Probabilistic Automata: System Types, Parallel Composition and Comparison

Ana Sokolova[1] and Erik P. de Vink[2]

[1] Department of Mathematics and Computer Science,
TU/e, Eindhoven
a.sokolova@tue.nl
[2] LIACS, Leiden University
evink@win.tue.nl

Abstract. We survey various notions of probabilistic automata and probabilistic bisimulation, accumulating in an expressiveness hierarchy of probabilistic system types. The aim of this paper is twofold: On the one hand it provides an overview of existing types of probabilistic systems and, on the other hand, it explains the relationship between these models. We overview probabilistic systems with discrete probabilities only. The expressiveness order used to built the hierarchy is defined via the existence of mappings between the corresponding system types that preserve and reflect bisimilarity. Additionally, we discuss parallel composition for the presented types of systems, augmenting the map of probabilistic automata with closedness under this compositional operator.

Keywords: probabilistic automata (transition systems), probabilistic bisimulation, preservation and reflection of bisimulation, non-determinism, parallel composition.

1 Introduction

The notion of a state machine has proved useful in many modelling situations, amongst others, the area of validation of stochastic systems. In the literature up to now, a great variety of types of probabilistic automata has been proposed and many of these have been actually used for verification purposes. In this paper we discuss a number of probabilistic automata with discrete probability distributions. For continuous-time probabilistic systems the interested reader is referred to [11, 33, 32, 17, 45, 4]. Models of stochastic systems that are not represented by transition systems can also be found in [22] and [70].

Due to the variety of proposed models it is often the case that results have to be interpreted from one type of systems to another. Therefore we compare the considered types of probabilistic automata in terms of their expressiveness. The comparison is achieved by placing a partial order on the classes of such automata, where one class is less then another if each automaton in the class can be translated to an automaton of the other class such that translations both reflect and preserve the respective notions of bisimilarity. Hence, bisimulation and

C. Baier et al. (Eds.): Validation of Stochastic Systems, LNCS 2925, pp. 1–43, 2004.

bisimilarity are central notions in this overview. Other comparison criteria are important as well, e.g. logical properties, logical characterization of bisimulation [61], complexity of algorithms for deciding bisimulation [9, 13, 31, 80] and so on. We choose the comparison criterion formulated in terms of strong bisimulation because of its simplicity and because we work with transition labelled systems, for which bisimulation semantics arises naturally from the step-by-step behavior.

A major distinction of probabilistic automata is that between fully probabilistic vs. non-deterministic ones. In a fully probabilistic automaton every choice is governed by a probability distribution (over set of states or states combined with actions). The probability distribution captures the uncertainty about the next state. If we abstract away from the actions in a fully probabilistic automaton, we are left with a discrete time Markov chain. Subsequently, standard techniques can be applied to analyze the resulting Markov chains. Sometimes, the incomplete knowledge about the system behavior can not be represented probabilistically. In these cases we should consider more than one transition possible. We speak in this case of a non-deterministic probabilistic automaton. Most of the models that we consider include some form of non-determinism and hence fall in the category of non-deterministic probabilistic automata. As pointed out by various authors, e.g. [47, 76, 3, 81] non-determinism is essential for modelling scheduling freedom, implementation freedom, the external environment and incomplete information. Furthermore, non-determinism is essential for the definition of an asynchronous parallel composition operator that allows interleaving. Often two kinds of non-deterministic choices are mentioned in the literature (see for e.g. [81]), *external* non-deterministic choices influenced by the environment, specified by having several transitions with different labels leaving from the same state, and *internal* non-determinism, exhibited by having several transitions with the same label leaving from a state. We use the term non-determinism for *full non-determinism* including both internal and external non-deterministic choices.

We introduce several classes of automata, ranging from the simplest models to more complex ones. The questions that we will address for each individual class are:

- the definition of the type of automaton and the respective notion of strong bisimulation;
- the relation of the model with other models;
- presence and form of non-determinism;
- the notion of a product or parallel composition in the model.

The set-up of the paper is as follows: Section 2 presents the necessary notions considering probability theory, automata (transition systems), and concurrency theory, in particular compositional operators. In section 3 we focus on the various definitions of probabilistic automata in isolation with their corresponding notions of bisimulation. In section 4 the operators of parallel composition are discussed. We address the interrelationship between the introduced types of automata in section 5. Section 6 wraps up with some conclusions.

Acknowledgements

We would like to thank Holger Hermanns for editorial support and for the plentitude of useful ideas and directions, as well as the other organizers of VOSS GI/Dagstuhl 2002 for initiating and organizing this nice event. We are in dept to the referees for various remarks. Special thanks go to Falk Bartels for his major contribution regarding the hierarchy of probabilistic systems, as well as for numerous comments, suggestions and his friendly cooperation.

2 Basic Ingredients

2.1 Probability Distributions

Let Ω be a set. A function $\mu\colon \Omega \to [0,1]$ is called a *discrete probability distribution*, or distribution for short, on Ω if $\{x \in \Omega |\ \mu(x) > 0\}$ is finite or countably infinite and $\sum_{x\in\Omega} \mu(x) = 1$. The set $\{x \in \Omega |\ \mu(x) > 0\}$ is called the *support* of μ and is denoted by $spt(\mu)$. If $x \in \Omega$, then μ_x^1 denotes the unique probability distribution with $\mu_x^1(x) = 1$, also known as the *Dirac distribution* for x. When μ is a distribution on Ω we use the notation $\mu[X]$ for $\sum_{x\in X} \mu(x)$ where $X \subseteq \Omega$. By $\mathcal{D}(\Omega)$ we denote the set of all discrete probability distributions on the set Ω. If μ is a distribution with finite support $\{s_1, \ldots, s_n\}$, we sometimes write $\{s_1 \mapsto \mu(s_1), \ldots, s_n \mapsto \mu(s_n)\}$. With this notation, $\mu_x^1 = \{x \mapsto 1\}$.

Let $\mu_1 \in \mathcal{D}(S)$ and $\mu_2 \in \mathcal{D}(T)$. The product $\mu_1 \times \mu_2$ of μ_1 and μ_2 is a distribution on $S \times T$ defined by $(\mu_1 \times \mu_2)(s,t) = \mu_1(s) \cdot \mu_2(t)$, for $\langle s,t\rangle \in S \times T$.

If $\mu \in \mathcal{D}(S \times T)$, we use the notation $\mu[s,T]$ for $\mu[\{s\} \times T]$ and $\mu[S,t]$ for $\mu[S \times \{t\}]$. We adopt from [51] the lifting of a relation between two sets to a relation between distributions on these sets.

Definition 1. *Let $R \subseteq S \times T$ be a relation between the sets S and T. Let $\mu \in \mathcal{D}(S)$ and $\mu' \in \mathcal{D}(T)$ be distributions. Define $\mu \equiv_R \mu'$ if and only if there exists a distribution $\nu \in \mathcal{D}(S \times T)$ such that*

1. *$\nu[s,T] = \mu(s)$ for any $s \in S$*
2. *$\nu[S,t] = \mu'(t)$ for any $t \in T$*
3. *$\nu(s,t) \neq 0$ if and only if $\langle s,t\rangle \in R$.*

The lifting of a relation R preserves the characteristic properties of preorders and equivalences (cf. [52]). For the special case of an equivalence relation there is a simpler way to define the lifting (cf. [52, 81, 9]).

Proposition 1. *Let R be an equivalence relation on the set S and let $\mu, \mu' \in \mathcal{D}(S)$. Then $\mu \equiv_R \mu'$ if and only if $\mu[C] = \mu'[C]$ for all equivalence classes $C \in S/R$.* □

Lifting of an equivalence relation on a set S to a relation $\equiv_{R,A}$ on the set $\mathcal{D}(A \times S)$, for a fixed set A, will also be needed.

Definition 2. *Let R be an equivalence relation on a set S, A a set, and let $\mu, \mu' \in \mathcal{D}(A \times S)$. Define*

$$\mu \equiv_{R,A} \mu' \iff \forall C \in S/R, \forall a \in A\colon \mu[a,C] = \mu'[a,C]$$

2.2 Non-probabilistic Automata, Markov Chains, Bisimilarity

Throughout the paper we will use the terms automaton, transition system or just system as synonyms.

Non-probabilistic Automata

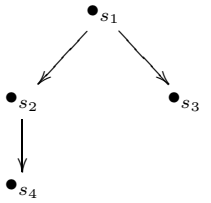
Definition 3. *A transition system, TS for short, is a pair* $\langle S, \alpha \rangle$ *where*

1. S *is a set of states*
2. $\alpha : S \to \mathcal{P}(S)$ *is a transition function, where* $\mathcal{P}$ *denotes the powerset of* S*.*

If $\langle S, \alpha \rangle$ is a transition system such that $s, s' \in S$ and $s' \in \alpha(s)$ we write $s \to s'$ and call it a transition.

Often in the literature a TS is given as a triple, including besides the set of states and the transition function also a subset of initial states, or a single initial state. In this paper we will consider **no initial states** and therefore they are not present in the definition. Instead of a transition function one could equivalently consider a transition relation as a subset of $S \times S$. Our choice here is to always present the transitions via a **transition function**.

A way of representing a TS is via its transition diagram. For example, the system $\langle S, \alpha \rangle$ where $S = \{s_1, s_2, s_3, s_4\}$ and $\alpha(s_1) = \{s_2, s_3\}$, $\alpha(s_2) = \{s_4\}$, $\alpha(s_3) = \alpha(s_4) = \emptyset$, is represented as follows:



The states s_3 and s_4 are *terminating* states, with no outgoing transitions.

It is often of use to model the phenomenon that a change of state in a system happens as a result of executing an *action*. Therefore, labelled transition systems evolve from transition systems. There are two ways to incorporate labels in a TS: by labelling the states (usually with some values of variables, or a set of propositions true in a state), or by explicitly labelling the transitions with actions or action names. In this paper we focus on **transition labelled systems**.

Definition 4. *A labelled transition system (LTS) (or a non-deterministic automaton) is a triple* $\langle S, A, \alpha \rangle$ *where*

1. S *is a set of states*
2. A *is a set of actions*
3. $\alpha : S \to \mathcal{P}(A \times S)$ *is a transition function.*

When $\langle S, A, \alpha \rangle$ is a LTS, then the transition function α can equivalently be considered as a function from S to $\mathcal{P}(S)^A$, the collection of functions from A

to $\mathcal{P}(S)$. As in the case of TSs, for any state $s \in S$ of a LTS, every element $\langle a, s' \rangle \in \alpha(s)$ determines a transition which is denoted by $s \xrightarrow{a} s'$.

The class of non-deterministic automata (LTSs) is denoted by **NA**. Deterministic automata, given by the next definition, form a subclass of **NA**.

Definition 5. *A deterministic automaton is a triple $\langle S, A, \alpha \rangle$ where*

1. *S is a set of states*
2. *A is a set of actions*
3. *$\alpha : S \to (S+1)^A$ is a transition function.*

Notation 1 *We denote by $+$ the disjoint union of two sets. The set 1 is a singleton set containing the special element $*$, i.e. $1 = \{*\}$. We assume that $* \notin S$. The notation $(S+1)^A$ stands for the collection of all functions from A to $S+1$.*

The special set 1 and the disjoint union construction allow us to write partial functions as functions. Hence, in a deterministic automaton each state s is assigned a partial function $\alpha(s)\colon A \to S+1$ from the set of actions to the set of states, meaning that whenever $\alpha(s)(a) = s'$ for some $s' \in S$, i.e. $\alpha(s) \neq *$, then there is a transition $s \xrightarrow{a} s'$ enabled in S. We denote the class of all deterministic automata by **DA**.

We note that the class of automata **DA** exhibits external non-determinism, while in **NA** there is full non-determinism.

external non-determinism full non-determinism

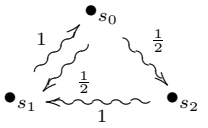
Markov Chains. The simplest class of fully probabilistic automata is the class of discrete time Markov chains. The theory of Markov chains is rich and huge (see, e.g., [57, 48, 16, 43]) and we only provide a simple definition of a discrete time Markov chain here.

Definition 6. *A Markov chain is a pair $\langle\, S, \alpha \,\rangle$ where*

1. *S is a set of states*
2. *$\alpha : S \to \mathcal{D}(S)$ is a transition function.*

Markov chains evolve from transition systems, when probability is added to each transition such that for any state the sum of the probabilities of all outgoing transitions equals 1. The class of all Markov chains is denoted by **MC**. If $s \in S$ and $\alpha(s) = \mu$ with $\mu(s') = p > 0$ then the Markov chain $\langle\, S, \alpha \,\rangle$ is said to go from a state s with probability p to a state s'. Notation: $s \rightsquigarrow \mu$ and $s \overset{p}{\rightsquigarrow} s'$.

Example 1.



$S = \{s_0, s_1, s_2\}$
$\alpha(s_0) = \{s_0 \mapsto 0, s_1 \mapsto \frac{1}{2}, s_2 \mapsto \frac{1}{2}\}$
$\alpha(s_1) = \mu^1_{s_0}$
$\alpha(s_2) = \mu^1_{s_1}$

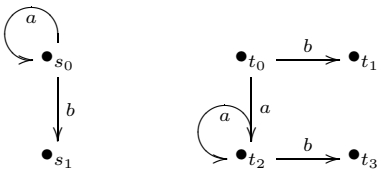
Bisimulation and Bisimilarity. Different semantics or notions of behavior can be given to labelled transition systems. We work with the bisimulation semantics (Milner [65, 66]) stating that two states in a system represented by LTSs are equivalent whenever there exists a bisimulation relation that relates them. A bisimulation relation compares the one-step behavior of two states and has a nice extension to the probabilistic case (as explored in [61]). In [54] probabilistic extensions of a number of other well known process equivalences have been studied like probability trace, completed trace, failure and ready equivalence. Other probabilistic process equivalences are probabilistic simulation and bisimulation by Segala and Lynch [78, 76], Yi and Larsen's testing equivalence [88], and CSP equivalences of Morgan et al. [67], Lowe [59] and Seidel [77]. An overview of several probabilistic process equivalences can be found in [58].

