. h \sim h'$ and $l \sim l' \Rightarrow h * l \sim h' * l'$*

Proof. Let $\mathfrak{H}$ be the rate-bisimulation between h and h' and let $\mathfrak{L}$ be that between l and l'. We define the operator $\bullet : (\mathbb{H}_{\checkmark} \times \mathbb{H}_{\checkmark}) \times \mathbb{H}_{\checkmark} \times \mathbb{H}_{\checkmark} \to (\mathbb{H}_{\checkmark} \times \mathbb{H}_{\checkmark})$ such that:

$$\bullet(\mathfrak{Q}, d, d') = \{(z, z') \mid \exists (c, c') \in \mathfrak{Q}. c, c' \neq \checkmark \text{ and } z = c \cdot d \text{ and } z' = c' \cdot d'\} \cup \{(d, d')\}$$

We construct a new rate bisimulation $\mathfrak{S}$ for $(h * l)$ and $(h' * l')$ as the smallest equivalence relations such that:

$$\mathfrak{S} \supseteq \mathfrak{H} \cup \mathfrak{L} \cup \bullet(\mathfrak{H}, (h * l), (h' * l'))$$

It remains to prove that $(z, z') \in \mathfrak{S} \Rightarrow$ any $\mathfrak{S}$-closed $S \in \mathbb{H}$ and any $a \in \mathbb{A}$ $\xi(z)(a)(S) = \xi(z')(a)(S)$ with $\xi(z) = \mu, (z') = \mu'$.
If $(z, z') \in \mathfrak{S}$ then we get three cases:

- *if $(z, z') \in \mathfrak{H}$ result follows by the fact that $\mathfrak{S} \supseteq \mathfrak{H}$ and so any $\mathfrak{S}$-closed $S \in \mathbb{H}$ is $\mathfrak{H}-$closed. It follows $\mu(a)(S) = \mu'(a)(S)$*
- *if $(z, z') \in \mathfrak{L}$: proof is the same.*
- *if $(z, z') \in \bullet(\mathfrak{H}, (h * l), (h' * l'))$ then*
  - *$z = c \cdot s$ and $z' = c' \cdot (h' * l')$ with $(c, c') \in \mathfrak{H}$. Using semantics rules $z \rightsquigarrow \odot_{(h * l)}\mu$ and $z \rightsquigarrow \odot_{(h' * l')}\mu'$. Using definition*

$$[\odot_{(h*l)}\mu](a)(S) = \sum_{p \in S} \begin{cases} \mu(a)([u]^{\equiv}) & \text{if } \exists u \in \mathbb{H}. p \equiv u \cdot (h * l) \\ \mu(a)(\checkmark) & \text{if } p \equiv (h * l), p \not\equiv \delta \\ 0 & \text{otherwise} \end{cases} =$$

$$= \sum_{u \in U} \mu(a)(u) + \mu(a)(\checkmark) = \mu(a)(U) + \mu(a)(\checkmark)$$

for $U = \{u \mid \exists p \in S.p \equiv u \cdot (h * l)\}$ and

$$[\odot_{(h'*l')}\mu'](a)(S) =$$

$$= \sum_{p\in S} \begin{cases} \mu'(a)([u']^{\equiv}) & \text{if } \exists u' \in \mathbb{H}.p \equiv u' \cdot (h' * l') \\ \mu'(a)(\checkmark) & \text{if } p \equiv (h' * l'), p \not\equiv \delta \\ 0 & \text{otherwise} \end{cases} =$$

$$= \sum_{u'\in U'} \mu'(a)(u') + \mu(a)(\checkmark) = \mu'(a)(U') + \mu'(a)(\checkmark)$$

with $U' = \{u' \mid \exists p \in S.p \equiv u' \cdot (h' * l')\}$. Thesis follows pointing out that $U = U'$ and U, U' are $\mathfrak{H}-$closed.

- *$z = (h * l)$ and $z' = (h' * l')$. It follows by operational rules that $z \rightsquigarrow \odot_{(h*l)}\mu \oplus \nu, z' \rightsquigarrow \odot_{(h*l)}\mu' \oplus \nu'$ with $h \rightsquigarrow \mu, h' \rightsquigarrow \mu', l \rightsquigarrow \nu, l' \rightsquigarrow \nu'$. Using the same argumentation as before, as S is $\mathfrak{L}$-closed, $\nu(a)(S) = \nu'(a)(S)$. As already shown $[\odot_{(h*l)}\mu](a)(S) = [\odot_{(h'*l')}\mu'](a)(S)$.*

