Dynamic Probabilistic Input Output Automata

Pierre Civit
Sorbonne Université, CNRS, LIP6, Paris, France

Maria Potop-Butucaru
Sorbonne Université, CNRS, LIP6, Paris, France

Abstract

We present *probabilistic dynamic I/O automata*, a framework to model dynamic probabilistic systems. Our work extends *dynamic I/O Automata* formalism of Attie & Lynch [2] to the probabilistic setting. The original dynamic I/O Automata formalism included operators for parallel composition, action hiding, action renaming, automaton creation, and behavioral sub-typing by means of trace inclusion. They can model mobility by using signature modification. They are also hierarchical: a dynamically changing system of interacting automata is itself modeled as a single automaton. Our work extends all these features to the probabilistic setting. Furthermore, we prove necessary and sufficient conditions to obtain the monotonicity of automata creation/destruction with implementation preorder. Our construction uses a novel proof technique based on homomorphism that can be of independent interest. Our work lays down the foundations for extending *composable secure-emulation* of Canetti et al. [5] to dynamic settings, an important tool towards the formal verification of protocols combining probabilistic distributed systems and cryptography in dynamic settings (e.g. blockchains, secure distributed computation, cybersecure distributed protocols, etc).

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases Automata, Distributed Computing, Formal Verification, Dynamic systems

Digital Object Identifier 10.4230/LIPIcs.DISC.2022.15

Related Version *Full Version*: `https://eprint.iacr.org/2021/798`

1 Introduction

Distributed computing area faces today important challenges coming from modern applications such as peer-to-peer networks, cooperative robotics, dynamic sensor networks, adhoc networks and more recently, cryptocurrencies and blockchains which have a tremendous impact in our society. These newly emerging fields of distributed systems are characterized by an extreme dynamism in terms of structure, content and load. Moreover, they have to offer strong guarantees over large scale networks which is usually impossible in deterministic settings. Therefore, most of these systems use probabilistic algorithms and randomized techniques in order to offer scalability features. However, the vulnerabilities of these systems may be exploited with the aim to provoke an unforeseen execution that diverges from the understanding or intuition of the developers. Therefore, formal validation and verification of these systems has to be realized before their industrial deployment.

It is difficult to attribute the pioneering of formalization of concurrent systems to some particular authors [16, 10, 1, 15, 11, 13, 9]. Lynch and Tuttle [12] proposed the formalism of *Input/Output Automata* to model deterministic asynchronous distributed systems. Relationship between process algebra and I/O automata are discussed in [20, 14]. Later, this formalism is extended by Segala in [19] with Markov decision processes [17]. In order to model randomized distributed systems Segala proposes *Probabilistic Input/Output Automata*. In this model each process in the system is an automaton with probabilistic transitions. The probabilistic protocol is the parallel composition of the automata modeling each participant.

The modelisation of dynamic behavior in distributed systems has been addressed by Attie & Lynch in [2] where they propose *Dynamic Input Output Automata* formalism. This formalism extends the *Input/Output Automata* with the ability to change their signature dynamically (i.e. the set of actions in which the automaton can participate) and to create other I/O automata or destroy existing I/O automata. The formalism introduced in [2] does not cover the case of probabilistic distributed systems and therefore cannot be used in the verification of recent blockchains such as Algorand [6].

In order to respond to the need of formalisation in secure distributed systems, Canetti & al. proposed in [3] *task-structured probabilistic Input/Output automata* (TPIOA) specifically designed for the analysis of cryptographic protocols. Task-structured probabilistic Input/Output automata are Probabilistic Input/Output automata extended with tasks that are equivalence classes on the set of actions. The task-structure allows a generalisation of "off-line scheduling" where the non-determinism of the system is resolved in advance by a *task-scheduler*, i.e. a sequence of tasks chosen in advance that trigger the actions among the enabled ones. They define the parallel composition for this type of automata, as well as the notion of implementation for TPIOA. Informally, the implementation of a Task-structured probabilistic Input/Output automata should look "similar" to the specification whatever will be the external environment of execution. Furthermore, they provide compositional results for the implementation relation. Even thought the formalism proposed in [5] (built on top of the one of [3]) has been already used in the formal proof of various cryptographic protocols [4, 21], this formalism does not capture the dynamicity of probabilistic dynamic systems such as peer-to-peer networks or blockchains systems where the set of participants dynamically changes or where subchains can be created or destroyed at run time [18].

Our contribution. In order to cope with dynamicity and probabilistic nature of modern distributed systems we propose an extension of the two formalisms introduced in [2] and [3]. Our extension uses a refined definition of probabilistic configuration automata in order to cope with dynamic actions. The main result of our formalism is as follows: the implementation of probabilistic configuration automata is monotonic to automata creation and destruction. That is, if systems $X_{\mathcal{A}}$ and $X_{\mathcal{B}}$ differ only in that $X_{\mathcal{A}}$ dynamically creates and destroys automaton $\mathcal{A}$ instead of creating and destroying automaton $\mathcal{B}$ as $X_{\mathcal{B}}$ does, and if $\mathcal{A}$ implements $\mathcal{B}$ (in the sense they cannot be distinguished by any external observer), then $X_{\mathcal{A}}$ implements $X_{\mathcal{B}}$. This result enables a design and refinement methodology based solely on the notion of externally visible behavior and permits the refinement of components and subsystems in isolation from the rest of the system. In our construction, we exhibit the need of considering only *creation-oblivious* schedulers in the implementation relation, i.e. a scheduler that, upon the (dynamic) creation of a sub-automaton $\mathcal{A}$, does not take into account the previous internal actions of $\mathcal{A}$ to output (randomly) a transition. Surprisingly, the task-schedulers introduced by Canetti & al. [3] are not creation-oblivious. Interestingly, an important contribution of the paper of independent interest is the proof technique we used in order to obtain our results. Differently from [2] and [3] which build their constructions mainly on induction techniques, we developed an elegant homomorphism based technique which aim to render the proofs modular. This proof technique can be easily adapted in order to further extend our framework with cryptography and time.

It should be noted that our work is an intermediate step before extending composable secure-emulation [5, 8] to dynamic settings. This extension is necessary for formal verification of secure dynamic distributed systems (e.g. blockchain systems).

Paper organization. The paper is organized as follows. Section 2 is dedicated to a brief introduction of the notion of probabilistic measure and recalls notations used in defining Signature I/O automata of [2]. Section 3 builds on the frameworks proposed in [2] and [3] in order to lay down the preliminaries of our formalism. More specifically, we introduce the definitions of probabilistic signed I/O automata and define their composition and implementation. In Section 4 we extend the definition of configuration automata proposed in [2] to probabilistic configuration automata then we define the composition of probabilistic configuration automata. Section 5 contains definitions related to the behavioural semantic of automata, e.g. executions, traces, etc. Section 6 introduces implementation relationship, which allows to formalise the idea that a concrete system is meeting the specification of an abstract object. The key result of our formalisation, the monotonicity of PSIOA implementations with respect to creation and destruction, is presented in Section 7 and demonstrated in the extended version. For a big picture, we recommend the reading of the warm up section of the extended version [7].

2 Preliminaries on probability and measure

We assume our reader is comfortable with basic notions of probability theory, such as σ-algebra and (discrete) probability measures. A measurable space is denoted by $(S, \mathcal{F}_S)$, where S is a set and $\mathcal{F}_S$ is a σ-algebra over S that is, $\mathcal{F}_S \subseteq \mathcal{P}(S)$, is closed under countable union and complementation and its members are called measurable sets ($\mathcal{P}(S)$ denotes the power set of S). The union of a collection $\{S_i\}_{i \in I}$ of pairwise disjoint sets indexed by a set I is written as $\biguplus_{i\in I} S_i$. A measure over $(S, \mathcal{F}_S)$ is a function $\eta : \mathcal{F}_s \rightarrow \mathbb{R}^{\geq 0}$, such that $\eta(\emptyset) = 0$ and for every countable collection of disjoint sets $\{S_i\}_{i\in I}$ in $\mathcal{F}_S$, $\eta(\biguplus_{i\in I} S_i) = \Sigma_{i \in I} \eta(S_i)$. A probability measure (resp. sub-probability measure) over $(S, \mathcal{F}_S)$ is a measure η such that $\eta(S) = 1$ (resp. $\eta(S) \leq 1$). A measure space is denoted by $(S, \mathcal{F}_S, \eta)$ where η is a measure on $(S, \mathcal{F}_S)$.

The product measure space $(S_1, \mathcal{F}_{s_1}, \eta_1) \otimes (S_2, \mathcal{F}_{s_2}, \eta_2)$ is the measure space $(S_1 \times S_2, \mathcal{F}_{s_1} \otimes \mathcal{F}_{s_2}, \eta_1 \otimes \eta_2)$, where $\mathcal{F}_{s_1} \otimes \mathcal{F}_{s_2}$ is the smallest σ-algebra generated by sets of the form $\{A \times B | A \in \mathcal{F}_{s_1}, B \in \mathcal{F}_{s_2}\}$ and $\eta_1 \otimes \eta_2$ is the unique measure s.t. for every $C_1 \in \mathcal{F}_{s_1}, C_2 \in \mathcal{F}_{s_2}$, $\eta_1 \otimes \eta_2(C_1 \times C_2) = \eta_1(C_1) \cdot \eta_2(C_2)$. If S is countable, we note $\mathcal{P}(S) = 2^S$. If S_1 and S_2 are countable, we have $2^{S_1} \otimes 2^{S_2} = 2^{S_1 \times S_2}$.

A discrete probability measure on a set S is a probability measure η on $(S, 2^S)$, such that, for each $C \subset S, \eta(C) = \sum_{c \in C} \eta(\{c\})$. We define $Disc(S)$ and $SubDisc(S)$ to be respectively, the set of discrete probability and sub-probability measures on S. In the sequel, we often omit the set notation when we denote the measure of a singleton set. For a discrete probability measure η on a set S, $supp(\eta)$ denotes the support of η, that is, the set of elements $s \in S$ such that $\eta(s) \neq 0$. Given set S and a subset $C \subset S$, the Dirac measure δ_C is the discrete probability measure on S that assigns probability 1 to C. For each element $s \in S$, we note δ_s for $\delta_{\{s\}}$.

If $\{m_i\}_{i\in I}$ is a countable family of measures on $(S, \mathcal{F}_S)$, and $\{p_i\}_{i \in I}$ is a family of non-negative values, then the expression $\sum_{i \in I} p_i m_i$ denotes a measure m on $(S, \mathcal{F}_S)$ such that, for each $C \in \mathcal{F}_S, m(C) = \sum_{i\in I} m_i f_i(C)$. A function $f : X \rightarrow Y$ is said to be measurable from $(X, \mathcal{F}_X) \rightarrow (Y, \mathcal{F}_Y)$ if the inverse image of each element of $\mathcal{F}_Y$ is an element of $\mathcal{F}_X$, that is, for each $C \in \mathcal{F}_Y$, $f^{-1}(C) \in \mathcal{F}_X$. In such a case, given a measure η on $(X, \mathcal{F}_X)$, the function $f(\eta)$ defined on $\mathcal{F}_Y$ by $f(\eta)(C) = \eta(f^{-1}(C))$ for each $C \in Y$ is a measure on $(Y, \mathcal{F}_Y)$ and is called the image measure of η under f.