Definition 7. *Let $\langle S, A, \alpha\rangle$ and $\langle T, A, \alpha\rangle$ be two LTSs. A relation $R \subseteq S \times T$ is a* bisimulation *relation if for all $\langle s, t\rangle \in R$ and all $a \in A$ the following holds*

if $s \xrightarrow{a} s'$ then there exists $t' \in T$ such that $t \xrightarrow{a} t'$ and $\langle s', t'\rangle \in R$, and
if $t \xrightarrow{a} t'$ then there exists $s' \in S$ such that $s \xrightarrow{a} s'$ and $\langle s', t'\rangle \in R$.

Let $s \in S$ and $t \in T$. The states s and t are called bisimilar*, denoted by $s \approx t$ if there exists a bisimulation relation R with $\langle s, t\rangle \in R$.*

Example 2. For the following LTSs we have, for example, $s_0 \approx t_0$ since $R = \{\langle s_0, t_0\rangle, \langle s_0, t_2\rangle, \langle s_1, t_1\rangle, \langle s_1, t_3\rangle\}$ is a bisimulation.



Remark 1. Instead of comparing states in two systems $\langle S, A, \alpha\rangle$ and $\langle T, A, \beta\rangle$ we can always consider one joined system $\langle S + T, A, \gamma\rangle$ with $\gamma(s) = \alpha(s)$ for $s \in S$ and $\gamma(t) = \beta(t)$ for $t \in T$. Therefore bisimulation can be defined as a relation on the set of states of a system. Furthermore, if $R \subseteq S \times S$ is a bisimulation, then it is reflexive and symmetric, and the transitive closure of R is also a bisimulation. Hence bisimilarity $\approx$ is not affected by the choice of defining bisimulation as an equivalence.

Definition 8. *An equivalence relation R on a set of states S of a LTS is an equivalence bisimulation if for all $\langle s, t\rangle \in R$ and all $a \in A$*

$$\text{if } s \xrightarrow{a} s' \text{ then } \exists t' \in S\colon\ t \xrightarrow{a} t',\ \langle s', t'\rangle \in R$$

The states s and t are called bisimilar*, denoted by $s \approx_e t$ if there exists an equivalence bisimulation R with $\langle s, t\rangle \in R$.*

By Remark 1, the following proposition holds.

Proposition 2. *Let $\langle S, A, \alpha\rangle$ and $\langle T, A, \beta\rangle$ be two LTSs, and let $s \in S,\ t \in T$. Then $s \approx t$ if and only if $s \approx_e t$.*

□

Bisimulation on **DA** is defined exactly the same as for **NA** i.e. with Definition 8.

The standard notion of probabilistic bisimulation is the one introduced by Larsen and Skou [61] originally formulated for reactive systems (see next subsection). An early reference to probabilistic bisimulation can be found in [23]. In the case of Markov chains, bisimulation corresponds to ordinary lumpability of Markov chains [57, 44, 27]. In [86, 85] it is shown that the concrete notion of bismulation for Markov-chains coincides with a general coalgebraic notion of bisimulation [68, 53, 74, 64].

The idea behind probabilistic bisimulation is as follows. Since bisimilar states are considered "the same", it does not matter which element within a bisimulation class is reached. Hence, a bisimulation relation should compare the probability to reach an equivalence class and not the probability to reach a single state. In order to define bisimulation for Markov chains the lifting of a relation on a state S to a relation on $\mathcal{D}(S)$, as defined in Definition 1 and explained with Proposition 1, is used. Note that the comments of Remark 1 are in place here as well.

Definition 9. *An equivalence relation R on a set of states S of a Markov chain $\langle S, \alpha\rangle$ is a bisimulation if and only if for all $\langle s, t\rangle \in R$*

$$\textit{if } s \rightsquigarrow \mu \textit{ then there is a transition } t \rightsquigarrow \mu' \textit{ with } \mu \equiv_R \mu'.$$

The states s and t are called bisimilar*, denoted by $s \approx t$, if there exists a bisimulation R with $\langle s, t\rangle \in R$.*

Definition 9 will be used, with some variations, for defining bisimulation relations for all types of probabilistic automata that we consider in this overview. However, note that in the case of Markov chains any two states of any two Markov chains are bisimilar, according to the given definition, since $\nabla = S \times S$ is a bisimulation on the state set of any Markov chain $\langle S, \alpha\rangle$. Namely, let $\langle S, \alpha\rangle$ be a Markov chain and $s, t \in S$, such that $\alpha(s) = \mu, \alpha(t) = \mu'$, i.e., $s \rightsquigarrow \mu, t \rightsquigarrow \mu'$. Then for the only equivalence class of ∇, S, we have $\mu[S] = 1 = \mu'[S]$ i.e. $\mu \equiv_R \mu'$ which makes $s \approx t$. This phenomenon can be explained with the fact that bisimilarity compares the observable behavior of two states in a system and the Markov chains are very simple systems in which there is not much to observe. Therefore the need comes to enrich Markov chains with actions or at least termination.

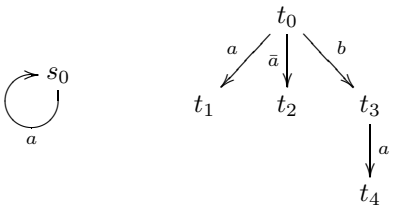
Notation. In Section 3 we will introduce ten other types of probabilistic automata, with corresponding notions of bisimulation. In order to avoid repetition we collect the following.

- A type of automata will always be a triple $\langle S, A, \alpha \rangle$ where S is a set of states, A is a set of actions and α is a transition function. The difference between the system types is expressed with the difference in the codomains of the corresponding transition functions.
- A bisimulation relation will always be defined as an equivalence on the set of states of a system. Depending on the type of systems the "transfer conditions" in the definition of bisimulation vary.
- For a particular type of system, the bisimilarity relation, denoted by $\approx$ is defined by: $s \approx t$ if and only if there exists a bisimulation R that relates s and t, i.e. $\langle s, t \rangle \in R$. Although we use the same notation $\approx$ for bisimilarity in different types of systems, it should be clear that for each type of systems, $\approx$ is a different relation.

2.3 Parallel Composition of LTSs and MCs

Compositional operators serve the need of modular specification and verification of systems. They arise from process calculi, such as CCS ([66]), CSP ([47]) and ACP ([19]), where process terms (models of processes) are built from atomic process terms with the use of compositional operators. Usually a model of a process calculi is a suitable class of transition systems. Therefore it is often the case that process terms are identified with their corresponding transition systems, and the compositional operators of the process calculus can be considered as operators for combining transition systems. In this overview we focus on the parallel composition operator. The definition of parallel composition varies a lot throughout different process calculi. In this section we consider the non-probabilistic case (LTSs) in order to explain variants of different parallel compositions, and the parallel composition of Markov chains in order to present the basics of probabilistic parallel composition.

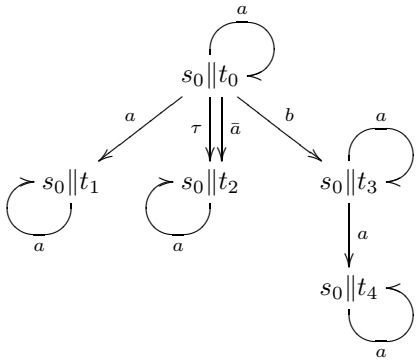
Labelled Transition Systems. A major distinction between different parallel composition operators is whether they are *synchronous*, where the components are forced to synchronize whenever they can, or *asynchronous* where the components can either synchronize or act independently. Furthermore, different approaches for synchronization exist. The result of the parallel composition of two automata $\mathcal{A}_1 = \langle S_1, A, \alpha_1 \rangle$ and $\mathcal{A}_2 = \langle S_2, A, \alpha_2 \rangle$ is an automaton $\mathcal{A}_1 \| \mathcal{A}_2 = \langle S_1 \times S_2, A, \alpha \rangle$ where the definition of α varies. Instead of a pair $\langle s, t \rangle \in S_1 \times S_2$ we will write $s \| t$ for a state in the composed automaton. Throughout this subsection we will use as running example, the parallel composition of the following two automata.



CCS style: The set of actions in this case contains compatible actions $a, \bar{a} \in A$ and a special idle or internal action $\tau \in A$. If one of the automata in state s can perform an action a changing to a state s' and the other one in state t can perform a's compatible action $\bar{a}$ moving to state t' then the composite automaton in state $s \| t$ can perform the idle action τ and move to state $s' \| t'$. Furthermore, independent behavior of each of the automata is possible within the composed automaton.

$s \| t \xrightarrow{a} s' \| t'$ if and only if

1. $s \xrightarrow{b} s', t \xrightarrow{\bar{b}} t', a = \tau$, for b and $\bar{b}$ compatible actions, or
2. $s \xrightarrow{a} s'$ and $t' = t$, or
3. $t \xrightarrow{a} t'$ and $s' = s$.

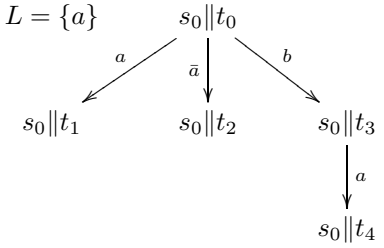


The presented CCS parallel composition is asynchronous. A synchronous variant (SCCS [65]) is defined by omitting clauses 2. and 3. in the definition above.

CSP style: Communication or synchronization in a CSP style parallel composition occurs on a set of synchronizing actions. Thus actions that are intended to synchronize are listed in a set $L \subseteq A$ and the rest of the actions can be performed independently.

$s \|_L t \xrightarrow{a} s' \| t'$ if and only if

1. $s \xrightarrow{a} s'$ and $t \xrightarrow{a} t'$ and $a \in L$, or
2. $s \xrightarrow{a} s'$, $t = t'$ and $a \notin L$, or
3. $t \xrightarrow{a} t'$, $s = s'$ and $a \notin L$.



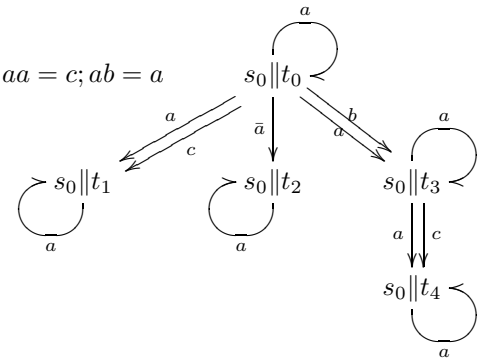
This type of parallel composition operator is synchronous for $L = A$, expresses only interleaving (shuffling) composition if $L = \emptyset$ and is never fully asynchronous with both independent behavior and communication allowed. An asynchronous CSP style parallel composition can be defined by omitting the clause "$a \notin L$"

in clauses 2. and 3. above. In case of different action sets A_1 and A_2, of the two component automata, L is taken to be a subset of $A_1 \cap A_2$. If $L = A_1 \cap A_2$ then we say that synchronization on common actions occurs.

ACP style: In ACP, parallel composition is fully asynchronous, allowing both interleaving (independent behavior) and synchronization via a communication function. A communication function is a commutative and associative partial function $\gamma : A \times A \hookrightarrow A$. Instead of $\gamma(a, b)$ we will write ab.

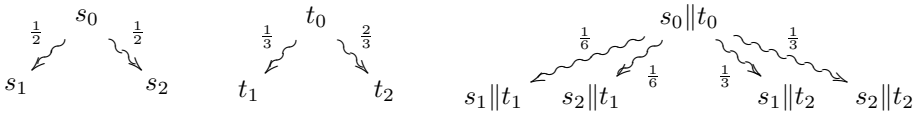
$s\|t \xrightarrow{a} s'\|t'$ if and only if

1. $s \xrightarrow{b} s'$ and $t \xrightarrow{c} t'$ with $bc = a$ defined, or
2. $s \xrightarrow{a} s'$, $t = t'$, or
3. $t \xrightarrow{a} t'$, $s = s'$.



Note that if A contains compatible actions and an idle action τ, and if $a\bar{a} = \tau$ for any compatible $a, \bar{a} \in A$ and undefined otherwise, then the ACP parallel composition operator specializes to the CCS parallel composition operator. On the other hand, for $aa = a$, $(a \in L \subseteq A)$ we get the asynchronous variant of the CSP parallel composition operator. If clauses 2. and 3. are dropped from the definition, we get a synchronous variant of the ACP parallel composition operator called communication merge.

Markov Chains. Let $\mathcal{M}_1 = \langle S_1, \alpha_1 \rangle, \mathcal{M}_2 = \langle S_2, \alpha_2 \rangle$ be two Markov chains. Their parallel product is the Markov chain $\mathcal{M}_1 \| \mathcal{M}_2 = \langle S_1 \times S_2, \alpha \rangle$, where $\alpha(s\|t) = \alpha_1(s) \times \alpha_2(t)$, $\times$ denoting the product of distributions. Hence $s\|t \rightsquigarrow \mu$ if and only if $s \rightsquigarrow \mu_1, t \rightsquigarrow \mu_2$ and $\mu = \mu_1 \times \mu_2$.



Note that the parallel composition of two Markov chains is synchronous, since each step in the composed automaton consists of independent steps performed by each of the components. The way of defining the product of two distributions goes in favor of the interpretation that when put in parallel, each of the automata independently chooses its transition that contributes to a transition in the composed automaton.