– *if $(z, z') \in \mathfrak{S} \setminus (\mathfrak{H} \cup \mathfrak{L} \cup \bullet(\mathfrak{H}, (h * l), (h' * l')))$ then it is obtained by reflexivity, symmetry or transitivity. The first case is absurd since $\mathfrak{H} \cup \mathfrak{L}$ is an equivalence relation on $\mathbb{H}^{\equiv}$. In the second case (z', z) is in $\mathfrak{H} \cup \mathfrak{L} \cup \bullet(\mathfrak{H}, (h^* l), (h'^* l'))$ or is obtained by transitivity. In the case a new relation is obtained by transitivity we get the result by transitivity of $=$ in definition of bisimulation $\rightarrow$.*

Lemma A.2. $\forall h_1, h_2, l.(h_1 \sim h_2, k_1 \equiv l \cdot h_1, k_2 \equiv l \cdot h_2 \Rightarrow k_1 \sim k_2)$

Proof. By structural induction on l:

– *case $l = \delta$:*
  $k_2 \equiv k_1 \equiv \delta$, thesis immediate by Theorem 2.4.
– *case $l = (a, \alpha)$:*
  $k_1 \equiv (a, \alpha) \cdot h_1, k_2 \equiv (a, \alpha) \cdot h_2$. It's easy to see that $(a, \alpha) \rightsquigarrow \nu$ with $\nu(b)(H) = 0$ for every H and $b \neq a$, moreover $\nu(a)(k) \neq 0$ iff $k \equiv \checkmark$.

$$k_1 \rightsquigarrow \odot_{h_1}\nu(a)(H) = \sum_{k\in H} \begin{cases} \nu(a)([l]^{\equiv}) & \text{if } \exists l \in \mathbb{H}.k \equiv l \cdot h_1 \\ \nu(a)(\checkmark) & \text{if } k \equiv h_1, k \not\equiv \delta \\ 0 & \text{otherwise} \end{cases} =$$

$$= \sum_{k\in H} \begin{cases} \alpha & \text{if } k \equiv h_1 \\ 0 & \text{otherwise} \end{cases}$$

and symmetrically on k_2. Thesis follow by the fact that every set $\sim$-closed that contain h_1 contains h_2 and so

$$\sum_{k\in H} \begin{cases} \alpha & \text{if } k \equiv h_1 \\ 0 & \text{otherwise} \end{cases} = \sum_{k\in H} \begin{cases} \alpha & \text{if } k \equiv h_2 \\ 0 & \text{otherwise} \end{cases}$$

- *case* $l = h^1 + h^2$*:*
  $k_1 \equiv (h^1 + h^2) \cdot h_1 \equiv h^1 \cdot h_1 + h^2 \cdot h_1, k_2 \equiv (h^1 + h^2) \cdot h_2 \equiv h^1 \cdot h_2 + h^2 \cdot h_2$*.* *By inductive hypothesis we get* $h^1 \cdot h_1 \sim h^1 \cdot h_2$ *and* $h^2 \cdot h_1 \sim h^2 \cdot h_2$*. Thesis follows immediately pointing out that* $+$ *semantics is measures sum.*
- *case* $l = h^1 \cdot h^2$*:*
  $k_1 \equiv h^1 \cdot (h^2 \cdot h_1), k_2 \equiv h^1 \cdot (h^2 \cdot h_2)$*. Thesis follows by inductive hypothesis applied twice to obtain* $(h^2 \cdot h_1), \sim (h^2 \cdot h_2)$ *first and then* $h^1 \cdot (h^2 \cdot h_1) \sim h^1 \cdot (h^2 \cdot h_2)$*.*
- *case* $l = h^1 * h^2$*:*
  Already proved by Lemma A.1

□

Proof of Theorem (2.4). By structural induction:

- $h_1 \sim h'_1$ implies $\psi[h_1] \sim \psi[h'_1]$: immediate result.
- $h_1 \sim h'_1, h_2 \sim h'_2$ implies $h_1 + h_2 \sim h'_1 + h'_2$:
  For arbitrary $H \in \mathbb{H}^{\equiv}$ $\sim$-closed assuming $h_1 \rightsquigarrow \mu_1, h_2 \rightsquigarrow \mu_2, h'_1 \rightsquigarrow \mu'_1, h'_2 \rightsquigarrow \mu'_2$ we get $(\mu_1 \oplus \mu_2)(a)(H) = \mu_1(a)(H) + \mu_2(a)(H)$. By inductive hypothesis $\mu_1(a)(H) = \mu'_1(a)(H)$ and $\mu_2(a)(H) = \mu'_2(a)(H)$ and then $\mu_1(a)(H) + \mu_2(a)(H) = \mu'_1(a)(H) + \mu'_2(a)(H) = (\mu'_1 \oplus \mu'_2)(a)(H)$.
- $h_1 \sim h'_1, h_2 \sim h'_2$ implies $h_1 \cdot h_2 \sim h'_1 \cdot h'_2$:

  Instead of considering a $\sim$-closed C set (reunion of more equivalence classes) we use H as if it were a single equivalence class (set of bisimilar state). The general result for $\sim$-closed sets will be obtained by σ-additivity and $C = \cup_{i \in I} H_i$ for generic H_i equivalence class. We assume $h_1 \rightsquigarrow \mu_1, h'_1 \rightsquigarrow \mu_2$:

$$[\odot_{h_2}\mu_1](a)(H) = \sum_{k \in H} \begin{cases} \mu_1(a)([l]^{\equiv}) & \text{if } \exists l \in \mathbb{H}.k \equiv l \cdot h_2 \\ \mu_1(a)(\checkmark) & \text{if } k \equiv h_2, k \not\equiv \delta \\ 0 & \text{otherwise} \end{cases} =$$

$$= \mu_1(a)(L) + \mu_1(a)(\checkmark)$$

$$[\odot_{h'_2}\mu_2](a)(H) = \sum_{k \in H} \begin{cases} \mu_2(a)([l']^{\equiv}) & \text{if } \exists l' \in \mathbb{H}.k \equiv l' \cdot h'_2 \\ \mu_2(a)(\checkmark) & \text{if } k \equiv h'_2, k \not\equiv \delta \\ 0 & \text{otherwise} \end{cases} =$$

$$= \mu_2(a)(L') + \mu_2(a)(\checkmark)$$

  with $L = \{l \mid \exists k \in H.k \equiv l \cdot h_2\}$ and $L' = \{l' \mid \exists k \in H.k \equiv l' \cdot h'_2\}$. Thesis follows pointing out that $L = L'$ and they are $\sim$-closed, see Lemma A.2.
- $h_1 \sim h'_1, h_2 \sim h'_2$ implies $h_1 * h_2 \sim h'_1 * h'_2$:
  Already proved by Lemma A.1 as every bisimulation $\mathfrak{S}$ for $h_1 * h_2$ and $h'_1 * h'_2$ the bisimilarity $\sim \supset \mathfrak{S}$.

□

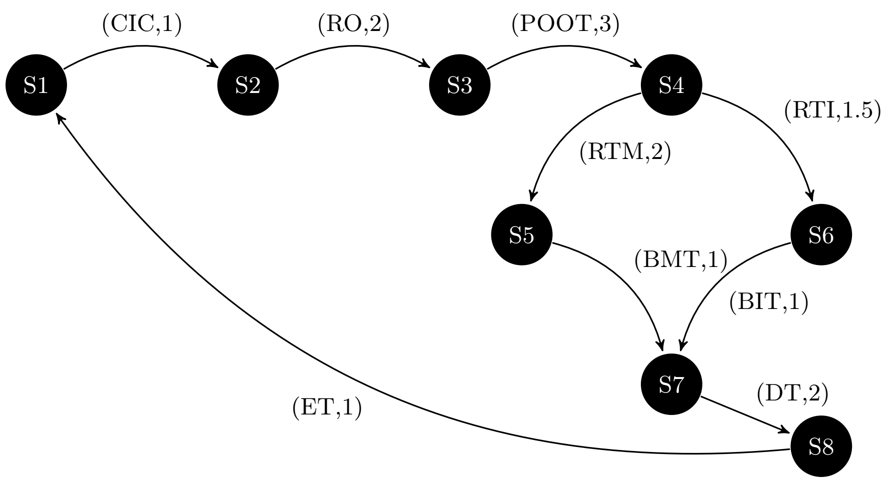

Figure B.1: The graph representation

B Working example datas

We pictorially represent the CTMC describing the behaviour of the stochastic history expression in the working example with the graph in Figure B.1. The graph is strongly connected because of the no-exit iteration present in the stochastic history expression $h_{Vestiti}$ of the working example. For brevity we use there the abbreviations in Table B.3 and Table 4.1.