Let $(Q_1, 2^{Q_1})$ and $(Q_2, 2^{Q_2})$ be two measurable sets. Let $(\eta_2, \eta_2) \in Disc(Q_1) \times Disc(Q_2)$. Let $f : Q_1 \rightarrow Q_2$. We note $\eta_1 \overset{f}{\leftrightarrow} \eta_2$ if the following is verified: (1) the restriction $\tilde{f}$ of f to $supp(\eta_1)$ is a bijection from $supp(\eta_1)$ to $supp(\eta_2)$ and (2) $\forall q \in supp(\eta)$, $\eta(q_1) = \eta_2(f(q_1))$.

3 Probabilistic Signature Input/Output Automata (PSIOA)

This section aims to introduce the first brick of our formalism: the probabilistic signature input/output automata (PSIOA). A PSIOA $\mathcal{A}$ is an automaton that can move from one *state* to another through *actions*. At each state q some actions can be triggered in its signature $sig(\mathcal{A})(q)$. Such an action leads to a new state with a certain probability. The fact that the signature can evolve throughout an execution is particularly convenient to model dynamicity.

3.1 PSIOA

We combine the SIOA of [2] with the PIOA of [19]. We use the signature approach from [2]. We assume the existence of a countable set $Autids$ of unique probabilistic signature input/output automata (PSIOA) identifiers, an underlying universal set $Auts$ of PSIOA, and a mapping $aut : Autids \rightarrow Auts$. $aut(\mathcal{A})$ is the PSIOA with identifier $\mathcal{A}$. We use "the automaton $\mathcal{A}$" to mean "the PSIOA with identifier $\mathcal{A}$". We use the letters $\mathcal{A}, \mathcal{B}$, possibly subscripted or primed, for PSIOA identifiers.

▶ **Definition 1** (PSIOA). *A PSIOA $\mathcal{A} = (Q_{\mathcal{A}}, \bar{q}_{\mathcal{A}}, sig(\mathcal{A}), D_{\mathcal{A}})$, where:*

- *$Q_{\mathcal{A}}$ is a countable set of* states*, $(Q_{\mathcal{A}}, 2^{Q_{\mathcal{A}}})$ is the* state space*,*
- *$\bar{q}_{\mathcal{A}}$ is the unique* start state*.*
- *$sig(\mathcal{A}) : q \in Q_{\mathcal{A}} \mapsto sig(\mathcal{A})(q) = (in(\mathcal{A})(q), out(\mathcal{A})(q), int(\mathcal{A})(q))$ is the signature function that maps each state to a triplet of mutually disjoint countable set of* actions*, respectively called* input*,* output *and* internal *actions.*
- *$D_{\mathcal{A}} \subset Q_{\mathcal{A}} \times acts(\mathcal{A}) \times Disc(Q_{\mathcal{A}})$ is the set of* probabilistic discrete transitions *where $\forall (q, a, \eta) \in D_{\mathcal{A}} : a \in \widehat{sig}(\mathcal{A})(q)$. If (q, a, η) is an element of $D_{\mathcal{A}}$, we write $q \xrightarrow{a} \eta$ and action a is said to be* enabled *at q. We note $enabled(\mathcal{A}) : q \in Q_{\mathcal{A}} \mapsto enabled(\mathcal{A})(q)$ where $enabled(\mathcal{A})(q)$ denotes the set of enabled actions at state q. We also note $steps(\mathcal{A}) \triangleq \{(q, a, q') \in Q_{\mathcal{A}} \times acts(\mathcal{A}) \times Q_{\mathcal{A}} | \exists (q, a, \eta) \in D_{\mathcal{A}}, q' \in supp(\eta)\}$.*

  In addition $\mathcal{A}$ must satisfy the following conditions:
- $\mathbf{E_1}$ (input enabling) $\forall q \in Q_{\mathcal{A}}$, $in(\mathcal{A})(q) \subseteq enabled(\mathcal{A})(q)$.[1]
- $\mathbf{T_1}$ (Transition determinism): For every $q \in Q_{\mathcal{A}}$ and $a \in \widehat{sig}(\mathcal{A})(q)$ there is at most one $\eta_{(\mathcal{A},q,a)} \in Disc(Q_{\mathcal{A}})$, such that $(q, a, \eta_{(\mathcal{A},q,a)}) \in D_{\mathcal{A}}$.

We define $ext(\mathcal{A})(q)$, the external signature of $\mathcal{A}$ in state q, to be $ext(\mathcal{A})(q) = (in(\mathcal{A})(q), out(\mathcal{A})(q))$. We define $loc(\mathcal{A})(q)$, the local signature of $\mathcal{A}$ in state q, to be $loc(\mathcal{A})(q) = (out(\mathcal{A})(q), int(\mathcal{A})(q))$. For any signature component, generally, the $\widehat{\cdot}$ operator yields the union of sets of actions within the signature, e.g., $\widehat{sig}(\mathcal{A}) : q \in Q \mapsto \widehat{sig}(\mathcal{A})(q) = in(\mathcal{A})(q) \cup out(\mathcal{A})(q) \cup int(\mathcal{A})(q)$. Also we define $acts(\mathcal{A}) = \bigcup_{q \in Q} \widehat{sig}(\mathcal{A})(q)$, that is $acts(\mathcal{A})$ is the "universal" set of all actions that A could possibly trigger, in any state.

Later, we will define *execution fragments* as alternating sequences of states and actions with classic and natural consistency rules. But a subtlety will appear with the composability of set of automata at reachable states. Hence, we will define *execution fragments* after "local composability" and "probabilistic configuration automata".

[1] Since the signature is dynamic, we can require $\widehat{sig}(\mathcal{A}) = enabled(\mathcal{A})$

3.2 Local composition

The main aim of a formalism of concurrent systems is to compose several automata $\mathbf{A} = \{\mathcal{A}_1, ..., \mathcal{A}_n\}$ to capture the idea of an interaction between them and provide guarantees by composing the guarantees of the different elements of the system. Some syntactical rules have to be satisfied before defining the composition operation.

▶ **Definition 2** (Compatible signatures). *Let $S = \{sig_i\}_{i \in \mathcal{I}}$ be a set of signatures. Then S is compatible iff, $\forall i, j \in \mathcal{I}$, $i \neq j$, where $sig_i = (in_i, out_i, int_i)$, $sig_j = (in_j, out_j, int_j)$, we have: 1. $(in_i \cup out_i \cup int_i) \cap int_j = \emptyset$, and 2. $out_i \cap out_j = \emptyset$.*

▶ **Definition 3** (Composition of Signatures). *Let $\Sigma = (in, out, int)$ and $\Sigma' = (in', out', int')$ be compatible signatures. Then we define their composition $\Sigma \times \Sigma = (in \cup in' - (out \cup out'), out \cup out', int \cup int')$[2].*

Signature composition is clearly commutative and associative. Now we can define the compatibility of several automata at a state with the compatibility of their attached signatures. First we define compatibility at a state, and discrete transition for a set of automata for a particular compatible state.

▶ **Definition 4** (Compatibility at a state). *Let $\mathbf{A} = \{\mathcal{A}_1, ..., \mathcal{A}_n\}$ be a set of PSIOA. A* state *of $\mathbf{A}$ is an element $q = (q_1, ..., q_n) \in Q_{\mathbf{A}} \triangleq Q_{\mathcal{A}_1} \times ... \times Q_{\mathcal{A}_n}$. We note $q \restriction \mathcal{A}_i \triangleq q_i$. We say $\mathcal{A}_1, ..., \mathcal{A}_n$ are (or $\mathbf{A}$ is)* compatible *at state q if $\{sig(\mathcal{A}_1)(q_1), ..., sig(\mathcal{A}_n)(q_n)\}$ is a set of compatible signatures. In this case we note $sig(\mathbf{A})(q) \triangleq sig(\mathcal{A}_1)(q_1) \times ... \times sig(\mathcal{A}_n)(q_n)$ as per definition 3 and we note $\eta_{(\mathbf{A},q,a)} \in Disc(Q_{\mathbf{A}})$, s.t. $\forall a \in \widehat{sig}(\mathbf{A})(q)$, $\eta_{(\mathbf{A},q,a)} = \eta_1 \otimes ... \otimes \eta_n$ where $\forall j \in [1, n]$, $\eta_j = \eta_{(\mathcal{A}_j, q_j, a)}$ if $a \in sig(\mathcal{A}_j)(q_j)$ and $\eta_j = \delta_{q_j}$ otherwise. Moreover, we note $steps(\mathbf{A}) = \{(q, a, q') | q, q' \in Q_{\mathbf{A}}, a \in sig(\mathbf{A})(q), q' \in supp(\eta_{(\mathbf{A},q,a)})\}$. Finally, we note $\bar{q}_{\mathbf{A}} = (\bar{q}_{\mathcal{A}_1}, ..., \bar{q}_{\mathcal{A}_n})$.*

Let us note that an action a shared by two automata becomes an output action and not an internal action after composition. First, it permits the possibility of further communication using a. Second, it allows associativity. If this property is counter-intuitive, it is always possible to use the classic hiding operator that "hides" the output actions transforming them into internal actions.

▶ **Definition 5** (Hiding operator). *Let $sig = (in, out, int)$ be a signature and H a set of actions. We note $hide(sig, H) \triangleq (in, out \setminus H, int \cup (out \cap H))$.*

Let $\mathcal{A} = (Q_{\mathcal{A}}, \bar{q}_{\mathcal{A}}, sig(\mathcal{A}), D_{\mathcal{A}})$ be a PSIOA. Let $h : q \in Q_{\mathcal{A}} \mapsto h(q) \subseteq out(\mathcal{A})(q)$. We note $hide(\mathcal{A}, h) \triangleq (Q_{\mathcal{A}}, \bar{q}_{\mathcal{A}}, sig'(\mathcal{A}), D_{\mathcal{A}})$, where $sig'(\mathcal{A}) : q \in Q_{\mathcal{A}} \mapsto hide(sig(\mathcal{A})(q), h(q))$. Clearly, $hide(\mathcal{A}, h)$ is a PSIOA.

4 Probabilistic Configuration Automata

We combine the notion of configuration of [2] with the probabilistic setting of [19]. A configuration is a set of automata attached with their current states. This will be a very useful tool to define dynamicity by mapping the state of an automaton of a certain "layer" to a configuration of automata of lower layer, where the set of automata in the configuration can dynamically change from on state of the automaton of the upper level to another one.