3 Probabilistic Models

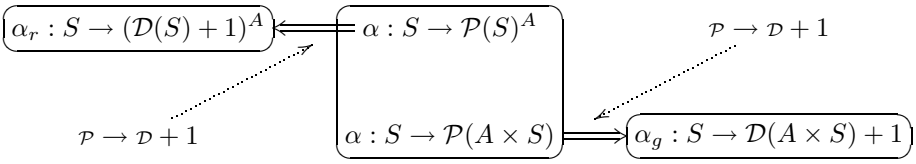
This section defines the advanced types of probabilistic automata. The automata types are grouped in several subsections reflecting their common properties. Basically, every type of probabilistic automata arises from the plain definition of a transition system with or without labels. Probabilities can then be added either to every transition, or to transitions labelled with the same action, or there can be a distinction between probabilistic and ordinary (non-deterministic) states, where only the former ones include probabilistic information, or the transition function can be equipped with structure that provides both non-determinism and probability distributions.

Each kind of probabilistic automata comes equipped with a notion of bisimulation, and all these notions, frequently only subtly different, will also find their way in this section.

3.1 Reactive, Generative and I/O Probabilistic Automata

Two classical extensions of LTSs with probabilities are the reactive and the generative model. Throughout the years a large amount of research has been devoted to reactive and generative probabilistic systems. It is hard to note who introduced these systems first, but the reactive model was treated e.g. in [61, 62, 40, 39], the generative in e.g. [40, 39, 42, 50, 30, 29, 28], and the classification of these systems together with a so-called stratified model was proposed in [39, 40].

The way these models arise from LTSs, by changing the transition function, can be explained with the following figure, where α denotes the transition function of a LTS, α_r and α_g the transition function of a reactive and a generative system, respectively.



Definition 10. *A reactive probabilistic automaton is a triple $\langle S, A, \alpha \rangle$ where the transition function is given by*

$$\alpha : S \to (\mathcal{D}(S) + 1)^A.$$

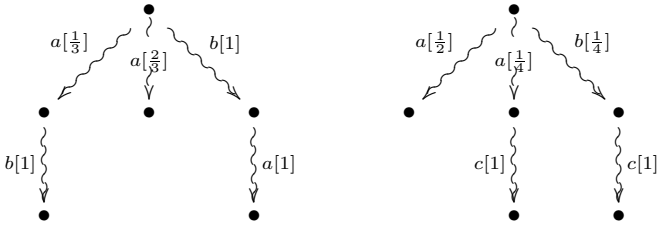
If $s \in S$ and $\alpha(s)(a) = \mu_a$ then we write $s \xrightarrow{a}\!\!\rightsquigarrow \mu_a$. More specifically, if $s' \in spt(\mu_a)$, $\mu_a(s') = p$ we write $s \overset{a[p]}{\rightsquigarrow} s'$.

A generative probabilistic automaton is a triple $\langle S, A, \alpha\rangle$ *with a transition function*

$$\alpha : S \to \mathcal{D}(A \times S) + 1.$$

When $s \in S$ *and* $\alpha(s) = \mu \in \mathcal{D}(A \times S)$ *then we write* $s \rightsquigarrow \mu$. *More particularly, if* $\langle a, s'\rangle \in spt(\mu)$ *with* $\mu(\langle a, s'\rangle) = p$ *we write* $s \overset{a[p]}{\rightsquigarrow} s'$. *We use* $s \not\rightsquigarrow$ *to denote that* $\alpha(s) = *$.

Remark 2. In Definition 10 both uses of the special singleton set 1 appear. The first one, as in Definition 5 helps expressing partial functions. The second one, in the definition of generative transition function, expresses the possibility of termination. If s is a state in a generative system with $\alpha(s) = *$ then s is a terminating state allowing no transition. For LTSs, termination is allowed by the fact that $\emptyset \in \mathcal{P}(A \times S)$. Hence, when changing from subsets to distributions, $*$ is added to play the role of the $\emptyset$.



Reactive system Generative system

In a reactive system probabilities are distributed over the outgoing transitions labelled with the *same action*, while in a generative system probabilities are distributed over *all* outgoing transitions from a state. A motivation for making this distinction is the different treatment of actions. In a reactive system actions are treated as *input* actions being provided by the environment. When a reactive system receives input from the environment then it acts probabilistically by choosing the next state according to a probability distribution assigned to this input. There are no probabilistic assumptions about the behavior of the environment. On the other hand, in a generative system, as the name suggests, actions are treated as *output* generated by the system. When a generative system is in a state s it chooses the next transition according to the probability distribution $\alpha(s)$ assigned to s. The transition being chosen, the system moves to another state while generating the output action which labels this transition. Note that in a generative system there is no non-determinism present, while in a reactive system there is only external non-determinism, as in **DA**. We denote by **React** and **Gen** the classes of reactive and generative probabilistic automata, respectively.

Definition 11. *An equivalence relation* R *on* S *is a bisimulation on the reactive probabilistic automaton* $\langle S, A, \alpha\rangle$ *if for all* $\langle s, t\rangle \in R$ *and for all actions* $a \in A$*:*

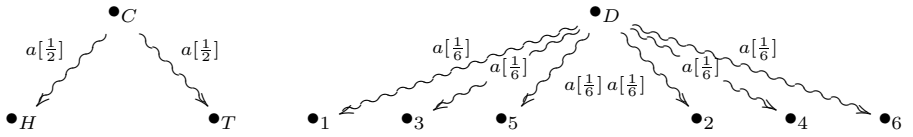
if $s \xrightarrow{a} \rightsquigarrow \mu$ *then there exists a distribution* μ' *with* $t \xrightarrow{a} \rightsquigarrow \mu'$ *and* $\mu \equiv_R \mu'$.

In order to state the definition of bisimulation for generative systems, the lifting from Definition 2 is used.

Definition 12. *An equivalence relation R on S is a bisimulation on the generative probabilistic automaton $\langle S, A, \alpha\rangle$ if for all $\langle s,t\rangle \in R$:*

$$\text{if } s \leadsto \mu \text{ then there exists a distribution } \mu' \text{ with } t \leadsto \mu' \text{ and } \mu \equiv_{R,A} \mu'.$$

Example 3. The equivalence relation R generated by the pairs $\langle C, D\rangle$, $\langle H, 1\rangle$, $\langle H, 3\rangle$, $\langle H, 5\rangle$, $\langle T, 2\rangle$, $\langle T, 4\rangle$, $\langle T, 6\rangle$ is a bisimulation for the probabilistic automaton given below. Hence, $C \approx D$. Note that this particular automaton belongs to both **React** and **Gen**.



An intuitive interpretation of this example is obtained by adding meaning "flip" to the action a in the left sub-automaton and a meaning "roll" to the action a in the right sub-automaton. Then the state C represents flipping of a fair coin, and the state D represents rolling a fair dice. The bisimilarity of the states C and D shows that it is the same whether one flips a fair coin or rolls a fair dice being interested only in whether the outcome is odd or even.

I/O Probabilistic Automata. The model of input/output probabilistic automata, introduced by Wu, Smolka and Stark in [87], exploiting the input/output automata by Lynch and Tuttle, (cf. [63]), presents a combination of the reactive and the generative model.

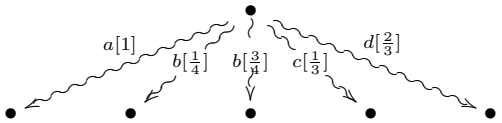
Definition 13. *An input/output probabilistic automaton is a triple $\langle S, A, \alpha\rangle$ where*

1. *the set of actions A is divided into input and output actions, $A = A^{in} + A^{out}$;*
2. *$\alpha : S \to \mathcal{D}(S)^{A^{in}} \times (\mathcal{D}(A^{out} \times S) + 1) \times \mathbb{R}_{\geq 0}$ is the transition function.*

The third component in the transition function assigns an output delay rate *to each state. If $s \in S$, then $\alpha(s) = \langle f^{in}, \mu^{out}, \delta_s\rangle$. We have that $\delta_s = 0$ iff $\mu^{out} = *$ i.e. delay is assigned only to states that generate output.*

Denote the class of I/O automata by **IO**. We use a similar notation for transitions as in the reactive and the generative model. If $s \in S$ with $\alpha(s) = \langle f^{in}, \mu^{out}, \delta_s\rangle$ then

- if $a \in A^{in}$ with $f^{in}(a) = \mu_a$ we write $s \xrightarrow{a}\!\!\rightsquigarrow \mu_a$, furthermore, if $s' \in spt(\mu_a)$ with $\mu_a(s') = p$ we write $s \overset{a[p]}{\rightsquigarrow} s'$.
- if $\mu^{out} \neq *$ we write $s \rightsquigarrow \mu^{out}$ and if $\mu^{out}(a, s') = p > 0$ we write $s \overset{a[p]}{\rightsquigarrow} s'$.



transitions from a state in an I/O probabilistic automaton
$A^{in} = \{a, b\},\ A^{out} = \{c, d\}$

In an I/O automaton for every input action there is a reactive transition. Note that f^{in} is always a function and not a partial function as in the reactive model. Hence each input action is enabled in each state of an I/O probabilistic automaton. The output actions are treated generatively. At most one generative probabilistic transition gives the output behavior of each state. The delay rate parameter δ_s is an aspect from continuous-time systems, and its meaning will become clear in section 4 when we discuss compositions of I/O automata.

The I/O automata will not be compared and placed in the hierarchy of section 5 since they involve a continuous element. It is obvious that, when ignoring the 0 delays, for $A^{out} = \emptyset$ one gets the reactive model (with all actions enabled) and for $A^{in} = \emptyset$ one gets the generative model with a delay rate assigned to each state. A connection exists between I/O automata and some models with structured transition relation (section 3.3). Combined systems similar to I/O automata appear as models of process terms in the process algebra EMPA [14, 15].

Since we do not compare I/O automata in Section 4, we do not need a notion of bisimulation for them, although it can be defined by combining the transfer conditions for reactive and generative bisimulation, and taking care of the delay rate. In [87] no notion of bisimulation is introduced, instead a different notion of behavior of I/O automata is considered. A definition of bisumulation for I/O automata can be found in [75].

3.2 Automata with Distinction Between States

So far we have seen some types of automata that allow modelling of probabilistic behavior, but none of those has the capability of also modelling full non-determinism. The types of systems introduced in a minute allow full non-determinism while making a distinction between probabilistic states with outgoing probabilistic transitions, and non-deterministic states with action labelled transitions.

Stratified Probabilistic Automata. The simplest system with a distinction on states appears under the name of stratified probabilistic automaton, and is discussed in [39, 40, 79, 49]. Stratified automata do not yet allow any form of non-determinism although there is a distinction on states.

Definition 14. *A stratified probabilistic automaton is a triple* $\langle S, A, \alpha \rangle$ *where the transition function* α *is given by*

$$\alpha : S \to \mathcal{D}(S) + (A \times S) + 1$$

The class of all stratified automata we denote by **Str**. Due to the disjoint union in the codomain of the transition function, there are three types of states in a stratified automaton: probabilistic states consisting of $s \in S$ such that $\alpha(s) \in \mathcal{D}(S)$, deterministic states $s \in S$ for which $\alpha(s) = \langle a, s' \rangle$ allowing a single action labelled transition and terminating states $s \in S$ with $\alpha(s) = *$.

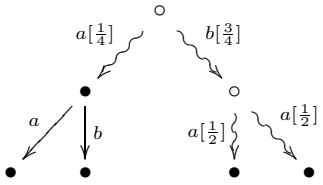
Definition 15. *An equivalence relation R on S is a bisimulation on the stratified probabilistic automaton $\langle S, A, \alpha \rangle$ if for all $\langle s, t \rangle \in R$:*

1. *if $s \leadsto \mu$ then there exists a distribution μ' with $t \leadsto \mu'$ and $\mu \equiv_R \mu'$;*
2. *if $s \xrightarrow{a} s'$ then there exists t' such that $t \xrightarrow{a} t'$ and $\langle s', t' \rangle \in R$.*

Vardi Probabilistic Automata. One of the earliest models of probabilistic automata was introduced by Vardi in [84] under the name *concurrent Markov chains*. The original definition of a concurrent Markov chain was given in terms of state labelled transition systems, for purposes of verification of logical properties. Therefore we slightly modify the definition, calling this class of automata Vardi probabilistic automata.

Definition 16. *A Vardi probabilistic automaton is a triple $\langle S, A, \alpha \rangle$ where the transition function α is given by*

$$\alpha : S \to \mathcal{D}(A \times S) \cup \mathcal{P}(A \times S)$$



Vardi probabilistic automaton

Remark 3. Note that $\cup$ is used in Definition 16 rather than $+$. One could consider the union disjoint, but it is of more use to identify $\mu^1_{\langle a,s' \rangle}$ with the singleton $\{\langle a, s' \rangle\}$, i.e. a state with a transition $s \overset{a[1]}{\rightsquigarrow} s'$ can be identified with a state allowing only one transition $s \xrightarrow{a} s'$.

In Vardi automata, the probabilistic states are of a generative kind, while the other states are non-deterministic with full non-determinism, as in an LTS. Therefore, the definition of bisimulation is a combination of Definition 8 and Definition 12.

Definition 17. *An equivalence relation R on S is a bisimulation on the Vardi probabilistic automaton $\langle S, A, \alpha \rangle$ if for all $\langle s, t \rangle \in R$:*

1. *if $s \leadsto \mu$ then there exists a distribution μ' with $t \leadsto \mu'$ and $\mu \equiv_{R,A} \mu'$;*
2. *if $s \xrightarrow{a} s'$ then there exists t' such that $t \xrightarrow{a} t'$ and $\langle s', t' \rangle \in R$.*

Remark 4. We note that in the literature, in particular in [84], there is no definition of bisimulation. However, the current understanding of probabilistic bisimulation, and the concept of a general coalgebraic definition of bisimulation allows us to state the previous definition.