The rate matrix, the Q matrix and the steady state, calculated solving $\pi Q = 0$, is:

$$R = \begin{pmatrix} 0&1&0&0&0&0&0&0\\ 0&0&2&0&0&0&0&0\\ 0&0&0&3&0&0&0&0\\ 0&0&0&0&2&1.5&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&2\\ 1&0&0&0&0&0&0&0 \end{pmatrix} Q = \begin{pmatrix} -1&1&0&0&0&0&0&0\\ 0&-2&2&0&0&0&0&0\\ 0&0&-3&3&0&0&0&0\\ 0&0&0&-3.5&2&1.5&0&0\\ 0&0&0&0&-1&0&1&0\\ 0&0&0&0&0&-1&1&0\\ 0&0&0&0&0&0&-2&2\\ 1&0&0&0&0&0&0&-1 \end{pmatrix} \pi = \begin{pmatrix} 0.216\\ 0.108\\ 0.072\\ 0.061\\ 0.123\\ 0.092\\ 0.108\\ 0.216 \end{pmatrix}$$

Table B.2: Rate matrix, Q matrix and steady state.

We show now the calculations involved in the policy checking. To do this we use the recursive characterization of Prb. We will present each Prb calculation

On the left column the name used to name states in B.1, on the right column the stochastic history expression the name stands for.

Abbreviation	Stochastic History Expression
S1	$\Big((\mathrm{CIC},1)\cdot(\mathrm{RO},2)\cdot(\mathrm{POT},3)\cdot \psi\Big[\big((\mathrm{RTM},2)\cdot(\mathrm{BMT},1)+(\mathrm{RTI},1.5)\cdot(\mathrm{BIT},1)\big)\cdot(\mathrm{DT},2)\cdot (\mathrm{ET},1)\Big]\Big)^{*}\,\delta$
S2	$(\mathrm{RO},2)\cdot(\mathrm{POT},3)\cdot\psi\Big[\big((\mathrm{RTM},2)\cdot(\mathrm{BMT},1)+(\mathrm{RTI},1.5)\cdot (\mathrm{BIT},1)\big)\cdot(\mathrm{DT},2)\cdot(\mathrm{ET},1)\Big]\cdot S1$
S3	$(\mathrm{POT},3)\cdot\psi\Big[\big((\mathrm{RTM},2)\cdot(\mathrm{BMT},1)+(\mathrm{RTI},1.5)\cdot(\mathrm{BIT},1)\big)\cdot (\mathrm{DT},2)\cdot(\mathrm{ET},1)\Big]\cdot S1$
S4	$(\psi\Big[\big((\mathrm{RTM},2)\cdot(\mathrm{BMT},1)+(\mathrm{RTI},1.5)\cdot(\mathrm{BIT},1)\big)\cdot(\mathrm{DT},2)\cdot (\mathrm{ET},1)\Big]\cdot S1$
S5	$(\mathrm{BMT},1)\cdot(\mathrm{DT},2)\cdot(\mathrm{ET},1)\cdot S1$
S6	$(\mathrm{BIT},1)\cdot(\mathrm{DT},2)\cdot(\mathrm{ET},1)\cdot S1$
S7	$(\mathrm{DT},2)\cdot(\mathrm{ET},1)\cdot S1$
S8	$(\mathrm{ET},1)\cdot S1$

Table B.3: States abbreviations

by postponing some recursive calculation that turns out necessary.

$$\begin{aligned}
&Prb(S4,\neg(\curvearrowright^{BMT}\vee\curvearrowright^{CIC})\,U^{[3,\infty]}\ \curvearrowright^{CIC})=\\
&=\int_0^\infty 1.5\cdot e^{-3.5\cdot x}\cdot Prb(S5,\neg(\curvearrowright^{BMT}\vee\curvearrowright^{CIC})\,U^{[3-x,\infty-x]}\ \curvearrowright^{CIC})\,dx+\\
&+\int_0^\infty 1.5\cdot e^{-3.5\cdot x}\cdot Prb(S6,\neg(\curvearrowright^{BMT}\vee\curvearrowright^{CIC})\,U^{[3-x,\infty-x]}\ \curvearrowright^{CIC})\,dx=\\
&=\int_0^\infty 1.5\cdot e^{-3.5\cdot x}\cdot Prb(S6,\neg(\curvearrowright^{BMT}\vee\curvearrowright^{CIC})\,U^{[3-x,\infty]}\ \curvearrowright^{CIC})\,dx=\\
&=\int_0^3 1.5\cdot e^{-3.5\cdot x}\cdot Prb(S6,\neg(\curvearrowright^{BMT}\vee\curvearrowright^{CIC})\,U^{[3-x,\infty]}\ \curvearrowright^{CIC})\,dx+\\
&+\int_3^\infty 1.5\cdot e^{-3.5\cdot x}\cdot Prb(S6,\neg(\curvearrowright^{BMT}\vee\curvearrowright^{CIC})\,U^{[3-x,\infty]}\ \curvearrowright^{CIC})\,dx=\\
&=\int_0^3 1.5\cdot e^{-3.5\cdot x}\cdot Prb(S6,\neg(\curvearrowright^{BMT}\vee\curvearrowright^{CIC})\,U^{[3-x,\infty]}\ \curvearrowright^{CIC})\,dx+\\
&+\int_3^\infty 1.5\cdot e^{-3.5\cdot x}\cdot 1\,dx=\\
&=0.127944+0.0000118013=\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad=0.127956
\end{aligned}$$