[2] not to be confused with Cartesian product. We keep this notation to stay as close as possible to the literature.

4.1 Configuration

▶ **Definition 6** (Configuration). *A configuration is a pair* $(\mathbf{A}, \mathbf{S})$ *where*

- $\mathbf{A} = \{\mathcal{A}_1, ..., \mathcal{A}_n\}$ *is a finite set of PSIOA identifiers and*
- $\mathbf{S}$ *maps each* $\mathcal{A}_k \in \mathbf{A}$ *to a state of* $\mathcal{A}_k$*.*

In distributed computing, configuration usually refers to the union of states of ***all*** *the automata of the "system". Here, there is a subtlety, since it captures a set of some automata (*$\mathbf{A}$*) in their current state (*$\mathbf{S}$*), but the set of automata of the systems will not be fixed in the time.*

Since, (1) $\{\mathbf{A} \in \mathcal{P}(Autids) | \mathbf{A} \text{ is finite}\}$ *is countable, (2)* $\forall \mathcal{A} \in Autids, Q_{\mathcal{A}}$ *is countable by definition 1 of PSIOA and (3) the cartesian product of countable sets is a countable set, we can deduce that the set* Q_{conf} *of configurations is countable.*

▶ **Definition 7** (Compatible configuration). *A configuration* $(\mathbf{A}, \mathbf{S})$*, with* $\mathbf{A} = \{\mathcal{A}_1, ..., \mathcal{A}_n\}$*, is compatible iff the set* $\mathbf{A}$ *is compatible at state* $(\mathbf{S}(\mathcal{A}_1), ..., \mathbf{S}(\mathcal{A}_n))$ *as per definition 4.*

▶ **Definition 8** (Intrinsic attributes of a configuration). *Let* $C = (\mathbf{A}, \mathbf{S})$ *be a compatible configuration. Then we define*

- $auts(C) = \mathbf{A}$ *represents the automata of the configuration,*
- $map(C) = \mathbf{S}$ *maps each automaton of the configuration with its current state,*
- $TS(C) = (\mathbf{S}(\mathcal{A}_1), ..., \mathbf{S}(\mathcal{A}_n))$ *yields the tuple of states of the automata of the configuration.*
- $sig(C) = (in(C), out(C), int(C)) = sig(auts(C), TS(C))$ *in the sense of definition 4, is called the intrinsic signature of the configuration*

Here we define a reduced configuration as a configuration deprived of the automata that are in the very particular state where their current signatures are the empty set. This mechanism will be used later to capture the idea of destruction of an automaton.

▶ **Definition 9** (Reduced configuration). $reduce(C) = (\mathbf{A}', \mathbf{S}')$*, where* $\mathbf{A}' = \{\mathcal{A} | \mathcal{A} \in \mathbf{A} \text{ and } sig(\mathcal{A})(\mathbf{S}(\mathcal{A})) \neq \emptyset\}$ *and* $\mathbf{S}'$ *is the restriction of* $\mathbf{S}$ *to* $\mathbf{A}'$*, noted* $\mathbf{S} \upharpoonright \mathbf{A}'$ *in the remaining.*

A configuration C *is a reduced configuration iff* $C = reduce(C)$*.*

We will define some probabilistic transition from configurations to others where some automata can be destroyed or created. To define it properly, we start by defining "preserving transition" where no automaton is neither created nor destroyed and then we define above this definition the notion of configuration transition.

▶ **Definition 10** (From preserving distribution to intrinsic transition).

- *(preserving distribution) Let* $\eta_p \in Disc(Q_{conf})$*. We say* η_p *is a* preserving distribution *if it exists a finite set of automata* $\mathbf{A}$*, called* family support of η_p*, s.t.* $\forall (\mathbf{A}', \mathbf{S}') \in supp(\eta_p), \mathbf{A} = \mathbf{A}'$*.*
- *(preserving configuration transition* $C \overset{a}{\rightharpoonup} \eta_p$*) Let* $C = (\mathbf{A}, \mathbf{S})$ *be a compatible configuration,* $a \in \widehat{sig}(C)$*. Let* η_p *be the unique preserving distribution of* $Disc(Q_{conf})$ *such that (1) the family support of* η_p *is* $\mathbf{A}$ *and (2)* $\eta_p \overset{TS}{\leftrightarrow} \eta_{(\mathbf{A}, TS(C), a)}$*. We say that* (C, a, η_p) *is a* preserving configuration transition*, noted* $C \overset{a}{\rightharpoonup} \eta_p$*.*
- *(*$\eta_p \uparrow \varphi$*) Let* $\eta_p \in Disc(Q_{conf})$ *be a preserving distribution with* $\mathbf{A}$ *as family support. Let* φ *be a finite set of of PSIOA identifiers with* $\mathbf{A} \cap \varphi = \emptyset$*. Let* $C_\varphi = (\varphi, S_\varphi) \in Q_{conf}$ *with* $\forall \mathcal{A}_j \in \varphi, S_\varphi(\mathcal{A}_j) = \bar{q}_{\mathcal{A}_j}$*. We note* $\eta_p \uparrow \varphi$ *the unique element of* $Disc(Q_{conf})$ *verifying* $\eta_p \overset{u}{\leftrightarrow} (\eta_p \uparrow \varphi)$ *with* $u : C \in supp(\eta_p) \mapsto (C \cup C_\varphi)$*.*

- *(distribution reduction) Let $\eta \in Disc(Q_{conf})$. We note $reduce(\eta)$ the element of $Disc(Q_{conf})$ verifying $\forall c \in Q_{conf}$, $(reduce(\eta))(c) = \Sigma_{(c' \in supp(\eta), c = reduce(c'))}\eta(c')$*
- *(intrinsic transition $C \overset{a}{\Longrightarrow}_{\varphi} \eta$) Let $C = (\mathbf{A}, \mathbf{S})$ be a compatible configuration, let $a \in \widehat{sig}(C)$, let φ be a finite set of of PSIOA identifiers with $\mathbf{A} \cap \varphi = \emptyset$. We note $C \overset{a}{\Longrightarrow}_{\varphi} \eta$, if $\eta = reduce(\eta_p \uparrow \varphi)$ with $C \overset{a}{\rightharpoonup} \eta_p$. In this case, we say that η is* generated by *η_p* and *φ.*

Preserving configuration transition (C, a, η_p) is the intuitive transition for configurations, corresponding to the transition $(TS(C), a, \eta_{(auts(C), TS(C), a)})$. The operator $\uparrow \varphi$ describes the deterministic creation of automata in φ, who will be appear at their respective start states. The *reduce* operator enables to remove "destroyed" automata from the possibly returned configurations (see Figure 1).

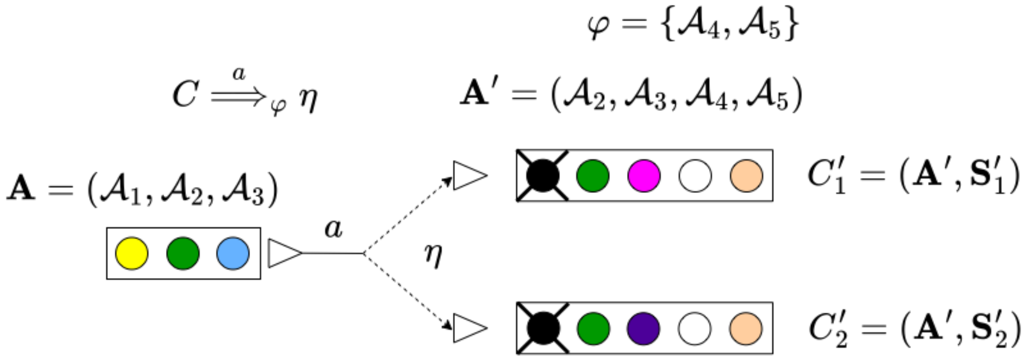

$\varphi = \{\mathcal{A}_4, \mathcal{A}_5\}$
$C \overset{a}{\Longrightarrow}_{\varphi} \eta$
$\mathbf{A}' = (\mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5)$
$\mathbf{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$
a
η
$C'_1 = (\mathbf{A}', \mathbf{S}'_1)$
$C'_2 = (\mathbf{A}', \mathbf{S}'_2)$

Figure 1 An intrinsic transition where $\mathcal{A}_1$ is destroyed deterministically and automata in $\varphi = \{\mathcal{A}_4, \mathcal{A}_5\}$ are created deterministically. First, we have the preserving disribution η_p s.t. $C \overset{a}{\rightharpoonup} \eta_p$ with $\eta_p \overset{TS}{\leftrightarrow} \eta_{(\mathbf{A}, TS(C), a)}$. Second, we take into account the created automata in φ, captured by the distribution $\eta_p \uparrow \varphi$. Third, we remove the automata in a particular state with associated empty signature ($\mathcal{A}_1$ in our example). This is captured by distribution *reduce*$(\eta_p \uparrow \varphi)$.

4.2 Probabilistic configuration automata (PCA)

Now we are ready to define our probabilistic configuration automata (see figure 2). Such an automaton define a strong link with a dynamic configuration.

▶ **Definition 11** (Probabilistic Configuration Automaton). *A probabilistic configuration automaton (PCA) X consists of the following components:*

1. *A probabilistic signature I/O automaton $psioa(X)$. For brevity, we define $Q_X = Q_{psioa(X)}, \bar{q}_X = \bar{q}_{psioa(X)}$, $sig(X) = sig(psioa(X)), steps(X) = steps(psioa(X))$, and likewise for all other (sub)components and attributes of $psioa(X)$.*
2. *A configuration mapping $config(X)$ with domain Q_X and such that, for all $q \in Q_X$, $config(X)(q)$ is a reduced compatible configuration.*
3. *For each $q \in Q_X$, a mapping $created(X)(q)$ with domain $sig(X)(q)$ and such that $\forall a \in sig(X)(q)$, $created(X)(q)(a) \subseteq Autids$ with $created(X)(q)(a)$ finite.*
4. *A hidden-actions mapping hidden-actions(X) with domain Q_X and such that hidden-actions$(X)(q) \subseteq out(config(X)(q))$.*

and satisfies the following constraints, for every $q \in Q_X$, $C = config(X)(q)$, $H =$ hidden-actions(q).