We denote the class of Vardi probabilistic automata by **Var**.

The Alternating Models of Hansson. Another model that treats separately (purely) probabilistic and non-deterministic states is the alternating model introduced by Hansson, see for example [41, 46]. We present the class of alternating probabilistic automata **Alt**, its subclass of strictly alternating probabilistic automata **SA** and, in turn, two subclasses of **SA**, denoted by $\mathbf{SA}_n$ and $\mathbf{SA}_p$.

Definition 18. *An alternating probabilistic automaton is a triple* $\langle S, A, \alpha \rangle$ *where*

$$\alpha : S \to \mathcal{D}(S) + \mathcal{P}(A \times S).$$

The class of alternating automata is denoted by **Alt**. *Denote by* N *and* P *the subsets of* S *containing non-deterministic and probabilistic states, respectively.*

A strictly alternating automaton is an alternating automaton where for all $s \in S$ *the following holds:*

1. *if* $s \in P$ *with* $\alpha(s) = \mu \in \mathcal{D}(S)$ *then* $spt(\mu) \subseteq N$;
2. *if* $s \in N$ *then for all* $\langle a, s' \rangle \in \alpha(s)$, $s' \in P$.

The class of all strictly alternating automata is denoted by **SA**.

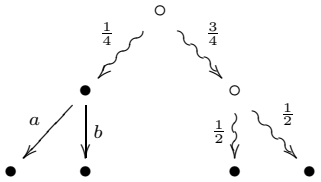
An automaton of **SA** *belongs to* $\mathbf{SA}_n$ *if and only if*

$$\forall s \in S\colon (\forall s' \in S, \forall a \in A, \forall p \in [0,1]\colon s' \overset{p}{\not\rightsquigarrow} s \wedge s' \overset{a}{\not\to} s) \Rightarrow s \in N. \qquad (1)$$

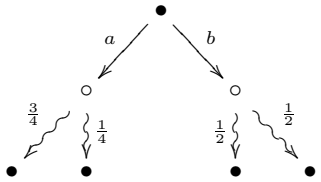
An automaton of **SA** *belongs to* $\mathbf{SA}_p$ *if and only if*

$$\forall s \in S\colon (\forall s' \in S, \forall a \in A, \forall p \in [0,1]\colon s' \overset{p}{\not\rightsquigarrow} s \wedge s' \overset{a}{\not\to} s) \Rightarrow s \in P. \qquad (2)$$

The well known of these classes are the class **SA** [41, 46] and the class $\mathbf{SA}_n$ [5, 6], but we have chosen for presenting all these classes structurally. The class **Alt** is a slight generalization of the class **SA** and is very much similar to the stratified and Vardi models. Therefore it deserves its place in this overview. In an alternating automaton only a distinction on states is imposed. In the strictly alternating model it is required that all successors of a non-deterministic state are probabilistic states and vice versa. Furthermore, the two subclasses $\mathbf{SA}_n$ and $\mathbf{SA}_p$ take care that any "initial state" is non-deterministic (1) and probabilistic (2), respectively. We define the subclasses $\mathbf{SA}_n$ and $\mathbf{SA}_p$ in order to make a precise comparison of the class **SA** with some of the other models (section 5).



alternating probabilistic automaton



strictly alternating automaton ($\mathbf{SA}_n$)

One definition of bisimulation fits all the introduced classes of alternating automata, where the transfer conditions are exactly the same as for the stratified model, given in Definition 15.

3.3 Probabilistic Automata with Structured Transition Function

In this subsection we focus on three types of probabilistic automata that provide orthogonal coexistence of full non-determinism and probabilities without distinguishing between states.

Segala and Simple Segala Probabilistic Automata. Two types of probabilistic automata were introduced by Segala and Lynch in [78, 76]. We call them Segala probabilistic automata and simple Segala probabilistic automata. An extensive overview of the simple Segala model is given in [80, 81] and they have been used for verification purposes and developing theoretical results in several situations as reported in [82, 83, 24, 8, 13, 21, 20, 55, 56].

Definition 19. *A Segala probabilistic automaton is a triple $\langle S, A, \alpha \rangle$ where*

$$\alpha : S \to \mathcal{P}(\mathcal{D}(A \times S))$$

If $s \in S$ such that $\mu \in \alpha(s)$ we write $s \rightarrow\!\!\!\!\sim\ \mu$, and if $\langle a, s' \rangle \in spt(\mu)$ with $\mu(a, s') = p$ then we write $s \stackrel{a[p]}{\rightarrow\!\!\!\!\sim} s'$.

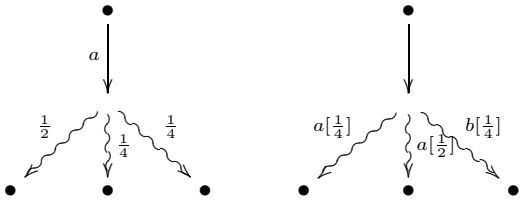
A simple Segala probabilistic automaton[1] *is a triple $\langle S, A, \alpha \rangle$ for a transition function*

$$\alpha : S \to \mathcal{P}(A \times \mathcal{D}(S))$$

If $s \in S$ with $\langle a, \mu \rangle \in \alpha(s)$ then we write $s \stackrel{a}{\rightarrow\!\!\!\!\sim}\ \mu$, and if $s' \in spt(\mu)$ we write $s \stackrel{a}{\rightarrow\!\!\!\!\sim}\ ^{p} s'$.

The simple Segala type of systems arise from **NA** by changing the target state with a distribution over possible target states. A transition in a simple Segala automaton and in a Segala automaton is shown in the next figure.

[1] Segala and Lynch call these models probabilistic automaton (PA) and simple probabilistic automaton, while Stoelinga calls them general PA and PA, respectively.



simple Segala transition Segala transition

There can be more then one transition available in a state and that is where non-determinism occurs. Hence, the non-deterministic choices exist between transitions, while the probabilities are specified within a transition. In the original definition by Segala and Lynch distributions over an extended set $A \times S + 1$ (or over $S + 1$ in the simple case) were treated i.e. substochastic distributions, where the probability assigned to the special symbol $*$ was interpreted as the deadlock probability. We choose not to include this in the definition for two reasons: it disturbs the comparison (Section 4) since the other models do not include substochastic distributions, and it can be imitated by adding an extra deadlock state to a system.

We denote the class of Segala probabilistic automata by **Seg** and the class of simple Segala automata by **SSeg**.

The simple Segala automaton is a generalization towards full non-determinism of the reactive model and of the purely probabilistic automata of Rabin [73]. A deterministic version of the simple Segala automaton equivalent to the reactive model is known as *Markov decision process* ([34]), while the name *probabilistic transition system* is used for this model in [52] and for a state labelled version in [36, 37]. A comparison of $\mathbf{SA}_n$ and the simple Segala model can be found in [25].

Bisimulation for the simple Segala systems is defined with the same transfer conditions as for reactive systems given in Definition 11, while for the Segala systems the transfer conditions for bisimulation of Definition 12 for generative systems apply, when changing $\rightsquigarrow$ to $\rightarrow\rightsquigarrow$.

A great novelty introduced with both types of Segala systems was the definition of a stronger probabilistic bisimulation relation that identifies states that have matching "combined transitions". For more information on this topic the interested reader is referred to [78, 76, 80, 81, 24].

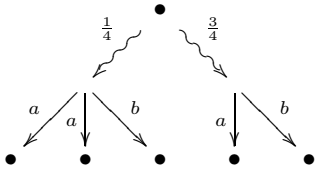
Bundle Probabilistic Automata. Another way to include both non-determinism and probability is to consider distributions over sets of transitions as in the *bundle* model, introduced in [35]. (Recall that Segala systems have sets of distributions over transitions.)

Definition 20. *A bundle probabilistic automaton is a triple* $\langle S, A, \alpha \rangle$ *where*

$$\alpha : S \to \mathcal{D}(\mathcal{P}(A \times S)) + 1$$

When $s \in S$ *and* $\alpha(s) = \mu$ *we write* $s \rightsquigarrow \mu$*, furthermore, if* $T \subseteq A \times S$, $\mu(T) = p > 0$ *we write* $s \overset{p}{\rightsquigarrow} T$ *and if* $\langle a, t \rangle \in T$ *then* $s \overset{p}{\rightsquigarrow} \overset{a}{\rightarrow} t$.

The bundle model can be considered as generative, since probabilities are also distributed over actions. Therefore the bundle model offers a solution to the absence of non-determinism in the generative setting. Note that the original definition is even slightly more general, namely the codomain of the transition function is $\mathcal{D}(\mathcal{M}(A \times S))$ where $\mathcal{M}(X)$ denotes all the (finite) multi-subsets of a set X. Hence it is possible to have multiple transitions from one state to another with the same action within one bundle. Since it is not essential for the material presented here, we will not add multi-sets in the bundle model. The class of bundle probabilistic automata is denoted by **Bun**. A typical bundle probabilistic automaton is depicted below:



bundle probabilistic automaton

In the literature, in particular in [35], there is no definition of bisimulation on bundle probabilistic automata, instead they are transformed to generative systems and then compared with generative bisimulation. We give here a definition of bisimulation for the bundle probabilistic automata that is deduced from the general coalgebraic definition of bisimulation (cf. [53, 74, 68]). A justification for doing so is that all previously stated definitions of bisimulation which were based on the probabilistic bisimulation of Larsen and Skou [61] coincide with the general coalgebraic definition of bisimulation for the particular type of systems. In the non-probabilistic case this coincidence is well known (see e.g. [74]). For Markov chains it was proven in [86], for the Segala probabilistic automata in [26] and the same proof technique extends to all other cases.

Prior to stating the definition we need a way to lift a relation on a set S to a relation on the set $\mathcal{P}(A \times S)$.

Definition 21. *Let R be a relation on S and let $X,\ Y \in \mathcal{P}(A \times S)$. Define $X \equiv_{R,\mathcal{P}} Y$ if and only if for all $a \in A$:*

1. *if $\langle a, x\rangle \in X$ then there exists $\langle a, y\rangle \in Y$ with $\langle x, y\rangle \in R$;*
2. *if $\langle a, y\rangle \in Y$ then there exists $\langle a, x\rangle \in X$ with $\langle x, y\rangle \in R$.*

It holds that, if R is an equivalence on S, then $\equiv_{R,\mathcal{P}}$ is an equivalence on $\mathcal{P}(A \times S)$.

Definition 22. *An equivalence relation R is a bisimulation on the state set of a bundle probabilistic automaton $\langle S, A, \alpha\rangle$ if for all $\langle s, t\rangle \in R$ it holds*

$$\text{if } s \rightsquigarrow \mu \text{ then there exists } \mu' \text{ such that } t \rightsquigarrow \mu' \text{ and } \mu \equiv_{\equiv_{R,\mathcal{P}}} \mu'$$

where $\equiv_{\equiv_{R,\mathcal{P}}}$ denotes the lifting of the relation $\equiv_{R,\mathcal{P}}$ to distributions on $\mathcal{P}(A \times S)$ as defined by Definition 1.

3.4 Complex Models – Pnueli-Zuck and General Probabilistic Automata

An early model including probabilities and a structured transition relation was proposed by Pnueli and Zuck [71, 72] under the name *finite-state probabilistic programs* and later used in [7]. We call this type of automata Pnueli-Zuck probabilistic automata, and denote the class of all such by $\mathbf{PZ}$[2]. The model of Pnueli and Zuck has the most complex transition function, it adds one more power set to the bundle model and so allows two types of non-determinism, both between the probabilistic transitions and inside the transitions. However, in order to get a top element for our hierarchy (section 5) we expand the model a bit further and define one most general type of probabilistic automata. The class of such will be denoted by $\mathbf{MG}$.

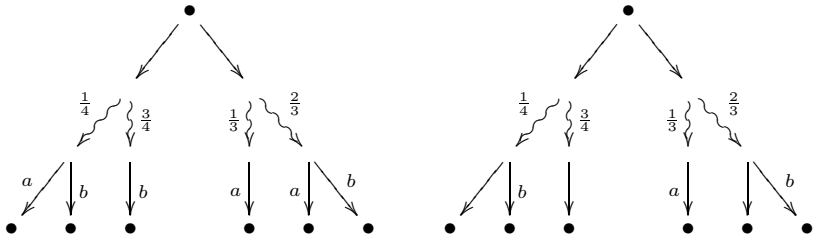
Definition 23. *A Pnueli-Zuck automaton is a triple* $\langle S, A, \alpha \rangle$ *where*

$$\alpha : S \to \mathcal{P}(\mathcal{D}(\mathcal{P}(A \times S)))$$

When $s \in S$ *and* $\mu \in \alpha(s)$ *we write* $s \to\!\!\!\!\rightsquigarrow \mu$*, further on, if* $T \subseteq A \times S$*,* $\mu(T) = p > 0$ *we write* $s \stackrel{p}{\rightsquigarrow} T$ *and if* $\langle a, t \rangle \in T$ *then* $s \stackrel{p}{\rightsquigarrow} \stackrel{a}{\to} t$*. A general probabilistic automaton is a triple* $\langle S, A, \alpha \rangle$ *where*

$$\alpha : S \to \mathcal{P}(\mathcal{D}(\mathcal{P}(A \times S + S)))$$

The notation for Pnueli-Zuck automata is also used for general automata. Furthermore, if $s \in S, \mu \in \alpha(s), T \subseteq A \times S + S$ *with* $\mu(T) = p > 0$ *and* $t \in T$*, then we write* $s \stackrel{p}{\rightsquigarrow} \to t$*.*



Pnueli-Zuck system most general system

The unlabelled transitions which appear in the right figure (most general system) correspond to pure probabilistic transitions in Markov chains or alternating systems, where a change of state can happen with certain probability without performing an action.