and

$$
\begin{aligned}
&Prb(S6, \neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[3-x,\infty]} \curvearrowright^{CIC}) = \\
&= \int_0^{\infty} 1 \cdot e^{-1\cdot z} \cdot Prb(S7, \neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[3-x-z,\infty-z]} \curvearrowright^{CIC})\, dz = \\
&= \int_0^{\infty} 1 \cdot e^{-1\cdot z} \cdot Prb(S7, \neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[3-x-z,\infty]} \curvearrowright^{CIC})\, dz = \\
&= \int_0^{3-x} 1 \cdot e^{-1\cdot z} \cdot Prb(S7, \neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[3-x-z,\infty]} \curvearrowright^{CIC})\, dz + \\
&+ \int_{3-x}^{\infty} 1 \cdot e^{-1\cdot z} \cdot Prb(S7, \neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[0,\infty]} \curvearrowright^{CIC})\, dz = \\
&= \int_0^{3-x} 1 \cdot e^{-1\cdot z} \cdot Prb(S7, \neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[3-x-z,\infty]} \curvearrowright^{CIC})\, dz + \int_{3-x}^{\infty} 1 \cdot e^{-1\cdot z}\, dz = \\
&\qquad = e^{-6+x}\left(e^{x} + e^{3}(5-2x)\right)
\end{aligned}
$$

and

$$
Prb(S5, \neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[3-x,\infty]} \curvearrowright^{CIC}) = 0
$$

and

$$
\begin{aligned}
&Prb(S7, \neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[3-x-z,\infty]} \curvearrowright^{CIC}) = \\
&= \int_0^{\infty} 2 \cdot e^{-2\cdot y} \cdot Prb(S8, \neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[3-x-z-y,\infty]} \curvearrowright^{CIC})\, dy = \\
&= \int_0^{3-x-z} 2 \cdot e^{-2\cdot y} \cdot Prb(S8, \neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[3-x-z-y,\infty]} \curvearrowright^{CIC})\, dy + \\
&+ \int_{3-x-z}^{\infty} 2 \cdot e^{-2\cdot y} \cdot Prb(S8, \neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[0,\infty]} \curvearrowright^{CIC})\, dy = \\
&= \int_0^{3-x-z} 2 \cdot e^{-2\cdot y} \cdot Prb(S8, \neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[3-x-z-y,\infty]} \curvearrowright^{CIC})\, dy + \int_{3-x-z}^{\infty} 2 \cdot e^{-2\cdot y}\, dy \\
&\qquad = 2e^{-6+x+z}\left(e^{3} - e^{x+z}\right) + e^{2(-3+x+z)}
\end{aligned}
$$

and

$$
\begin{aligned}
&Prb(S8, \neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[3-x-z-y,\infty]}\ \curvearrowright^{CIC}) = \\
&= \int_0^\infty 1 \cdot e^{-1\cdot h} \cdot Prb(S1, \neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[3-x-z-y-h,\infty]}\ \curvearrowright^{CIC})\, dh = \\
&= \int_0^{3-x-z-y} 1 \cdot e^{-1\cdot h} \cdot Prb(S1, \neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[3-x-z-y-h,\infty]}\ \curvearrowright^{CIC})\, dh + \\
&+ \int_{3-x-z-y}^\infty 1 \cdot e^{-1\cdot h} \cdot Prb(S1, \neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[0,\infty]}\ \curvearrowright^{CIC})\, dh = \\
&= \int_0^{3-x-z-y} 1 \cdot e^{-1\cdot h} \cdot Prb(S1, \neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[3-x-z-y-h,\infty]}\ \curvearrowright^{CIC})\, dh + \int_{3-x-z-y}^\infty 1 \cdot e^{-1\cdot h}\, dh = \\
&= \int_{3-x-z-y}^\infty 1 \cdot e^{-1\cdot h}\, dh = \\
&\qquad\qquad\qquad\qquad = e^{(-3+x+y+z)}
\end{aligned}
$$

and

$$
Prb(S1, \neg(\curvearrowright^{BMT} \vee \curvearrowright^{CIC})\, U^{[3-x-z-y-h,\infty]}\ \curvearrowright^{CIC}) = 0
$$