1. *(start states preservation) If* $config(X)(\bar{q}_X) = (\mathbf{A}, \mathbf{S})$*, then* $\forall \mathcal{A}_i \in \mathbf{A}, \mathbf{S}(\mathcal{A}_i) = \bar{q}_{\mathcal{A}_i}$*.*
2. *(top/down transition preservation) If* $(q, a, \eta_{(X,q,a)}) \in D_X$*, then* $\exists \eta' \in Disc(Q_{conf})$ *s.t.* $\eta_{(X,q,a)} \overset{c}{\leftrightarrow} \eta'$ *with* $C \overset{a}{\Longrightarrow}_{\varphi} \eta'$*, where* $\varphi = created(X)(q)(a)$ *and* $c = config(X)$*.*
3. *(bottom/up transition preservation) If* $q \in Q_X$ *and* $C \overset{a}{\Longrightarrow}_{\varphi} \eta'$ *for some action* a*,* $\varphi = created(X)(q)(a)$*, and reduced compatible probabilistic measure* $\eta' \in Disc(Q_{conf})$*, then* $(q, a, \eta_{(X,q,a)}) \in D_X$*, and* $\eta_{(X,q,a)} \overset{c}{\leftrightarrow} \eta'$ *where* $c = config(X)$*.*
4. *(signature preservation modulo hiding)* $\forall q \in Q_X$ *,* $sig(X)(q) = hide(sig(C), H)$*.*

This definition, proposed in a deterministic fashion in [2], captures dynamicity of the system. Each state is linked with a configuration. The set of automata of the configuration can change during an execution. A sub-automaton $\mathcal{A}$ is created from state q by the action a if $\mathcal{A} \in created(X)(q)(a)$. A sub-automaton $\mathcal{A}$ is destroyed if the non-reduced attached configuration distribution leads to a configuration where $\mathcal{A}$ is in a state $q^{\phi}_{\mathcal{A}}$ s.t. $\widehat{sig}(\mathcal{A})(q^{\phi}_{\mathcal{A}}) = \emptyset$. Then the corresponding reduced configuration will not hold $\mathcal{A}$. The last constraint states that the signature of a state q of X must be the same as the signature of its corresponding configuration $config(X)(q)$, except for the possible effects of hiding operators, so that some outputs of $config(X)(q)$ may be internal actions of X in state q.

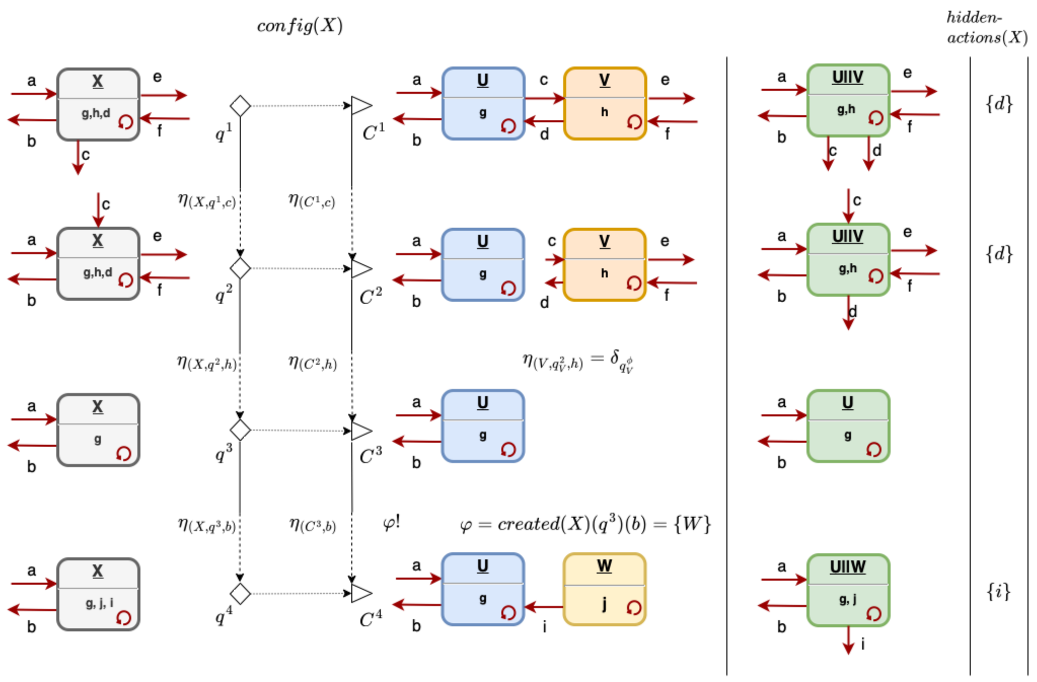


Figure 2 A PCA life cycle. V is destroyed at step (q^2, h, q^3), while W is created at step (q^3, b, q^4).

As for PSIOA, we can define hiding operator applied to PCA.

▶ **Definition 12** (Hiding on PCA). *Let* X *be a PCA. Let* $h : q \in Q_X \mapsto h(q) \subseteq out(X)(q)$*. We note* $hide(X, h)$ *the PCA* X' *that differs from* X *only on*

- $psioa(X') = hide(psioa(X), h)$
- $sig(X') = hide(sig(X), h)$ *and*
- $\forall q \in Q_X = Q_{X'}$*,* $hidden\text{-}actions(X')(q) = hidden\text{-}actions(X)(q) \cup h(q)$*.*

The notion of local compatibility can be naturally extended to set of PCA.

▶ **Definition 13** (PCA compatible at a state)**.** *Let $\mathbf{X} = \{X_1, ..., X_n\}$ be a set of PCA. Let $q = (q_1, ..., q_n) \in Q_{X_1} \times ... \times Q_{X_n}$. Let us note $C_i = (\mathbf{A}_i, \mathbf{S}_i) = config(X_i)(q_i)$, $\forall i \in [1, n]$. The PCA in $\mathbf{X}$ are compatible at state q iff*[3]*:*

1. *PSIOA compatibility: $psioa(X_1), ..., psioa(X_n)$ are compatible at $q_{\mathbf{X}}$.*
2. *Sub-automaton exclusivity: $\forall i, j \in [1:n], i \neq j : \mathbf{A}_i \cap \mathbf{A}_j = \emptyset$.*
3. *Creation exclusivity: $\forall i, j \in [1:n], i \neq j, \forall a \in \widehat{sig}(X_i)(q_i) \cap \widehat{sig}(X_j)(q_j)$: $created(X_i)(q_i)(a) \cap created(X_j)(q_j)(a) = \emptyset$.*

If $\mathbf{X}$ is compatible at state q, for every action $a \in \widehat{sig}(psioa(\mathbf{X}))(q)$, we note $\eta_{(\mathbf{X},q,a)} = \eta_{(psioa(\mathbf{X}),q,a)}$ and we extend this notation with $\eta_{(\mathbf{X},q,a)} = \delta_q$ if $a \notin \widehat{sig}(psioa(\mathbf{X}))(q)$.

5 Executions, reachable states, partially-compatible automata

In previous sections, we have described how to model probabilistic transitions that might lead to the creation and destruction of some components of the system. In this section, we will define pseudo execution fragments of a set of automata to model the run of a set $\mathbf{A}$ of several dynamic systems interacting with each others. With such a definition, we will kill two birds with one stone, since it will allow to define *reachable states* of $\mathbf{A}$ and then compatibility of $\mathbf{A}$ as compatibility of $\mathbf{A}$ at each reachable state.

5.1 Executions, reachable states, traces

▶ **Definition 14** (Pseudo execution, reachable states, partial-compatibility)**.** *Let $\mathbf{A} = \{\mathcal{A}_1, ..., \mathcal{A}_n\}$ be a finite set of PSIOA (resp. PCA). A* pseudo execution fragment *of $\mathbf{A}$ is a finite or infinite sequence $\alpha = q^0 a^1 q^1 a^2 ...$ of alternating states and actions, such that:*

1. *If α is finite, it ends with a state. In that case, we note $lstate(\alpha)$ the last state of α.*
2. *$\mathbf{A}$ is compatible at each state of α, with the potential exception of $lstate(\alpha)$ if α is finite.*
3. *for ever action a^i, $(q^{i-1}, a^i, q^i) \in steps(\mathbf{A})$.*

The first state of a pseudo execution fragment α is noted $fstate(\alpha)$. A pseudo execution fragment α of $\mathbf{A}$ is a pseudo execution of $\mathbf{A}$ if $fstate(\alpha) = \bar{q}_{\mathbf{A}}$. The length $|\alpha|$ of a finite pseudo execution fragment α is the number of actions in α. A state q of $\mathbf{A}$ is said reachable *if there is a pseudo execution α s.t. $lstate(\alpha) = q$. We note $Reachable(\mathbf{A})$ the set of* reachable *states of $\mathbf{A}$. If $\mathbf{A}$ is compatible at every reachable state q, $\mathbf{A}$ is said* partially-compatible.

▶ **Definition 15** (Executions, concatenations)**.** *Let $\mathcal{A}$ be an automaton. An* execution fragment *(resp.* execution*) of $\mathcal{A}$ is a pseudo execution fragment (resp. pseudo execution) of $\{\mathcal{A}\}$. We use $Frags(\mathcal{A})$ (resp., $Frags^*(\mathcal{A})$) to denote the set of all (resp., all finite) execution fragments of $\mathcal{A}$. $Execs(\mathcal{A})$ (resp. $Execs^*(\mathcal{A})$) denotes the set of all (resp., all finite) executions of $\mathcal{A}$.*

We define a concatenation operator $\frown$ for execution fragments as follows:
If $\alpha = q^0 a^1 q^1 ... a^n q^n \in Frags^*(\mathcal{A})$ and $\alpha' = q^{0\prime} a^{1\prime} q^{1\prime} ... \in Frags^*(\mathcal{A})$, we define $\alpha \frown \alpha' \triangleq q^0 a^1 q^1 ... a^n q^n \; a^{1\prime} q^{1\prime} ...$ only if $s^0 = q^n$, otherwise $\alpha \frown \alpha'$ is undefined. Hence the notation $\alpha \frown \alpha'$ implicitly means $fstate(\alpha') = lstate(\alpha)$. Let $\alpha, \alpha' \in Frags(\mathcal{A})$, then α is a prefix of α', noted $\alpha \leq \alpha'$, iff $\exists \alpha'' \in Frags(\mathcal{A})$ such that $\alpha' = \alpha \frown \alpha''$.

The *trace* of an execution α represents its externally visible part, i.e. the external actions.

[3] We can remark that the conjunction of PSIOA compatibility and sub-automata exclusivity implies the compatibility of respective configurations as defined later in definition 19

▶ **Definition 16** (Traces). *Let $\mathcal{A}$ be a PSIOA (resp. PCA). Let $q^0 \in Q_{\mathcal{A}}$, $(q, a, q') \in steps(\mathcal{A})$, $\alpha, \alpha' \in Execs^*(\mathcal{A}) \times Execs(\mathcal{A})$ with $fstate(\alpha') = lstate(\alpha)$.*

$trace_{\mathcal{A}}(q^0)$ *is the empty sequence, noted* λ,

$$trace_{\mathcal{A}}(qaq') \begin{cases} a \text{ if } a \in \widehat{ext}(\mathcal{A})(q) \\ \lambda \text{ otherwise.} \end{cases},$$

$$trace_{\mathcal{A}}(\alpha^\frown\alpha') = trace_{\mathcal{A}}(\alpha)^\frown trace_{\mathcal{A}}(\alpha')$$

We say that β is a trace of $\mathcal{A}$ if $\exists \alpha \in Execs(\mathcal{A})$ with $\beta = trace_{\mathcal{A}}(\alpha)$. We note $Traces(\mathcal{A})$ (resp. $Traces^*(\mathcal{A})$, resp. $Traces^\omega(\mathcal{A})$) the set of traces (resp. finite traces, resp. infinite traces) of $\mathcal{A}$. When the automaton $\mathcal{A}$ is understood from context, we write simply $trace(\alpha)$.