As for bundle systems, there is no notion of bisimulation for Pnueli-Zuck systems in the literature. A bisimulation definition can be formulated out of the

[2] Like Vardi's model, these automata appear in the literature in a state labelled version for model checking purposes. Therefore we change the definition towards transition labels.

general coalgebraic definition, and it leads the same transfer conditions as in Definition 22 when changing $\rightsquigarrow$ to $\rightarrow\rightsquigarrow$. A small modification is needed for the general probabilistic automata.

4 Composing Probabilistic Systems in Parallel

Having introduced the probabilistic models, we consider possible definitions of the parallel composition operator for these extended systems. Lots of results on this topic exist in the literature. For a broad overview the reader is referred to [35, 9]. An overview of probabilistic process algebras covering other probabilistic operators as well is presented in [58].

For the classes **MC** and **IO** there is a unique parallel composition defined. In **MC** this operation is purely synchronous given by the product of distributions (cf. Section 2.3), whereas in **IO** the definition of the parallel composition operation strongly relies on the specific structure of the systems (cf. Section 4.3 below). For all other classes it is meaningful to consider various definitions of parallel composition. Such operations might be synchronous or asynchronous in nature and moreover might be based upon the styles CCS, CSP and ACP described in Section 2. The style CSP plays a special role in this respect since it is by its definition partly synchronous and partly asynchronous and hence gives rise to a somehow mixed variant of parallel composition.

The classes of (probabilistic) systems can be divided into three groups dependent on whether they show reactive, generative or alternating behavior. Classes belonging to the same of these groups allow in essence similar definition and investigation of parallel composition.

Instead of going through all, obviously quite numerous, variants of parallel composition for each single class of systems, we shall in the subsequent sections 4.1, 4.2 and 4.4. discuss a couple of instructive cases in some detail. However, let us give a complete scheme of possible (and/or already studied) definitions of parallel composition operator by means of a comprehensive table. In the table below each column is dedicated to one class of probabilistic automata, and each row to one of the introduced styles of parallel composition. In the intersecting cells a symbol representing the definability status of the corresponding parallel composition operator in the corresponding class is placed. Neighboring cells containing the same symbol within one column are merged. We use the following symbols:

+ : defined in the literature or straightforward
+: definable but not carried out
−: not definable
p: defined in the literature with parameters
p: parameterized version definable, but not carried out
n: normalized version definable, but not carried out
+/n: "+" for total communication function, "n" otherwise

Table 1 presents the overall situation concerning definability of parallel composition on probabilistic automata. A brief analysis of these summary results shows that allowing full non-determinism enables definition of any type of parallel composition.

Table 1. Definability of $\|$

		- - - reactive - - -				- - - - - generative - - - - -					- - - alternating - - -		
		DA	**NA**	**React**	**SSeg**	**Gen**	**Seg**	**Bun**	**PZ**	**MG**	**Var**	**Alt, SA, ..**	**Str**
sync	CCS	−		−		n	n				n		
	CSP	**+**	**+**	**+**	**+**	n	n	**+**	+	+	n	**+**	**+**
	ACP	−		−		**+**/n	+/n				+/n		
	CSP	−	**+**	−	**+**	**p**	p	**+**	+	+	p	**+**	−
async	CCS												
	CSP	−	**+**	−	**+**	**p**	p	**+**	+	+	p	**+**	−
	ACP												

4.1 Parallel Composition in the Reactive Setting

Systems with reactive behavior are systems in the classes **React** and **SSeg**, as well as **NA** and **DA** in the non-probabilistic case. Any parallel composition operator on LTS (Section 2.3.) nicely extends to the class **SSeg**. Let $\mathcal{A}_1 = \langle S_1, A, \alpha_1 \rangle, \mathcal{A}_2 = \langle S_2, A, \alpha_2 \rangle$ be in **SSeg**. Then $\mathcal{A}_1 \| \mathcal{A}_2 = \langle S_1 \times S_2, A, \alpha \rangle$ where α is defined as follows:

[*CCS style*]: $s \| t \xrightarrow{a} \!\!\rightsquigarrow \mu$ if and only if

1. $a = \tau$, $s \xrightarrow{b} \!\!\rightsquigarrow \mu_1$, $t \xrightarrow{\bar{b}} \!\!\rightsquigarrow \mu_2$ and $\mu = \mu_1 \times \mu_2$, or
2. $s \xrightarrow{a} \!\!\rightsquigarrow \mu_1$ and $\mu = \mu_1 \times \mu_t^1$, or
3. $t \xrightarrow{a} \!\!\rightsquigarrow \mu_2$ and $\mu = \mu_s^1 \times \mu_2$.

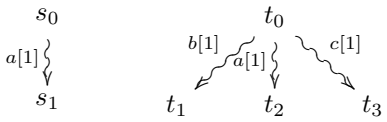
[*CSP style*]: $s \|_L t \xrightarrow{a} \!\!\rightsquigarrow \mu$ if and only if

1. $a \in L$, $s \xrightarrow{a} \!\!\rightsquigarrow \mu_1$, $t \xrightarrow{a} \!\!\rightsquigarrow \mu_2$ and $\mu = \mu_1 \times \mu_2$, or
2. $a \notin L$, $s \xrightarrow{a} \!\!\rightsquigarrow \mu_1$ and $\mu = \mu_1 \times \mu_t^1$, or
3. $a \notin L$, $t \xrightarrow{a} \!\!\rightsquigarrow \mu_2$ and $\mu = \mu_s^1 \times \mu_2$.

[*ACP style*]: $s \| t \xrightarrow{a} \!\!\rightsquigarrow \mu$ if and only if

1. $a = bc$ defined, $s \xrightarrow{b} \!\!\rightsquigarrow \mu_1$, $t \xrightarrow{c} \!\!\rightsquigarrow \mu_2$ and $\mu = \mu_1 \times \mu_2$, or
2. $s \xrightarrow{a} \!\!\rightsquigarrow \mu_1$ and $\mu = \mu_1 \times \mu_t^1$, or
3. $t \xrightarrow{a} \!\!\rightsquigarrow \mu_2$ and $\mu = \mu_s^1 \times \mu_2$.

The definition of any of these operators is problematic for the class **React**. For $\mathcal{A}_1, \mathcal{A}_2 \in$ **React** it might happen that $\mathcal{A}_1 \| \mathcal{A}_2 \notin$ **React** in any variant of parallel composition. Even in the synchronous CCS style, multiple transitions labelled with τ may appear. In the CSP style, 2. and 3. may introduce internal non-determinism. However, if L contains all the common actions of $\mathcal{A}_1$ and $\mathcal{A}_2$, then this problem disappears. In case of ACP all of 1., 2. and 3. introduce internal non-determinism, hence **React** is not closed under this operator for any arbitrary communication function γ[3]. For example, if $ab = ac = a$ then the ACP parallel product of the following two automata



is not defined in **React**, since the definition yields: $s_0 \| t_0 \xrightarrow{a} \rightsquigarrow \mu_x^1$ for $x \in \{s_1 \| t_0, s_0 \| t_2, s_1 \| t_1, s_1 \| t_3\}$ i.e. more than one transition corresponds to the action a, which is prohibited in **React**.

An asynchronous parallel composition in CCS style on simple Segala systems was defined in [21], a synchronous parallel composition in CCS style on reactive systems was defined in [40, 39, 52], the last reference working with simple Segala systems. A synchronous CSP style parallel composition is defined for reactive systems in [55, 69], while an asynchronous CSP style parallel composition with synchronization on common actions is used in [78, 76, 80] for simple Segala systems.

4.2 Parallel Composition in the Generative Setting

Systems with generative behavior belong to the classes **Gen**, **Var**, **Seg**, **Bun**, **PZ** and **MG**. The Vardi systems express also alternating behavior and they will be discussed with the alternating systems. A common property of the generative systems is that always probability distributions over actions and states appear. This leads to difficulties in defining parallel composition operators (see [41, 30, 76, 35]), especially in the asynchronous case. Namely, a generative type system defines in each state a probability distribution over a set of enabled actions, offered by the environment. When two such systems are composed in parallel it is not clear how the common set of enabled actions should be defined, nor how the two probability distributions should be composed into one (cf. [52]). In this section we explain several approaches for solving this problem.

Let $\mathcal{A}_1 = \langle S_1, A, \alpha_1 \rangle, \mathcal{A}_2 = \langle S_2, A, \alpha_2 \rangle$ be two generative systems. Their parallel composition in all cases will be denoted by $\mathcal{A}_1 \| \mathcal{A}_2 = \langle S_1 \times S_2, A, \alpha \rangle$, possibly with parameters.

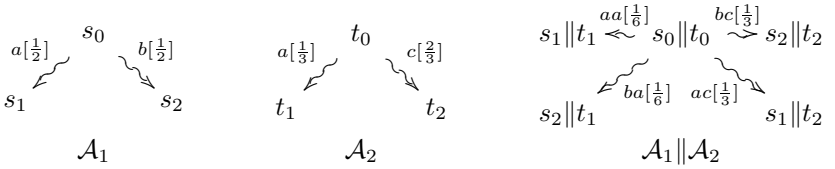
[3] The same problems arise in the class **DA** opposed to **NA**, namely parallel composition introduces internal non-determinism, and therefore **DA** is not closed under $\|$.

Synchronous CCS, CSP, ACP style parallel composition can be defined on generative systems, as done in [40, 39, 35] by:

$$s \overset{a[p]}{\rightsquigarrow} s', t \overset{b[q]}{\rightsquigarrow} t' \iff s\|t \overset{ab[pq]}{\rightsquigarrow} s'\|t'$$

where the set of actions is assumed to form a commutative semigroup (PCCS, [38]) and ab stands for the product of a and b in $A(\cdot)$.

The following figure presents an example of synchronous parallel composition of two generative systems.



In order to capture possible asynchronous behavior, several parallel composition operators were defined in the literature that use *bias factors*. In most of the cases the composition is not symmetric. Namely, the main problem in defining asynchronous parallel composition is that any definition introduces non-determinism. In the proposed solutions, these non-deterministic choices are changed to probabilistic ones, by specifying parameters of the parallel composition.

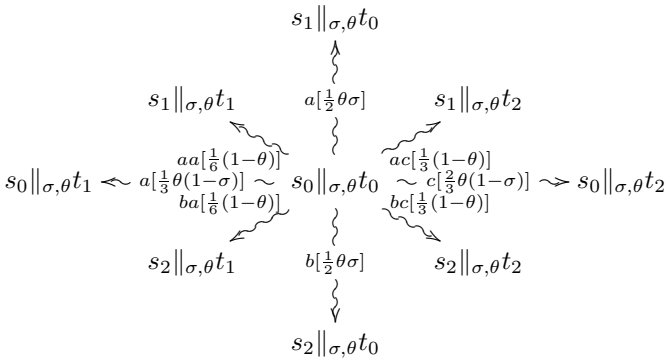
An ACP style parallel composition operator for generative systems was defined in [10]. The definition follows the non-probabilistic definition of the ACP parallel operator, while changing the non-deterministic choices introduced by interleaving and/or communication into corresponding probabilistic choices. The operator is parameterized with two parameters σ and θ, denoted by $\mathcal{A}_1\|_{\sigma,\theta}\mathcal{A}_2$. In the product state $s\|_{\sigma,\theta}t$, synchronization between s and t can occur with probability $1-\theta$ and an autonomous action of either s or t with probability θ. Furthermore, given that an autonomous move occurs, then it comes from s with probability σ and from t with probability $1-\sigma$. For definability of $\mathcal{A}_1\|_{\sigma,\theta}\mathcal{A}_2$ it is necessary that the communication function is a total function.

We define $s\|_{\sigma,\theta}t \overset{a[P]}{\rightsquigarrow} s'\|_{\sigma,\theta}t'$ if and only if

1. $s \overset{b[p]}{\rightsquigarrow} s'$ and $t \overset{c[q]}{\rightsquigarrow} t'$, $bc = a$ and $P = (1-\theta)pq$, or
2. $s \overset{a[p]}{\rightsquigarrow} s'$, $t = t'$, and $P = p\theta\sigma$, or
3. $t \overset{a[q]}{\rightsquigarrow} t'$, $s = s'$ and $P = q\theta(1-\sigma)$, or
4. $s \overset{a[p]}{\rightsquigarrow} s'$, $t\not\rightsquigarrow$, $t = t'$ and $P = p$, or
5. $s\not\rightsquigarrow$, $t \overset{a[q]}{\rightsquigarrow} t'$, $s = s'$ and $P = q$.

Note that by this definition we might get two transitions $s\|_{\sigma,\theta}t \overset{a[p_1]}{\rightsquigarrow} s'\|_{\sigma,\theta}t'$ and $s\|_{\sigma,\theta}t \overset{a[p_2]}{\rightsquigarrow} s'\|_{\sigma,\theta}t'$, which then are replaced by one transition $s\|_{\sigma,\theta}t \overset{a[p_1+p_2]}{\rightsquigarrow} s'\|_{\sigma,\theta}t'$.