The projection of a pseudo-execution α on an automaton $\mathcal{A}_i$, noted $\alpha \upharpoonright \mathcal{A}_i$, represents the contribution of $\mathcal{A}_i$ to this execution.

▶ **Definition 17** (Projection). *Let $\mathbf{A}$ be a set of PSIOA (resp. PCA), let $\mathcal{A}_i \in \mathbf{A}$. We define projection operator $\upharpoonright$ recursively as follows: For every $(q, a, q') \in steps(\mathbf{A})$, for every α, α' being two pseudo executions of $\mathbf{A}$ with $fstate(\alpha') = lstate(\alpha)$.*

$$(q, a, q') \upharpoonright \mathcal{A}_i = \begin{cases} (q \upharpoonright \mathcal{A}_i), a, (q' \upharpoonright \mathcal{A}_i) \text{ if } a \in \widehat{sig}(\mathcal{A}_i)(q \upharpoonright \mathcal{A}_i) \\ (q \upharpoonright \mathcal{A}_i) = (q' \upharpoonright \mathcal{A}_i) \text{ otherwise.} \end{cases},$$

$$(\alpha^\frown\alpha') \upharpoonright \mathcal{A}_i = (\alpha \upharpoonright \mathcal{A}_i)^\frown(\alpha' \upharpoonright \mathcal{A}_i)$$

5.2 PSIOA and PCA composition

We are ready to define composition operator, the most important operator for concurrent systems.

▶ **Definition 18** (PSIOA partial-composition). *If $\mathbf{A} = \{\mathcal{A}_1, ..., \mathcal{A}_n\}$ is a partially-compatible set of PSIOA, with $\mathcal{A}_i = (Q_{\mathcal{A}_i}, \bar{q}_{\mathcal{A}_i}, sig(\mathcal{A}_i), D_{\mathcal{A}_i})$, then their partial-composition $\mathcal{A}_1||...||\mathcal{A}_n$, is defined to be $\mathcal{A} = (Q_{\mathcal{A}}, \bar{q}_{\mathcal{A}}, sig(\mathcal{A}), D_{\mathcal{A}})$, where:*

- $Q_{\mathcal{A}} = Reachable(\mathbf{A})$
- $\bar{q}_{\mathcal{A}} = (\bar{q}_{\mathcal{A}_1}, ..., \bar{q}_{\mathcal{A}_n})$
- $sig(\mathcal{A}) : q \in Q_{\mathcal{A}} \mapsto sig(\mathcal{A})(q) = sig(\mathbf{A})(q)$
- $D_{\mathcal{A}} = \{(q, a, \eta_{(\mathbf{A},q,a)}) | q \in Q_{\mathcal{A}}, a \in \widehat{sig}(\mathbf{A})(q)\}$

▶ **Definition 19** (Union of configurations). *Let $C_1 = (\mathbf{A}_1, \mathbf{S}_1)$ and $C_2 = (\mathbf{A}_2, \mathbf{S}_2)$ be configurations such that $\mathbf{A}_1 \cap \mathbf{A}_2 = \emptyset$. Then, the union of C_1 and C_2, denoted $C_1 \cup C_2$, is the configuration $C = (\mathbf{A}, \mathbf{S})$ where $\mathbf{A} = \mathbf{A}_1 \cup \mathbf{A}_2$ and $\mathbf{S}$ agrees with $\mathbf{S}_1$ on $\mathbf{A}_1$, and with $\mathbf{S}_2$ on $\mathbf{A}_2$. Moreover, if $C_1 \cup C_2$ is a compatible configuration, we say that C_1 and C_2 are* compatible configurations*. It is clear that configuration union is commutative and associative. Hence, we will freely use the n-ary notation $C_1 \cup ... \cup C_n$, whenever $\forall i, j \in [1:n], i \neq j, auts(C_i) \cap auts(C_j) = \emptyset$.*

▶ **Definition 20** (PCA partial-composition). *If $\mathbf{X} = \{X_1, ..., X_n\}$ is a partially-compatible set of PCA, then their partial-composition $X_1||...||X_n$, is defined to be the automaton X, with the same components than a PCA, s.t. $psioa(X) = psioa(X_1)||...||psioa(X_n)$ and $\forall q \in Q_X$:*

- $config(X)(q) = \bigcup_{i \in [1,n]} config(X_i)(q \upharpoonright X_i)$
- $\forall a \in \widehat{sig}(X)(q)$, *$created(X)(q)(a) = \bigcup_{i \in [1,n]} created(X_i)(q \upharpoonright X_i)(a)$, with the convention $created(X_i)(q_i)(a) = \emptyset$ if $a \notin \widehat{sig}(X_i)(q_i)$*
- $hidden\text{-}actions(q) = \bigcup_{i \in [1,n]} hidden\text{-}actions(X_i)(q \upharpoonright X_i)$

▶ **Theorem 21** (PCA closeness under composition). *Let $X_1, ..., X_n$, be partially-compatible PCA. Then $X = X_1||...||X_n$ is a PCA.*

6 Scheduler, measure on executions, implementation

An inherent non-determinism appears for concurrent systems. Indeed, after composition (or even before), it is natural to obtain a state with several enabled actions. The most common case is the reception of two concurrent messages in flight from two different processes. This non-determinism must be solved if we want to define a probability measure on the automata executions and be able to say that a situation is likely to occur or not. To solve the non-determinism, we use a scheduler that chooses an enabled action from a signature.

6.1 General definition and probabilistic space on execution fragments

A scheduler is hence a function that takes an execution fragment as input and outputs the probability distribution on the set of transitions that will be triggered. We reuse the formalism from [19] with the syntax from [3].

▶ **Definition 22** (Scheduler). *A* scheduler *of a PSIOA (resp. PCA) $\mathcal{A}$ is a function*

$$\sigma : Frags^*(\mathcal{A}) \to SubDisc(D_{\mathcal{A}}) \text{ such that } (q, a, \eta) \in supp(\sigma(\alpha)) \text{ implies } q = lstate(\alpha).$$

Here $SubDisc(D_{\mathcal{A}})$ is the set of discrete sub-probability distributions on $D_{\mathcal{A}}$. Loosely speaking, σ decides (probabilistically) which transition to take after each finite execution fragment α. Since this decision is a discrete sub-probability measure, it may be the case that σ chooses to halt after α with non-zero probability: $1 - \sigma(\alpha)(D_{\mathcal{A}}) > 0$. We note $schedulers(\mathcal{A})$ the set of schedulers of $\mathcal{A}$.

▶ **Definition 23** (Measure $\epsilon_{\sigma,\alpha}$ generated by a scheduler and a fragment). *A scheduler σ and a finite execution fragment α generate a measure $\epsilon_{\sigma,\alpha}$ on the sigma-algebra $\mathcal{F}_{Frags(\mathcal{A})}$ generated by cones of execution fragments, where each cone $C_{\alpha'}$ is the set of execution fragments that have α' as a prefix, i.e. $C_{\alpha'} = \{\alpha \in Frags(\mathcal{A}) | \alpha' \leq \alpha\}$. The measure of a cone $C_{\alpha'}$ is defined recursively as follows:*

$$\epsilon_{\sigma,\alpha}(C_{\alpha'}) =: \begin{cases} 0 & \text{if both } \alpha' \nleq \alpha \text{ and } \alpha \nleq \alpha' \\ 1 & \text{if } \alpha' \leq \alpha \\ \epsilon_{\sigma,\alpha}(C_{\alpha''}) \cdot \sigma(\alpha'')(\eta_{(\mathcal{A},q',a)}) \cdot \eta_{(\mathcal{A},q',a)}(q) & \text{if } \alpha \leq \alpha'' \text{ and } \alpha' = \alpha'' {}^\frown q'aq \end{cases}$$

In the remaining part of the paper, we will mainly focus on probabilistic executions of $\mathcal{A}$ of the form $\epsilon_\sigma \triangleq \epsilon_{\sigma,\delta_{\bar{q}_{\mathcal{A}}}} = \epsilon_{\sigma,\bar{q}_{\mathcal{A}}}$. Hence, we will deal with probablistic space of the form $(Execs(\mathcal{A}), \mathcal{F}_{Execs(\mathcal{A})}, \epsilon_\sigma)$.

Scheduler Schema

Without restriction, a scheduler could become a too powerful adversary for practical applications. Hence, it is common to only consider a subset of schedulers, called a *scheduler schema*. Typically, a classic limitation is often described by a scheduler with "partial online information". Some formalism has already been proposed in [19] (section 5.6) to impose the scheduler that its choices are correlated for executions fragments in the same equivalence class where both the equivalence relation and the correlation must to be defined. This idea has been reused and simplified in [4] that defines equivalence classes on actions, called *tasks*. Then, a task-scheduler (a.k.a. "off-line" scheduler) selects a sequence of tasks $T_1, T_2, ...$ in advance that it cannot modify during the execution of the automaton. After each transition, the next task T_i triggers an enabled action if there is no ambiguity and is ignored otherwise. One of our main contribution, the theorem of implementation monotonicity w.r.t. PSIOA creation, is ensured only for a certain scheduler schema, so-called *creation-oblivious*. However, we will see that the practical set of task-schedulers are not creation-oblivious.

▶ **Definition 24** (Scheduler schema). *A* scheduler schema *is a function that maps every PSIOA (resp. PCA) $\mathcal{A}$ to a subset of schedulers$(\mathcal{A})$.*

6.2 Implementation

In last subsection, we defined a measure of probability on executions with the help of a scheduler to solve non-determinism. Now we can define the notion of implementation. The intuition behind this notion is the fact that any environment $\mathcal{E}$ that would interact with both $\mathcal{A}$ and $\mathcal{B}$, would not be able to distinguish $\mathcal{A}$ from $\mathcal{B}$. The classic use-case is to formally show that a (potentially very sophisticated) algorithm implements a specification.

For us, an environment is simply a partially-compatible automaton, but in practice, he will play the role of a "distinguisher".

▶ **Definition 25** (Environment). *A probabilistic environment for PSIOA $\mathcal{A}$ is a PSIOA $\mathcal{E}$ such that $\mathcal{A}$ and $\mathcal{E}$ are partially-compatible. We note $env(\mathcal{A})$ the set of environments of $\mathcal{A}$.*

Now we define *perception function* which is a function that captures the pieces of information that could be obtained by an external observer to attempt a distinction.

▶ **Definition 26** (Perception function). *A* perception-function *is a function $f_{(.,.)}$ parametrized by a pair $(\mathcal{E}, \mathcal{A})$ of PSIOA (resp. PCA) where $\mathcal{E} \in env(\mathcal{A})$ s.t.*

- *(Measurability) For every pair $(\mathcal{E}, \mathcal{A})$ of PSIOA (resp. PCA) where $\mathcal{E} \in env(\mathcal{A})$, $f_{(\mathcal{E},\mathcal{A})}$ is a measurable function from $(Execs(\mathcal{E}||\mathcal{A}), \mathcal{F}_{Execs(\mathcal{E}||\mathcal{A})})$ to some measurable space $(G_{(\mathcal{E},\mathcal{A})}, \mathcal{F}_{G_{(\mathcal{E},\mathcal{A})}})$ that has to be made explicit.*
- *(Stability by composition) For every quadruplet of PSIOA $(\mathcal{A}_1, \mathcal{A}_2, \mathcal{B}, \mathcal{E})$, s.t. $\mathcal{B}$ is partially compatible with $\mathcal{A}_1$ and $\mathcal{A}_2$, $\mathcal{E} \in env(\mathcal{B}||\mathcal{A}_1) \cap env(\mathcal{B}||\mathcal{A}_2)$, $\forall (C_1, C_2) \in \mathcal{F}_{Execs(\mathcal{E}||\mathcal{B}||\mathcal{A}_1)} \times \mathcal{F}_{Execs(\mathcal{E}||\mathcal{B}||\mathcal{A}_2)}$, $f_{(\mathcal{E}||\mathcal{B},\mathcal{A}_1)}(C_1) = f_{(\mathcal{E}||\mathcal{B},\mathcal{A}_2)}(C_2) \Longrightarrow f_{(\mathcal{E},\mathcal{B}||\mathcal{A}_1)}(C_1) = f_{(\mathcal{E},\mathcal{B}||\mathcal{A}_2)}(C_2)$.*

The first property is a standard measurability requirement, while the second captures the fact that an environment $\mathcal{E}$ does not have a greater power of distinction than $\mathcal{E}$ composed with another system $\mathcal{B}$. Any reasonable function that captures the perception of an automaton $\mathcal{A}$ by an environment $\mathcal{E} \in env(\mathcal{A})$ should be a perception function.

▶ **Lemma 27.** *The function $trace_{(.,.)}$ and $proj_{(.,.)}$ s.t. for every PSIOA (resp. PCA) $\mathcal{A}$, $\forall \mathcal{E} \in env(\mathcal{A})$, $trace_{(\mathcal{E},\mathcal{A})} : \alpha \in Execs(\mathcal{E}||\mathcal{A}) \mapsto trace_{(\mathcal{E}||\mathcal{A})}(\alpha)$ and $proj_{(\mathcal{E},\mathcal{A})} : \alpha \in Execs(\mathcal{E}||\mathcal{A}) \mapsto \alpha \upharpoonright \mathcal{E}$ are perception functions.*

Since a perception-function $f_{(.,.)}$ is measurable, we can define the image measure of $\epsilon_{\sigma,\mu}$ under $f_{(\mathcal{E},\mathcal{A})}$, i.e. the probability to obtain a certain external perception under a certain scheduler σ and a certain probability distribution μ on the starting executions.

▶ **Definition 28** (f-dist). *Let $f_{(.,.)}$ be a perception-function. Let $(\mathcal{E}, \mathcal{A})$ be a pair of PSIOA where $\mathcal{E} \in env(\mathcal{A})$. Let μ be a probability measure on $(Execs(\mathcal{E}||\mathcal{A}), \mathcal{F}_{Execs(\mathcal{E}||\mathcal{A})})$, and $\sigma \in schedulers(\mathcal{E}||\mathcal{A})$. We define $f\text{-}dist_{(\mathcal{E},\mathcal{A})}(\sigma, \mu)$, to be the image measure of $\epsilon_{\sigma,\mu}$ under $f_{(\mathcal{E},\mathcal{A})}$ (i.e. the function that maps any $C \in \mathcal{F}_{G_{(\mathcal{E},\mathcal{A})}}$ to $\epsilon_{\sigma,\mu}(f^{-1}_{(\mathcal{E},\mathcal{A})}(C))$) . We note $f\text{-}dist_{(\mathcal{E},\mathcal{A})}(\sigma)$ for $f\text{-}dist_{(\mathcal{E},\mathcal{A})}(\sigma, \delta_{\bar{q}(\mathcal{E}||\mathcal{A})})$.*

We can see next definition of f-implementation as the incapacity of an environment to distinguish two automata if it uses only information filtered by the perception function f.

▶ **Definition 29** (f-implementation). *Let $f_{(.,.)}$ be an insight-function. Let S be a scheduler schema. We say that $\mathcal{A}$ f-*implements *$\mathcal{B}$* according to *S, noted $\mathcal{A} \leq_0^{S,f} \mathcal{B}$, if $\forall \mathcal{E} \in env(\mathcal{A}) \cap env(\mathcal{B})$, $\forall \sigma \in S(\mathcal{E}||\mathcal{A})$, $\exists \sigma' \in S(\mathcal{E}||\mathcal{B})$, $f\text{-}dist_{(\mathcal{E},\mathcal{A})}(\sigma) \equiv f\text{-}dist_{(\mathcal{E},\mathcal{B})}(\sigma')$, i.e. $\forall C \in supp(f\text{-}dist_{(\mathcal{E},\mathcal{A})}(\sigma)) \cup supp(f\text{-}dist_{(\mathcal{E},\mathcal{B})}(\sigma'))$, $f\text{-}dist_{(\mathcal{E},\mathcal{A})}(\sigma)(C) = f\text{-}dist_{(\mathcal{E},\mathcal{B})}(\sigma')(C)$.*

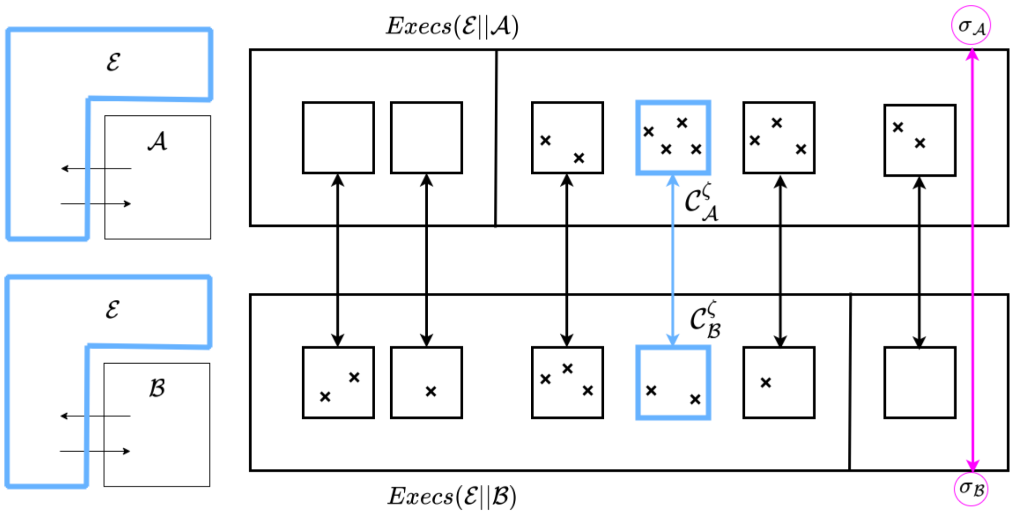

$\mathcal{E}$
$\mathcal{A}$
$\mathcal{E}$
$\mathcal{B}$
$Execs(\mathcal{E}||\mathcal{A})$
$Execs(\mathcal{E}||\mathcal{B})$
$\sigma_{\mathcal{A}}$
$\sigma_{\mathcal{B}}$
$C^{\zeta}_{\mathcal{A}}$
$C^{\zeta}_{\mathcal{B}}$

Figure 3 We say that $\mathcal{A}$ implements $\mathcal{B}$ if no environment $\mathcal{E}$ is able to distinguish $\mathcal{A}$ from $\mathcal{B}$, i.e. $\forall \sigma_{\mathcal{A}} \in schedulers(\mathcal{E}||\mathcal{A})$, $\exists \sigma_{\mathcal{B}} \in schedulers(\mathcal{E}||\mathcal{B})$ (linked by pink arrow) s.t. every pair of corresponding classes of equivalence of executions, related to the same perception by the environment (e.g. $(C^{\zeta}_{\mathcal{A}}, C^{\zeta}_{\mathcal{B}})$ in blue for perception ζ) are equiprobable, i.e. $f\text{-}dist_{(\mathcal{E},\mathcal{A})}(\sigma_{\mathcal{A}})(\zeta) = f\text{-}dist_{(\mathcal{E},\mathcal{B})}(\sigma_{\mathcal{B}})(\zeta)$.

We can restate classic theorem of (horizontal) substitutability of implementation in a quite general form.

▶ **Theorem 30** (Implementation substitutability). *Let $f_{(.,.)}$ be a perception-function. Let S be a scheduler schema. Let $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$, $\mathcal{B}$, $\mathcal{B}_1$, $\mathcal{B}_2$ be some PSIOA (resp. PCA).*

- *(Composability) If $\mathcal{A}_1 \leq_0^{S,f} \mathcal{A}_2$ and $\mathcal{B}$ is partially compatible with $\mathcal{A}_1$ and $\mathcal{A}_2$, then $\mathcal{B}||\mathcal{A}_1 \leq_0^{S,f} \mathcal{B}||\mathcal{A}_2$.*
- *(Transitivity) If $\mathcal{A}_1 \leq_0^{S,f} \mathcal{A}_2$ and $\mathcal{A}_2 \leq_0^{S,f} \mathcal{A}_3$, then $\mathcal{A}_1 \leq_0^{S,f} \mathcal{A}_3$.*
- *(Substitutability) If $\mathcal{A}_1 \leq_0^{S,f} \mathcal{A}_2$, $\mathcal{B}_1 \leq_0^{S,f} \mathcal{B}_2$, and both $\mathcal{B}_1$ and $\mathcal{B}_2$ are partially compatible with both $\mathcal{A}_1$ and $\mathcal{A}_2$, then $\mathcal{A}_1||\mathcal{B}_1 \leq_0^{S,f} \mathcal{A}_2||\mathcal{B}_2$.*

Substitutability constitutes one of the most important properties that an implementation relation should satisfy, since it allows to reason in a modular way and avoid overwhelming monolithic proof of correctness.

7 Dynamic vertical substitutability

In previous section, we have stated the classic horizontal substitutability of implementation relation, which allows us to replace an idealized abstract object by its concrete implementation without losing hyper-properties. In this section, we informally describe the main result of this paper: the dynamic vertical substitutability of p-implementation.