For $\mathcal{A}_1$ and $\mathcal{A}_2$ as in the previous figure, we get $\mathcal{A}_1 \|_{\sigma,\theta} \mathcal{A}_2$ which looks like:



Another two biased, parameterized, parallel composition operators are defined in [35], one asynchronous CCS style operator, denoted by $\mathcal{A}_1^{\theta} \|^{\sigma} \mathcal{A}_2$ and one CSP-style operator, denoted by $\mathcal{A}_1 \|_L^{\sigma} \mathcal{A}_2$. Denote by $s \overset{a[p]}{\rightsquigarrow}$ the clause $(\exists s')s \overset{a[p]}{\rightsquigarrow} s'$. We present the definition of $\mathcal{A}_1 \|_L^{\sigma} \mathcal{A}_2$ first:

$s\|_L^{\sigma} t \overset{a[P]}{\rightsquigarrow} s'\|_L^{\sigma} t'$ if and only if one of the following is satisfied:

1. $s \overset{a[p]}{\rightsquigarrow} s'$, $t \overset{b[q]}{\rightsquigarrow}$, $a, b \notin L$, $t' = t$ and $P = \frac{pq\sigma}{\nu(s,t,L)}$;
2. $s \overset{b[p]}{\rightsquigarrow}$, $t \overset{a[q]}{\rightsquigarrow} t'$, $a, b \notin L$, $s' = s$ and $P = \frac{pq(1-\sigma)}{\nu(s,t,L)}$;
3. $s \overset{a[p]}{\rightsquigarrow} s'$, $t \overset{b[q]}{\rightsquigarrow}$, $a \notin L, b \in L$, $t' = t$ and $P = \frac{pq}{\nu(s,t,L)}$;
4. $s \overset{b[p]}{\rightsquigarrow}$, $t \overset{a[q]}{\rightsquigarrow} t'$, $a \notin L, b \in L$, $s' = s$ and $P = \frac{pq}{\nu(s,t,L)}$;
5. $s \overset{a[p]}{\rightsquigarrow} s'$, $t \not\rightsquigarrow$, $a \notin L$, $t' = t$ and $P = \frac{p}{\nu'(s,L)}$;
6. $s \not\rightsquigarrow$, $t \overset{a[q]}{\rightsquigarrow} t'$, $a \notin L$, $s' = s$ and $P = \frac{q}{\nu'(t,L)}$;
7. $s \overset{a[p]}{\rightsquigarrow} s'$, $t \overset{a[q]}{\rightsquigarrow} t'$, $a \in L$ and $P = \frac{pq}{\nu(s,t,L)}$.

Where the normalization factors are calculated by

$$\nu'(s,L) = 1 - \sum_{s \overset{a[p]}{\rightsquigarrow},\ a \in L} p,\ \nu(s,t,L) = 1 - \sum_{s \overset{a[p]}{\rightsquigarrow},\ t \overset{b[q]}{\rightsquigarrow},\ a,b \in L,\ a \neq b} pq.$$

For this CSP style operator only one parameter is needed since the only non-determinism occurs if both systems autonomously decide to perform actions not in the synchronizing set L. In $s\|_L^{\sigma} t$, the parameter σ denotes the probability that s performs an autonomous action, given that both s and t have decided not to synchronize. Furthermore, normalization factors are used to determine the actual probability of every transition. These normalization factors redistribute the probability mass that is due to autonomous decisions of both processes that would otherwise lead to deadlock.

For the asynchronous CCS parallel composition $\mathcal{A}_1^{\theta} \|^{\sigma} \mathcal{A}_2$ the interpretation of the probabilistic parameters $\theta, \sigma \in (0,1)$ is similar to the ACP approach. They

provide the relevant information that an adversary needs in order to resolve nondeterminism that arises when composing two systems. In $s^{\theta}\|^{\sigma}t$, σ denotes the probability that s performs an autonomous action given that both s and t do not want to synchronize, and θ denotes the probability that some autonomous action occurs, given that synchronization is possible. Hence, if synchronization is possible, it will take place with probability $1-\theta$.

The earliest biased parallel composition operator for generative systems was defined in [30] and treated in detail in [58]. There the parallel composition $\mathcal{A}_1\|_{\rho}\mathcal{A}_2 = \langle S_1 \times S_2, A, \alpha\rangle$ uses one bias parameter ρ. A state $s\|_{\rho}t$ in the composed automaton can either do an action a with a certain probability if both s and t in their components can do an action a (CSP style), or can do a τ action if either s or t can do a τ action. However, whether a τ action from s or from t is chosen is biased with the bias factor ρ. The probability of the synchronous execution of a is calculated via a normalization function $\nu : S_1 \times S_2 \to [0,1]$. Basically, $\nu(s,t)$ sums up the probabilities of the possible outgoing transitions of the new state which would be obtained if asynchronous behavior introduced non-determinism. Then $\nu(s,t)$ is used to calculate the actual (conditional) probabilities of the distribution assigned to $s\|_{\rho}t$.

Finally, a completely different solution of the problem of defining a parallel composition operator in the generative setting is provided in [35], the introduction of the class of bundle systems **Bun**. The bundle systems possess nondeterminism, which allows for an elegant definition of any asynchronous parallel composition operator, as follows.

Let $\mathcal{A}_1 = \langle\, S_1, A, \alpha_1 \,\rangle, \mathcal{A}_2 = \langle\, S_2, A, \alpha_2 \,\rangle \in \mathbf{Bun}$. Then $\mathcal{A}_1\|\mathcal{A}_2 = \langle\, S_1 \times S_2, A, \alpha \,\rangle$ where, for $s \in S_1, t \in S_2$,

$$s\|t \rightsquigarrow \mu = P(\mu_1,\mu_2) \iff s \rightsquigarrow \mu_1, t \rightsquigarrow \mu_2$$

and $P(\mu_1,\mu_2)$ denotes a specific product of distributions, defined as follows: For $\mu_1 \in \mathcal{D}(\mathcal{P}(A \times S_1)), \mu_2 \in \mathcal{D}(\mathcal{P}(A \times S_2))$, $\mu = P(\mu_1,\mu_2) \in \mathcal{D}(\mathcal{P}(A \times (S_1 \times S_2)))$ where for all $B_s \in spt(\mu_1), B_t \in spt(\mu_2)$, $\mu(B_s \otimes B_t) = \mu_1(B_s) \cdot \mu_2(B_t)$ and

$$\begin{aligned} &B_s \otimes B_t = \\ &\{\langle\, a, \langle\, s', t \,\rangle\,\rangle \mid \langle\, a, s' \,\rangle \in B_s\} \cup \\ &\{\langle\, b, \langle\, s, t' \,\rangle\,\rangle \mid \langle\, b, t' \,\rangle \in B_t\} \cup \\ &\{\langle\, ab, \langle\, s', t' \,\rangle\,\rangle \mid \langle\, a, s' \,\rangle \in B_s, \langle\, b, t' \,\rangle \in B_t\} \end{aligned}$$

Furthermore, $P(\mu,*) = P(*,\mu) = \mu$ and $P(*,*) = *$.

Note that the defined parallel composition for bundle systems is ACP style. By a slight modification of the definition of $\otimes$, all the other variants can be obtained. In a similar manner asynchronous parallel composition can be defined on the classes **PZ** and **MG**. In the literature there is no definition of a parallel composition operator for the class **Seg**.

4.3 Parallel Composition in the I/O Setting

A rather clean solution to the problems in the generative setting is given for the class of I/O automata in [87]. The view taken there is that the actions are divided

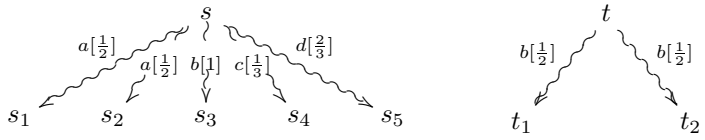
into input and output, and while there can be synchronization on input actions, as in the reactive setting, the sets of output actions in each of the components must be disjoint.

Let $\mathcal{A}_1 = \langle S_1, A_1, \alpha_1 \rangle, \mathcal{A}_2 = \langle S_2, A_2, \alpha_2 \rangle$ be two I/O automata. The automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are *compatible* if and only if $A_1^{out} \cap A_2^{out} = \emptyset$. Parallel composition is only defined on compatible automata. Let $A = A_1 \cup A_2$. We use the following convention: if s is a state in $\mathcal{A}_1$ ($\mathcal{A}_2$) and $a \in A \setminus A_1^{out}$ ($A \setminus A_2^{out}$), such that there is no transition from s involving a, then we consider that $s \overset{a[1]}{\rightsquigarrow} s$. This convention will enforce the "input always enabled" requirement for the composite automaton.

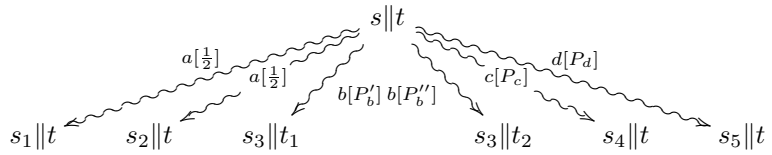
The parallel composition of $\mathcal{A}_1$ and $\mathcal{A}_2$ is the I/O automaton $\mathcal{A}_1 \| \mathcal{A}_2 = \langle S_1 \times S_2, A, \alpha \rangle$ where:

1. $A^{out} = A_1^{out} \cup A_2^{out}$,
2. $A^{in} = A \setminus A^{out} = (A_1^{in} \cup A_2^{in}) \setminus A^{out}$,
3. the transition function α is defined by the following:
   $s\|t \overset{a[P]}{\rightsquigarrow} s'\|t'$ if and only if one of the following holds
   a. $a \in A^{in}$, $s \overset{a[p]}{\rightsquigarrow} s'$, $t \overset{a[q]}{\rightsquigarrow} t'$ and $P = pq$;
   b. $a \in A_1^{out}$, $s \overset{a[p]}{\rightsquigarrow} s'$, $t \overset{a[q]}{\rightsquigarrow} t'$ and $P = \frac{\delta_1(s)}{\delta_1(s)+\delta_2(t)} pq$
   c. $a \in A_2^{out}$, $s \overset{a[p]}{\rightsquigarrow} s'$, $t \overset{a[q]}{\rightsquigarrow} t'$ and $P = \frac{\delta_2(t)}{\delta_1(s)+\delta_2(t)} pq$
   Hence, $\alpha(s\|t) = \langle f^{in}, \mu^{out}, \delta \rangle$ where $\delta(s\|t) = \delta_1(s) + \delta_2(t)$, $f^{in}(a) = \mu_a$ is determined by a., for $a \in A^{in}$, and μ^{out} is determined by b. and c., for $a \in A^{out}$.

Example 4. Let $\mathcal{A}_1 = \langle S_1, A_1, \alpha_1 \rangle, \mathcal{A}_2 = \langle S_2, A_2, \alpha_2 \rangle$ be two I/O automata, with $s \in S_1, t \in S_2$ and their corresponding transitions as in the following diagram.



Take $A_1^{in} \supseteq \{a, b\}$, $A_1^{out} \supseteq \{c, d\}$, $A_2^{in} \supseteq \emptyset$, $A_2^{out} \supseteq \{b\}$. Assume the automata are compatible i.e. $A_1^{out} \cap A_2^{out} = \emptyset$ (clearly the states s and t are compatible). Then $A^{out} \supseteq \{b, c, d\}$ and $A^{in} \supseteq \{a\}$. Due to the convention we consider that $t \overset{a[1]}{\rightsquigarrow} t$, $t \overset{c[1]}{\rightsquigarrow} t$ and $t \overset{d[1]}{\rightsquigarrow} t$. The transitions from $s\|t$ are then given with the following diagram.



For $P'_b = P''_b = \frac{\delta_2(t)}{\delta_1(s)+\delta_2(t)} \cdot \frac{1}{2}$, $P_c = \frac{\delta_1(s)}{\delta_1(s)+\delta_2(t)} \cdot \frac{1}{3}$ and $P_d = \frac{\delta_1(s)}{\delta_1(s)+\delta_2(t)} \cdot \frac{2}{3}$. Note that indeed $P'_b + P''_b + P_c + P_d = 1$.

The proof that $\mathcal{A}_1 \| \mathcal{A}_2$ is well defined in the class **IO** can be found in [87]. Let us now informally explain the definition of parallel composition, and the role of the functions δ_1, δ_2 and δ. If s is a state of $\mathcal{A}_1$, then $\delta_1(s)$ is a positive real number corresponding to the delay rate in state s. It is a rate of an exponential distribution, determining the time that the automaton waits in state s until it generates one of its output actions. If no output actions are enabled in this state then $\delta_1(s) = 0$. When determining the distribution on output actions for $s \| t$, denoted by $\mu^{out}_{s\|t}$, the components' distributions μ^{out}_s and μ^{out}_t are joined in one such that any probability of μ^{out}_s is multiplied with normalization factor $\frac{\delta_1(s)}{\delta_1(s)+\delta_2(t)}$ and any probability of μ^{out}_t is multiplied with $\frac{\delta_2(t)}{\delta_1(s)+\delta_2(t)}$. Note that by the compatibility assumption, no action appears both in the support of μ^{out}_s and in the support of μ^{out}_t. The normalization factor models a racing policy between the states s and t for generating their own output actions. The value $\frac{\delta_1(s)}{\delta_1(s)+\delta_2(t)}$ is the probability that the state s has less waiting time left then the state t and therefore wins the race and generates one of its own output actions. On the other hand, synchronization occurs on all input actions, no autonomous behavior is allowed by the components on input actions, corresponding to the assumption that the input is provided by the environment and must be enabled in any state.

4.4 Parallel Composition in the Alternating Setting

In this section we focus on the classes **Str**, **Alt**, **SA** (**SA**$_n$, **SA**$_p$) and **Var** that exhibit alternating behavior i.e. make a distinction between probabilistic and non-deterministic states. In [40, 39] and in [41] a rather elegant parallel composition for the classes **Str** and **SA**, respectively, is defined.