Informally, if (1) $X_{\mathcal{A}}$ and $X_{\mathcal{B}}$ are PCA that differ only on the fact that $\mathcal{B}$ supplants $\mathcal{A}$ in $X_{\mathcal{B}}$ and (2) $\mathcal{A}R\mathcal{B}$ for some preorder R implies (3) $X_{\mathcal{A}}RX_{\mathcal{B}}$, then we say that R is monotonic w.r.t. PSIOA creation/destruction. Monotonicity of implementation w.r.t. PSIOA creation/destruction is the main contribution of the paper.

▶ **Definition 31** ((Informal) corresponding w.r.t. $\mathcal{A}$, $\mathcal{B}$). *Intuitively, $X_{\mathcal{A}}$ and $X_{\mathcal{B}}$ are corresponding w.r.t. $\mathcal{A}$, $\mathcal{B}$ if they differ only in that $X_{\mathcal{A}}$ dynamically creates and destroys automaton $\mathcal{A}$ instead of creating and destroying automaton $\mathcal{B}$ as $X_{\mathcal{B}}$ does. Some technical minor assumptions have to be verified:*

- *$X_{\mathcal{A}}$ is $\mathcal{A}$-conservative and $X_{\mathcal{B}}$ is $\mathcal{B}$-conservative: Each state of $X_{\mathcal{A}}$ (resp. $X_{\mathcal{B}}$) is perfectly defined by its configuration deprived of sub-automaton $\mathcal{A}$ (resp. $\mathcal{B}$) and external actions of $\mathcal{A}$ (resp. $\mathcal{B}$) are not hidden.*
- *$X_{\mathcal{A}}$ is $\mathcal{A}$-creation explicit and $X_{\mathcal{B}}$ is $\mathcal{B}$-creation explicit: the creation of $\mathcal{A}$ and $\mathcal{B}$ respectively, are equivalent to the triggering of an action in a dedicated set.*
- *$config(X_{\mathcal{A}})(\bar{q}_{X_{\mathcal{A}}}) \lhd_{AB} config(X_{\mathcal{B}})(\bar{q}_{X_{\mathcal{B}}})$: The associated configuration of respective start states are identical except that the automaton $\mathcal{B}$ supplants $\mathcal{A}$ but with the same external signature.*
- *$X_{\mathcal{A}}, X_{\mathcal{B}}$ are creation&hiding-corresponding w.r.t. $\mathcal{A}, \mathcal{B}$: the two PCA hide some output actions and create some PSIOA in the same manner, excepting for the creation of $\mathcal{B}$ that supplants the creation of $\mathcal{A}$.*
- *$\forall \mathcal{K} \in \{\mathcal{A}, \mathcal{B}\}$, $\forall q \in Q_{X_{\mathcal{K}}}$, for every $\mathcal{K}$-exclusive action a at state q, $created(X_{\mathcal{K}})(q)(a) = \emptyset$, where a $\mathcal{K}$-exclusive action is an action which is in the signature of sub-automaton $\mathcal{K}$ only.*

We would like to state the monotonicity of p-implementation, but it holds only for a certain class of schedulers, so-called *creation-oblivious* that does not take past internal behaviours of sub-automata into account to outputs the next action.

▶ **Definition 32** ((Informal) creation-oblivious scheduler). *Let $\tilde{A}$ be a PSIOA, $\tilde{W}$ be a PCA, $\tilde{\sigma} \in schedulers(\tilde{W})$. We say that $\tilde{\sigma}$ is $\mathcal{A}$-*creation oblivious *if for every triplet $(\tilde{\alpha}_1, \tilde{\alpha}_2, \tilde{\alpha}_3)$ s.t. (1) $lstate(\tilde{\alpha}_1) = lstate(\tilde{\alpha}_2) = fstate(\tilde{\alpha}_3)$ and (2) $\tilde{\alpha}_1$ and $\tilde{\alpha}_2$ differ only on $\mathcal{A}$-exclusive actions and internal states of sub-automaton $\mathcal{A}$, then (3) $\tilde{\sigma}(\tilde{\alpha}_1^\frown \tilde{\alpha}_3) = \tilde{\sigma}(\tilde{\alpha}_2^\frown \tilde{\alpha}_3)$.*

Formal definitions of two last concepts are available in the extended version. It is crucial to limit the power of the scheduler to reduce the measure of a class of comportment as a function of measures of classes of shorter comportment where no creation of $\mathcal{A}$ or $\mathcal{B}$ occurs excepting potentially at very last action. This reduction is more or less necessary to obtain monotonicity of implementation relation:

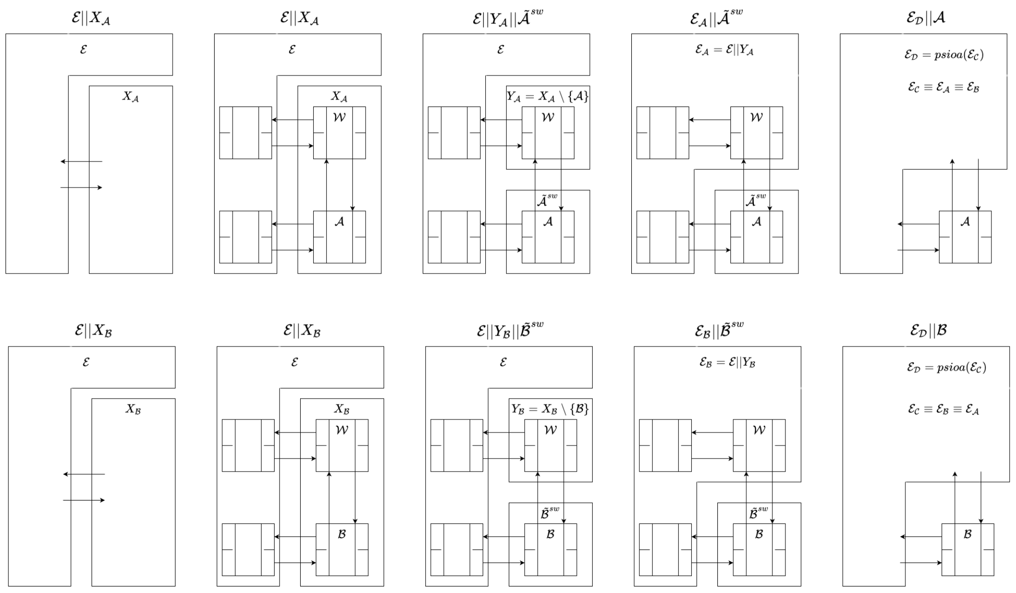
▶ **Theorem 33** (p-implementation monotonicity). *Let $\mathcal{A}, \mathcal{B} \in Autids$, $X_{\mathcal{A}}$ and $X_{\mathcal{B}}$ be PCA corresponding w.r.t. $\mathcal{A}$, $\mathcal{B}$. Let S the schema of creation-oblivious scheduler and $p = proj_{(.,.)}$. If $\mathcal{A} \leq_0^{S,p} \mathcal{B}$, then $X_{\mathcal{A}} \leq_0^{S,p} X_{\mathcal{B}}$*

Proof Sketch. First, we defined the notion of executions-matching to capture the idea that two automata have the same "comportment" along some corresponding executions. Basically an executions-matching from a PSIOA $\mathcal{A}$ to a PSIOA $\mathcal{B}$ is a morphism $f^{ex} : Execs'_{\mathcal{A}} \to Execs(\mathcal{B})$ where $Execs'_{\mathcal{A}} \subseteq Execs(\mathcal{A})$. This morphism preserves some properties along the pair of matched executions: signature, transition, ... in such a way that for every pair $(\alpha, \alpha') \in Execs(\mathcal{A}) \times Execs(\mathcal{B})$ s.t. $\alpha' = f^{ex}(\alpha)$, $\epsilon_\sigma(\alpha) = \epsilon_{\sigma'}(\alpha')$ for every pair of schedulers (σ, σ') (so-called *alter ego*) that are "very similar" in the sense they take into account only the "structure" of the argument to return a sub-probability distribution, i.e. $\alpha' = f^{ex}(\alpha)$ implies $\sigma(\alpha) = \sigma'(\alpha')$. When the executions-matching is a bijection function from $Execs(\mathcal{A})$ to $Execs(\mathcal{B})$, we say $\mathcal{A}$ and $\mathcal{B}$ are semantically-equivalent (they differ only syntactically). Second, we defined the notion of a PCA $X_{\mathcal{A}}$ deprived of a PSIOA $\mathcal{A}$, noted $(X_{\mathcal{A}} \setminus \{\mathcal{A}\})$. Such an automaton corresponds to the intuition of a similar automaton where $\mathcal{A}$ is systematically removed from the configuration of the original PCA. Thereafter we shew that under technical minor assumptions $X_{\mathcal{A}} \setminus \{\mathcal{A}\}$ and $\tilde{\mathcal{A}}^{sw}$ are partially-compatible where $\tilde{\mathcal{A}}^{sw}$ and $\mathcal{A}$ are semantically equivalent. In fact $\tilde{\mathcal{A}}^{sw}$ is the simpleton wrapper of $\mathcal{A}$, that is a PCA that only owns $\mathcal{A}$ in its attached configuration. Then we shew that there is an (incomplete) execution-matching from $X_{\mathcal{A}}$ to $(X_{\mathcal{A}} \setminus \{\mathcal{A}\})||\tilde{\mathcal{A}}^{sw}$. The domain of this executions-matching is the set of executions where $\mathcal{A}$ is not (re-)created before very last action. After this, we always try to reduce any reasoning on $X_{\mathcal{A}}$ (resp. $X_{\mathcal{B}}$) on a reasoning on $(X_{\mathcal{A}} \setminus \{\mathcal{A}\})||\tilde{\mathcal{A}}^{sw}$ (resp. $(X_{\mathcal{B}} \setminus \{\mathcal{B}\})||\tilde{\mathcal{B}}^{sw}$). We shew that, under certain reasonable technical assumptions (captured in the definition of corresponding PCA w.r.t. $\mathcal{A}$, $\mathcal{B}$), $(X_{\mathcal{A}} \setminus \{\mathcal{A}\})$ and $(X_{\mathcal{B}} \setminus \{\mathcal{B}\})$ are semantically-equivalent. We can note Y an arbitrary PCA semantically-equivalent to $(X_{\mathcal{A}} \setminus \{\mathcal{A}\})$ and $(X_{\mathcal{B}} \setminus \{\mathcal{B}\})$. Finally, a reasoning on $\mathcal{E}||X_{\mathcal{A}}$ (resp. $\mathcal{E}||X_{\mathcal{B}}$) can be reduced to a reasoning on $\mathcal{E}'||\tilde{\mathcal{A}}^{sw}$ (resp. $\mathcal{E}'||\tilde{\mathcal{B}}^{sw}$) with $\mathcal{E}' = \mathcal{E}||Y$. Since $\tilde{\mathcal{A}}^{sw}$ implements $\tilde{\mathcal{B}}^{sw}$, we have already some results on $\mathcal{E}'||\tilde{\mathcal{A}}^{sw}$ and $\mathcal{E}'||\tilde{\mathcal{B}}^{sw}$ and so on $\mathcal{E}||X_{\mathcal{A}}$ and $\mathcal{E}||X_{\mathcal{B}}$. However, this reduction, represented in figure 4, is valid only for the subset of executions without creation of neither $\mathcal{A}$ nor $\mathcal{B}$ before very last action. Ideally, we would like to decompose an "aggregated" class of perception, with arbitrary number of creations/destructions of $\mathcal{A}$ (resp. $\mathcal{B}$)), into "atomic" classes of perception without creation/destruction of $\mathcal{A}$ (resp. $\mathcal{B}$)) before last action. Some technical precautions have to be taken to be allowed to paste these fragments together to finally say that $\mathcal{A}$ implements $\mathcal{B}$ implies $X_{\mathcal{A}}$ implements $X_{\mathcal{B}}$. In fact, such a pasting is generally not possible for a fully information online scheduler. This observation motivated us to introduce the creation-oblivious scheduler, to manipulate independent atomic classes of perception. We proved monotonicity of external behaviour inclusion for schema of creation oblivious scheduler. Surprisingly, the fully-offline task-scheduler introduced in [3] (slightly modified to be adapted to dynamic setting) is not creation-oblivious and so does not allow monotonicity of implementation. ◀