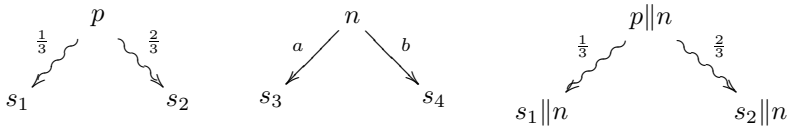
We present the definition for the class **Alt** and discuss that the same definition can be restricted to the classes **Str**, **Alt**, **SA** (**SA**$_n$, **SA**$_p$). Let $\mathcal{A}_1 = \langle S_1, A, \alpha_1 \rangle$ and $\mathcal{A}_2 = \langle S_2, A, \alpha_2 \rangle$ be two alternating automata, with $S_1 = N_1 + P_1$ and $S_2 = N_2 + P_2$. Their parallel composition is the alternating automaton $\mathcal{A}_1 \| \mathcal{A}_2 = \langle S, A, \alpha \rangle$ where $S = S_1 \times S_2 = N + P$ for $N = N_1 \times N_2$ and $P = P_1 \times P_2 + N_1 \times P_2 + P_1 \times N_2$ and the transition function is defined as follows. Let $p_1 \in P_1, p_2 \in P_2, n_1 \in N_1, n_2 \in N_2$ and $s_1 \in S_1, s_2 \in S_2$. For the probabilistic states in the composed automaton, we have:

$$\begin{array}{lll} p_1 \| p_1 \rightsquigarrow \mu & \iff & p_1 \rightsquigarrow \mu_1, p_2 \rightsquigarrow \mu_2, \mu = \mu_1 \times \mu_2 \\ p_1 \| n_2 \rightsquigarrow \mu & \iff & p_1 \rightsquigarrow \mu_1, \mu = \mu_1 \times \mu^1_{n_2} \\ n_1 \| p_2 \rightsquigarrow \mu & \iff & p_2 \rightsquigarrow \mu_2, \mu = \mu^1_{n_1} \times \mu_2. \end{array}$$

For the non-deterministic states in the composed automaton different variants (CCS, CSP or ACP style) can be chosen. We choose for the ACP style:
$n_1 \| n_2 \xrightarrow{a} s_1 \| s_2$ if and only if

1. $s \xrightarrow{b} s'$, $t \xrightarrow{c} t'$ and $bc = a$ defined, or
2. $s \xrightarrow{a} s'$, $t = t'$, or
3. $t \xrightarrow{a} t'$, $s = s'$.

Hence, when composing a probabilistic state with any other state the result is a probabilistic state. If the other state is non-deterministic, then the composed state basically behaves as the probabilistic state and leaves the second component of the state unchanged, as in the following example.



On the other hand, the composition of non-deterministic states is exactly the same as in the non-probabilistic case and therefore any possible parallel composition operator is definable here as in the case for LTSs.

By inspecting the definitions of the classes **SA**, $\mathbf{SA}_n$ and $\mathbf{SA}_p$ it is easy to see that the following statement is valid.

Proposition 3. *If* $\mathcal{A}_1, \mathcal{A}_2 \in \mathbf{SA}$ *(or* $\mathbf{SA}_n$*, or* $\mathbf{SA}_p$*), then* $\mathcal{A}_1 \| \mathcal{A}_2 \in \mathbf{SA}$ *(or* $\mathbf{SA}_n$*, or* $\mathbf{SA}_p$*), respectively.* □

The definition of parallel composition for stratified systems is given in [40, 39] with synchronous behavior when composing two (non-) deterministic states. This is necessary in order to stay in the class **Str** when composing two such automata, since in the stratified model there is only a single action transition possible from a (non-) deterministic state. A parallel composition operator with no synchronization but only interleaving, for the stratified class of systems, is defined in [49]. In the original definition for strictly alternating systems of [41], non-deterministic states are composed in the CCS fashion.

Complications arise in the case of **Var** models, due to their generative probabilistic behavior. The behavior of the composite states $n_1 \| n_2$, $n_1 \| p_2$ and $p_1 \| n_2$ can be defined in the same way as above. However, there is no convenient way to define $p_1 \| p_2$, since this coincides with defining generative parallel composition. Any of the approaches described in section 4.2 can be used.

5 Comparing Classes

This section is based on the results of [26]. There a hierarchy of probabilistic system types is presented in a coalgebraic setting. The coalgebraic theory proved useful in providing a uniform framework and shortening the proofs. However, the results can be restated and explained with the machinery introduced so far.

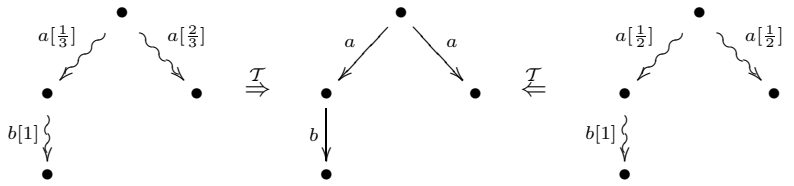
An Expressiveness Criterion

Let $\mathbf{C}_1$ and $\mathbf{C}_2$ be two classes of probabilistic automata. We say that the class $\mathbf{C}_1$ is *included* or *embedded* in the class $\mathbf{C}_2$, i.e. the class $\mathbf{C}_2$ is *at least as expressive as* the class $\mathbf{C}_1$ (notation $\mathbf{C}_1 \to \mathbf{C}_2$) if and only if there exists a translation function $\mathcal{T}$ that maps each automaton of the first class to an automaton of the second class such that bisimilarity is both reflected and preserved. More explicitly, the translation function $\mathcal{T} : \mathbf{C}_1 \to \mathbf{C}_2$ should satisfy:

1. for $\mathcal{A} = \langle S, A, \alpha \rangle$ in $\mathbf{C}_1$, $\mathcal{T}(\mathcal{A}) = \langle S, A, \alpha' \rangle$ with the same set of states S,
2. the translation function $\mathcal{T}$ is injective, and
3. if $s, t \in S$, then $s \approx_{\mathcal{A}} t \Leftrightarrow s \approx_{\mathcal{T}(\mathcal{A})} t$, i.e. two states are bisimilar in the translated automaton (according to bisimilarity in the class $\mathbf{C}_2$) if and only if they were bisimilar in the original automaton (according to bisimilarity for the class $\mathbf{C}_1$).

The relation $\rightarrow$ between the classes of (probabilistic) automata is a preorder.

Basically our expressiveness criterion states that the class $\mathbf{C}_1$ is really embedded in the class $\mathbf{C}_2$, i.e. the translations are nothing else but "suitable copies" of the automata of the first class existing in the second class. Note that only preservation of bisimulation is not enough. For example, we could define a translation from reactive systems to LTSs that preserves bisimulation, by forgetting the probabilities, as in the following example.



But we do not consider the class of LTSs more expressive than the class of reactive probabilistic automata, and the translation is by no means injective.

Another hierarchy result is the hierarchy of reactive, generative, stratified and non-deterministic automata of [40] and [39]. The expressiveness criterion used by van Glabbeek et al. is different. They consider a class $\mathbf{C}_2$ *more expressive* than a class $\mathbf{C}_1$ if there is an *abstraction* mapping that maps each automaton of the class $\mathbf{C}_2$ to an automaton of the class $\mathbf{C}_1$, such that the state set remains the same and bisimulation is preserved. The abstraction mappings are not injective by nature. An example of such an abstraction mapping is the translation that forgets the probabilities in the previous figure. Therefore, in their setting the class **React** is more expressive than the class **NA**.

The Hierarchy

Theorem 1. *[26] The class embeddings presented in Figure 1 hold among the probabilistic system types.* □

The proof of Theorem 1 (except for the strictly alternating classes) is given in [26] using a technical result in terms of injective natural transformations. Due to the correspondence of concrete bisimulation with coalgebraic bisimulation for all the treated systems, the theorem remains valid without mentioning the coalgebraic theory behind it. Instead of presenting a proof here, we will explicitly state the translations for each arrow in Figure 1, give some examples and illustrate how preservation and reflection of bisimulation can be proven in one concrete case. We present the translations for the arrows of Figure 1 in several groups: simple arrows based on inclusion, arrows that show a change from external to

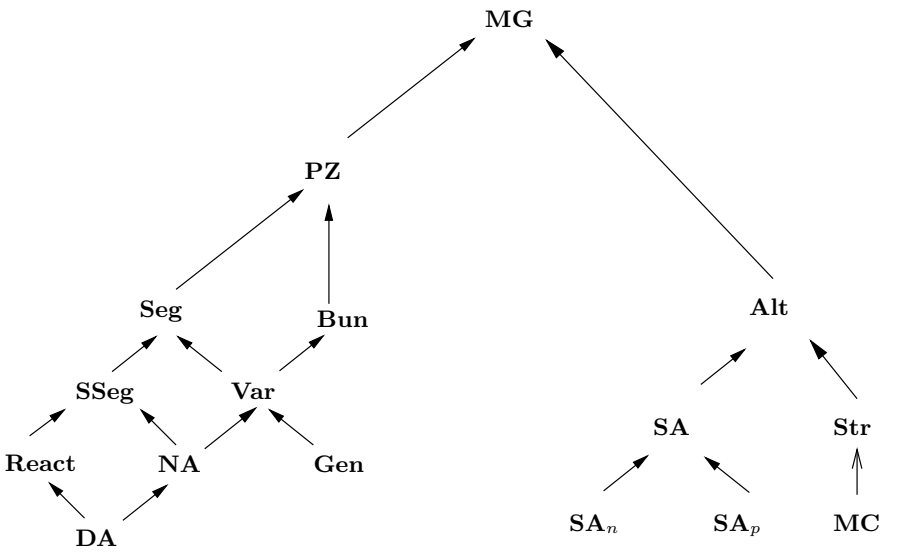


Fig. 1. Class embeddings

full non-determinism, arrows that change an element to a singleton, arrows that change an element to a corresponding Dirac distribution and more specific arrows. The translations are quite natural. It is the property of preservation and reflection of bisimilarity that adds justification to the translations.

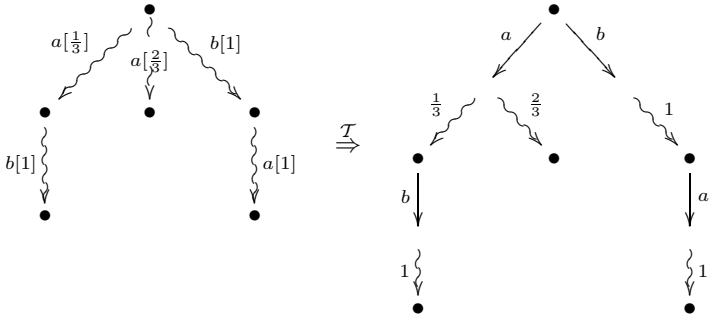
Simple Arrows. Let $\mathbf{C}_1$ and $\mathbf{C}_2$ be two classes of probabilistic automata. If $\mathbf{C}_1 \subseteq \mathbf{C}_2$ then $\mathbf{C}_1 \to \mathbf{C}_2$. Therefore, the embeddings $\mathbf{SA} \to \mathbf{Alt}$, $\mathbf{SA}_n \to \mathbf{SA}$ and $\mathbf{SA}_p \to \mathbf{SA}$ hold. Furthermore, if $\mathbf{C}_1$ is defined with a transition function $\alpha_1 : S \to C_1(S)$ and $\mathbf{C}_2$ has a transition function of type $\alpha_2 : S \to C_2(S)$ such that for all S, $C_1(S) \subseteq C_2(S)$ then every automaton of the class $\mathbf{C}_1$ can be considered an automaton of the class $\mathbf{C}_2$, by only extending the codomain of the transition function. In this case also $\mathbf{C}_1 \to \mathbf{C}_2$. The following arrows of Figure 1 hold due to extending the codomain of the transition function: $\mathbf{MC} \to \mathbf{Str}$, $\mathbf{Gen} \to \mathbf{Var}$, $\mathbf{NA} \to \mathbf{Var}$, $\mathbf{PZ} \to \mathbf{MG}$. In each of these cases the translation of the automata is basically the identity mapping. For example, every generative automaton is a Vardi automaton without non-deterministic states, or every Markov chain is a stratified automaton that has no action-transitions, i.e. no deterministic states.

From External to Full Non-Determinism. Two of the embedding arrows of Figure 1, $\mathbf{DA} \to \mathbf{NA}$ and $\mathbf{React} \to \mathbf{SSeg}$, show that every system with only external non-determinism can be considered as a system with full non-determinism that never uses the full-nondeterminism option. Let $\mathcal{A} = \langle \mathcal{S}, \mathcal{A}, \alpha \rangle \in \mathbf{DA}$. Then $\mathcal{T}(\mathcal{A}) = \langle S, A, \alpha' \rangle \in \mathbf{NA}$ is given by

$$\alpha'(s) = \{\langle a, s' \rangle \mid \alpha(s)(a) = s' \in S\} \in \mathcal{P}(A \times S).$$

If we consider automata as diagrams, i.e. transition graphs, then this translation does not change the transition graph of a **DA** system.

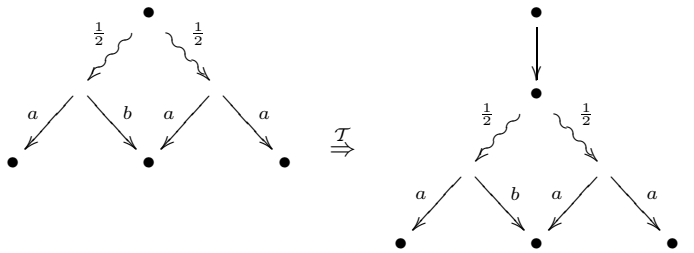
For **React** $\to$ **SSeg** a similar translation is used, changing a partial function to its graph. Due to the notation used for reactive systems in Section 3, the diagram of a reactive system when considered a simple Segala system has to be re-drawn, as in the next example.



a reactive system as a simple Segala system

Singleton Arrows. In several cases the translation only changes an element (state, or pair of state and action, or distribution) into a singleton set containing this element.