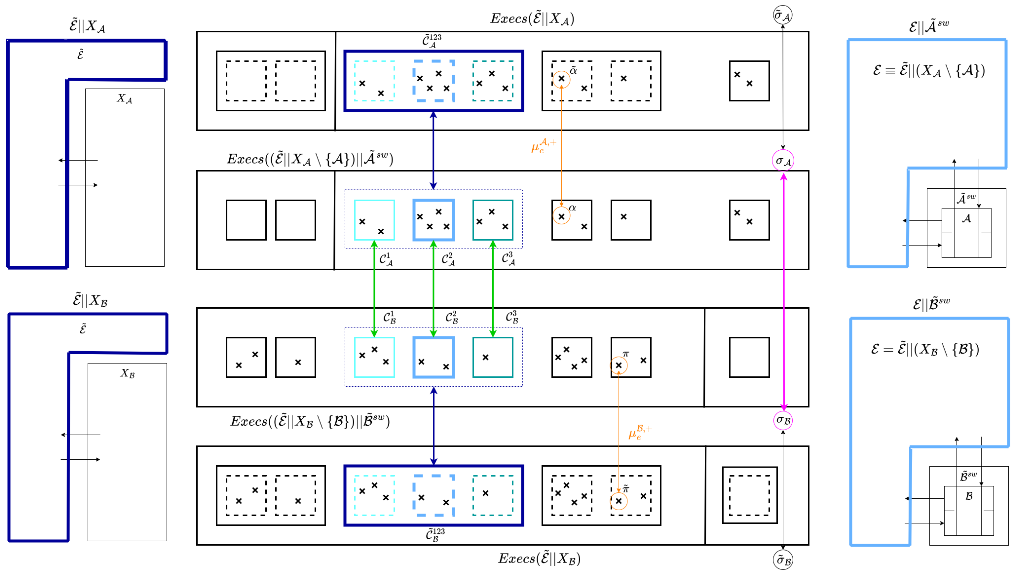
8 Conclusion

We have extended *dynamic I/O Automata* formalism of Attie & Lynch [2] to probabilistic setting in order to cope with emergent distributed systems such as peer-to-peer networks, robot networks, adhoc networks or blockchains. Our formalism includes operators for parallel composition, action hiding, action renaming, automaton creation and use a refined definition of probabilistic configuration automata in order to cope with dynamic actions. The key result of our framework is as follows: the implementation of probabilistic configuration automata is monotonic to automata creation and destruction. That is, if systems $X_{\mathcal{A}}$ and $X_{\mathcal{B}}$ differ only in that $X_{\mathcal{A}}$ dynamically creates and destroys automaton $\mathcal{A}$ instead of creating and destroying automaton $\mathcal{B}$ as $X_{\mathcal{B}}$ does, and if $\mathcal{A}$ implements $\mathcal{B}$ (in the sense they cannot be distinguished by any external observer), then $X_{\mathcal{A}}$ implements $X_{\mathcal{B}}$. This results is particularly interesting in the design and refinement of components and subsystems in isolation. In our construction we exhibit the need of considering only *creation-oblivious* schedulers in the implementation relation, i.e. a scheduler that, upon the (dynamic) creation of a sub-automaton $\mathcal{A}$, does not take into account the previous internal behaviour of $\mathcal{A}$ to output (randomly) a transition.

As future work we plan to extend the composable secure-emulation of Canetti et al. [5] to dynamic settings. This extension is necessary for formal verification of protocols combining probabilistic distributed systems and cryptography in dynamic settings (e.g. blockchains, secure distributed computation, cybersecure distributed protocols etc).



(a) The figure represents successive steps to reduce the problem of an environment $\mathcal{E}$ that tries to distinguish two PCA $X_{\mathcal{A}}$ and $X_{\mathcal{B}}$ (represented at first column) to a problem of an environment $\mathcal{E}_{\mathcal{D}}$ that tries to distinguish the automata $\mathcal{A}$ and $\mathcal{B}$ (represented at last column).



(b) The figure represents the homomorphism enabling the reduction reasoning, for set of executions that do not create neither $\mathcal{A}$ nor $\mathcal{B}$ before last action. For every environment $\mathcal{E}$, For every scheduler $\sigma_{\mathcal{A}}$, there exists a corresponding scheduler $\sigma_{\mathcal{B}}$ (mapped with pink arrow) s.t. for every possible perception ζ (represented in light blue), the probability to observe ζ is the same for $\mathcal{E}$ in each world. There is an homomorphism $\mu_e^{\mathcal{A},+}$ (orange arrow) between $\tilde{\mathcal{E}}||X_{\mathcal{A}}$ and $\mathcal{E}||\tilde{\mathcal{A}}^{sw}$ (and similarly for $X_{\mathcal{B}}$ and $\tilde{\mathcal{B}}^{sw}$) s.t. for every scheduler $\tilde{\sigma}_{\mathcal{A}}$, alter-ego of $\sigma_{\mathcal{A}}$, the measure of each corresponding perception is preserved. Hence, for every environment $\tilde{\mathcal{E}}$, for every scheduler $\tilde{\sigma}_{\mathcal{A}}$, there exists a corresponding scheduler $\tilde{\sigma}_{\mathcal{B}}$ s.t. for every possible perception $\tilde{\zeta}$ (represented in dark blue), the probability to observe $\tilde{\zeta}$ is the same for $\tilde{\mathcal{E}}$ in each world.

Figure 4 homomorphism-based-proof.

References

1 Edward A. Ashcroft. Proving assertions about parallel programs. *J. Comput. Syst. Sci.*, 10(1):110–135, 1975. doi:10.1016/S0022-0000(75)80018-3.

2 Paul C. Attie and Nancy A. Lynch. Dynamic input/output automata: A formal and compositional model for dynamic systems. *Inf. Comput.*, 249:28–75, 2016. doi:10.1016/j.ic.2016.03.008.

3 Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto Segala. Task-Structured Probabilistic {I/O} Automata. *Journal of Computer and System Sciences*, 94:63—-97, 2018. doi:10.1016/j.jcss.2017.09.007.

4 Ran Canetti, Ling Cheung, Dilsun Kirli Kaynar, Moses D. Liskov, Nancy A. Lynch, Olivier Pereira, and Roberto Segala. Using probabilistic I/O automata to analyze an oblivious transfer protocol. *IACR Cryptol. ePrint Arch.*, page 452, 2005. URL: http://eprint.iacr.org/2005/452.

5 Ran Canetti, Ling Cheung, Dilsun Kirli Kaynar, Nancy A. Lynch, and Olivier Pereira. Compositional security for task-pioas. In *20th IEEE Computer Security Foundations Symposium, CSF 2007, 6-8 July 2007, Venice, Italy*, pages 125–139. IEEE Computer Society, 2007. doi:10.1109/CSF.2007.15.

6 Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019. doi:10.1016/j.tcs.2019.02.001.

7 Pierre Civit and Maria Potop-Butucaru. Probabilistic dynamic input output automata (extended version). Cryptology ePrint Archive, Paper 2021/798, 2021. doi:10.4230/LIPIcs.DISC.2022.20.

8 Pierre Civit and Maria Potop-Butucaru. Brief announcement: Composable dynamic secure emulation. In Kunal Agrawal and I-Ting Angelina Lee, editors, *SPAA '22: 34th ACM Symposium on Parallelism in Algorithms and Architectures, Philadelphia, PA, USA, July 11 - 14, 2022*, pages 103–105. ACM, 2022. doi:10.1145/3490148.3538562.

9 C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

10 Richard M. Karp and Raymond E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969. doi:10.1016/S0022-0000(69)80011-5.

11 Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Software Eng.*, 3(2):125–143, 1977. doi:10.1109/TSE.1977.229904.

12 Nancy Lynch, Michael Merritt, William Weihl, and Alan Fekete. A theory of atomic transactions. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 326 LNCS:41–71, 1988. doi:10.1007/3-540-50171-1_3.

13 Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science.* Springer, 1980. doi:10.1007/3-540-10235-3.

14 Rocco De Nicola and Roberto Segala. A process algebraic view of input/output automata. *Theor. Comput. Sci.*, 138(2):391–423, 1995. doi:10.1016/0304-3975(95)92307-J.

15 Susan S. Owicki and David Gries. An axiomatic proof technique for parallel programs I. *Acta Informatica*, 6:319–340, 1976. doi:10.1007/BF00268134.

16 C.A. Petri. *Kommunikation mit Automaten.* PhD thesis, Institut für instrumentelle Mathematik, Bonn, 1962.

17 Martin L. Puterman. *Markov decision processes: discrete stochastic dynamic programming.* Wiley series in probability and mathematical statistics. John Wiley & Sons, 1 edition, 1994.

18 Alejandro Ranchal-Pedrosa and Vincent Gramoli. Platypus: Offchain protocol without synchrony. In Aris Gkoulalas-Divanis, Mirco Marchetti, and Dimiter R. Avresky, editors, *18th IEEE International Symposium on Network Computing and Applications, NCA 2019, Cambridge, MA, USA, September 26-28, 2019*, pages 1–8. IEEE, 2019. doi:10.1109/NCA.2019.8935037.

19 Roberto Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, Massachusettes Institute of technology, 1995.

20 Frits W. Vaandrager. On the relationship between process algebra and input/output automata. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 387–398. IEEE Computer Society, 1991. `doi:10.1109/LICS.1991.151662`.

21 Kazuki Yoneyama. Formal modeling of random oracle programmability and verification of signature unforgeability using task-pioas. *Int. J. Inf. Sec.*, 17(1):43–66, 2018. `doi:10.1007/s10207-016-0352-y`.