Bun $\to$ **PZ**: Let $\mathcal{A} = \langle S, A, \alpha \rangle$ be a bundle probabilistic automaton, i.e. $\alpha : S \to \mathcal{D}(\mathcal{P}(A \times S)) + 1$, then the translation to a Pnueli-Zuck automaton is achieved by putting $\mathcal{T}(\mathcal{A}) = \langle S, A, \alpha' \rangle$ for $\alpha'(s) = \{\alpha(s)\}$ if $\alpha(s)$ is a distribution, and $\alpha'(s) = \emptyset$ if $\alpha(s) = *$.



a bundle transition as a Pnueli-Zuck transition

Str $\to$ **Alt**: In this case $\mathcal{T}(\langle S, A, \alpha \rangle) = \langle S, A, \alpha' \rangle$ where $\alpha : S \to \mathcal{D}(S) + A \times S + 1$ and $\alpha' : S \to \mathcal{D}(S) + \mathcal{P}(A \times S)$ and

$$\alpha'(s) = \begin{cases} \{\alpha(s)\} & \text{if } \alpha(s) \in A \times S \\ \emptyset & \text{if } \alpha(s) = * \\ \alpha(s) & \text{otherwise} \end{cases}$$

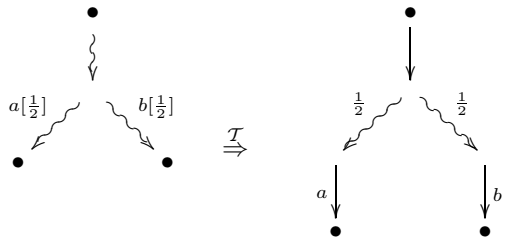
The diagram of a stratified automaton when translated to alternating automaton stays the same.

Seg → PZ: Let $\mathcal{A} = \langle \mathcal{S}, \mathcal{A}, \alpha \rangle$ be a Segala automaton, $\alpha : S \to \mathcal{P}(\mathcal{D}(A \times S))$. Then $\mathcal{T}(\mathcal{A}) = \langle S, A, \alpha' \rangle$ where α' is determined from α in the following way:

$$\alpha'(s) = \{\mu' \mid \mu \in \alpha(s)\}$$

where μ' is constructed from μ by changing a distribution over pairs to distribution over singletons of pairs, i.e.

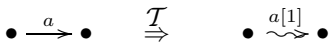
$$\mu' = \{\{\langle a, s' \rangle\} \mapsto \mu(a, s')\}.$$



a Segala transition as a Pnueli-Zuck transition

Dirac Arrows. Sometimes a transformation from an element to a Dirac distribution for this element is needed. This kind of translation embeds the non-probabilistic automata, **DA** and **NA**, into the reactive and simple Segala automata, respectively.

DA → React: In this case every transition has to be changed to a probabilistic transition with probability 1.



a **DA** transition as a reactive transition

For $\langle S, A, \alpha \rangle \in \mathbf{DA}$, the translation is $\mathcal{T}(\langle S, A, \alpha \rangle) = \langle S, A, \alpha' \rangle$, if $\alpha(s) \in (S + 1)^A$, we put $\alpha'(s) \in (\mathcal{D}(S) + 1)^A$ such that $\alpha(s)(a) = t \in S \iff \alpha'(s)(a) = \mu_t^1$.

NA → SSeg: For obtaining a simple Segala automaton out of a LTS, we change the next state of every transition to a Dirac distribution for this state.

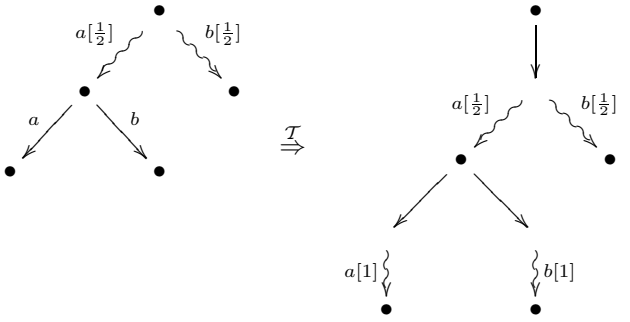
a **NA** transition as a simple Segala transition

Formally, $\mathcal{T}(\langle S, A, \alpha \rangle) = \langle S, A, \alpha' \rangle$ such that $\alpha'(s) = \{\langle a, \mu^1_{s'} \rangle \mid \langle a, s' \rangle \in \alpha(s)\}$.

Specific Arrows

Var → Seg: Let $\mathcal{A} = \langle S, A, \alpha \rangle$ be a Vardi automaton, with $\alpha(s) : S \to \mathcal{D}(A \times S) + \mathcal{P}(A \times S)$. The injective translation to a Segala automaton is given by $\mathcal{T}(\mathcal{A}) = \langle S, A, \alpha' \rangle$ for

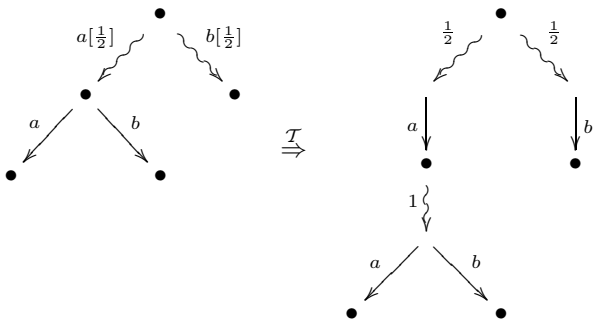
$$\alpha'(s) = \begin{cases} \{\alpha(s)\} & \text{if } \alpha(s) \in \mathcal{D}(A \times S) \\ \{\mu^1_{\langle a, s' \rangle} \mid \langle a, s' \rangle \in \alpha(s)\} & \text{if } \alpha(s) \in \mathcal{P}(A \times S) \end{cases}$$



a Vardi system as a Segala system

Var $\rightarrow$ **Bun**: This translation is orthogonal to the one that gives us **Var** $\rightarrow$ **Seg**. For a Vardi automaton $\mathcal{A} = \langle \mathcal{S}, \mathcal{A}, \alpha \rangle$ we put $\mathcal{T}(\mathcal{A}) = \langle S, A, \alpha' \rangle$ where

$$\alpha'(s) = \begin{cases} \{\{\langle a, s' \rangle\} \mapsto \mu(a, s')\} & \text{if } \alpha(s) = \mu \in \mathcal{D}(A \times S) \\ \mu^1_{\alpha(s)} & \text{if } \alpha(s) \in \mathcal{P}(A \times S) \end{cases}$$



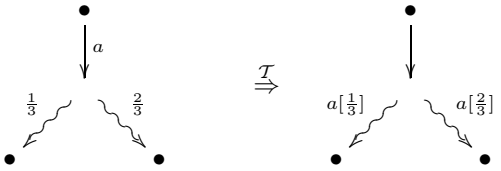
a Vardi system as a bundle system

Remark 5. Both in **Var** $\rightarrow$ **Seg** and in **Var** $\rightarrow$ **Bun** the translated transition function α' is well defined even when we consider $\mathcal{D}(A \times S) \cap \mathcal{P}(A \times S) \neq \emptyset$, i.e. we identify $\mu^1_{\langle a,s \rangle}$ with $\{\langle a, s \rangle\}$. Furthermore, this identification is needed to obtain injectivity of the translations.

Alt $\rightarrow$ **MG**: Similarly as when translating Vardi systems, with an extra singleton construction, an alternating automaton $\mathcal{A} = \langle \mathcal{S}, \mathcal{A}, \alpha \rangle$, $\alpha : S \rightarrow \mathcal{D}(S) + \mathcal{P}(A \times S)$ is translated into a general probabilistic automaton. We put $\mathcal{T}(\mathcal{A}) = \langle S, A, \alpha' \rangle$ where

$$\alpha'(s) = \begin{cases} \{\{\{\langle a, s' \rangle\} \mapsto \mu(a, s')\}\} & \text{if } \alpha(s) = \mu \in \mathcal{D}(S) \\ \{\mu^1_{\alpha(s)}\} & \text{if } \alpha(s) \in \mathcal{P}(A \times S) \end{cases}$$

SSeg $\rightarrow$ **Seg**: In order to change a transition of a simple Segala automaton to a transition of a Segala automaton it is enough to push the action label into the distribution.



a simple Segala transition as a Segala transition

Formally, if $\mathcal{A} = \langle S, A, \alpha \rangle$, $\alpha : S \to \mathcal{P}(A \times \mathcal{D})$ then $\mathcal{T}(\mathcal{A}) = \langle S, A, \alpha' \rangle$ where

$$\langle a, \mu \rangle \in \alpha(s) \iff \mu_a \in \alpha'(s) \in \mathcal{P}(\mathcal{D}(A \times S)$$

and $\mu_a(a, s') = \mu(s')$ for all $s' \in S$.

For this last translation, as an illustration, we give the proof of preservation and reflection of bisimilarity. Let μ be a distribution on S and a an action in A. Denote by μ_a the distribution on $A \times S$ obtained from μ by $\mu_a(a, s) = \mu(s)$. Clearly, for any subset $X \subseteq S$ we have $\mu[X] = \mu_a[a, X]$, which yields $\mu \equiv_R \mu' \iff \mu_a \equiv_{A,R} \mu'_a$. By the translation, the following holds:

1. If $s \xrightarrow{a}\!\!\rightsquigarrow_{\mathcal{A}} \mu$, then $s \to\!\!\rightsquigarrow_{\mathcal{T}(\mathcal{A})} \mu_a$.
2. If $s \to\!\!\rightsquigarrow_{\mathcal{T}(\mathcal{A})} \mu$ then there exists $a \in A$ such that $\mu = \nu_a$ for some distribution ν on S and $s \xrightarrow{a}\!\!\rightsquigarrow_{\mathcal{A}} \nu$.

Now assume $s \approx_{\mathcal{A}} t$, i.e. there exists a bisimulation R (Definition 11) with $\langle s, t \rangle \in R$. We prove that R is a bisimulation (Definition 12) on the state space of $\mathcal{T}(\mathcal{A})$. Let $s \to\!\!\rightsquigarrow_{\mathcal{T}(\mathcal{A})} \mu$. By 2. we have that $\mu = \nu_a$ for some $a \in A$ and some $\nu \in \mathcal{D}(S)$, and $s \xrightarrow{a}\!\!\rightsquigarrow_{\mathcal{A}} \nu$. Since R is a bisimulation on the state space of $\mathcal{A}$ we have that there exists a distribution ν' such that $t \xrightarrow{a}\!\!\rightsquigarrow_{\mathcal{A}} \nu'$ and $\nu \equiv_R \nu'$. Now it follows by 1. that $s \to\!\!\rightsquigarrow_{\mathcal{T}(\mathcal{A})} \nu'_a$, and furthermore $\mu = \nu_a \equiv_{A,R} \nu'_a$. So, R is a bisimulation on the state set of $\mathcal{T}(\mathcal{A})$.

The opposite is analogous. If R is a bisimulation (Definition 12) with $\langle s, t \rangle \in R$ on the set of states of $\mathcal{T}(\mathcal{A})$, we prove that R is a bisimulation (Definition 11) on the set of states of $\mathcal{A}$. Assume $s \xrightarrow{a}\!\!\rightsquigarrow_{\mathcal{A}} \mu$. Then by 1., $s \to\!\!\rightsquigarrow_{\mathcal{T}(\mathcal{A})} \mu_a$. Since R is a bisimulation and $\langle s, t \rangle \in R$ we get that there exists $\nu \in \mathcal{D}(A \times S)$ such that $t \to\!\!\rightsquigarrow_{\mathcal{T}(\mathcal{A})} \nu$ and $\mu_a \equiv_{A,R} \nu$. Now, by 2., we get that there exists $a' \in A$ and a distribution μ' on S such that $\nu = \mu'_{a'}$, $t \xrightarrow{a'}\!\!\rightsquigarrow_{\mathcal{A}} \mu'$ and $\mu \equiv_R \mu'$. However, from $\mu_a \equiv_{A,R} \nu = \mu'_{a'}$ we get that $a' = a$ and hence $t \xrightarrow{a}\!\!\rightsquigarrow_{\mathcal{A}} \mu'$ and $\mu \equiv_R \mu'$, which completes the proof.

Remark 6. All the translations presented in this section are injective and preserve and reflect bisimilarity.

Strict Alternation vs. Complex Transition Function

A translation between simple Segala automata and automata of the class $\mathbf{SA}_n$ has been known for a long time, and has been recently justified in [25]. Similarly,

the class $\mathbf{SA}_p$ can be compared with the class of bundle probabilistic automata. In order to carry out these comparisons in our framework we slightly change the comparison criterion itself so that it allows for translations that do not keep the same state set.

Relaxed Expressiveness Criteria. Let $\mathbf{C}_1$ and $\mathbf{C}_2$ be two classes of probabilistic automata and let $\mathcal{T} : \mathbf{C}_1 \to \mathbf{C}_2$ be a translation mapping, such that if $\mathcal{A} = \langle S, A, \alpha \rangle$ then $\mathcal{T}(\mathcal{A}) = \langle S', A, \alpha' \rangle$

We say that the class $\mathbf{C}_1$ *is embedded* in the class $\mathbf{C}_2$, by the translation $\mathcal{T}$, notation $\mathbf{C}_1 \twoheadrightarrow \mathbf{C}_2$ if the following conditions hold:

1. for any automaton of $\mathbf{C}_1$, $S \subseteq S'$ or for any automaton of $\mathbf{C}_1$, $S' \subseteq S$,
2. bisimilarity is preserved and reflected for common states, i.e., $\forall s, t \in S \cap S' \colon s \approx_{\mathcal{A}} t \iff s \approx_{\mathcal{T}(\mathcal{A})} t$, and
3. the translation $\mathcal{T}$ is injective.

Furthermore, we say that the class $\mathbf{C}_1$ *is embedded up to irrelevant bisimilarity* in the class $\mathbf{C}_2$, by the translation $\mathcal{T}$, notation $\mathbf{C}_1 \longrightarrow\!\!\!\!\!\!/_{\approx}\ \mathbf{C}_2$ if the following conditions hold:

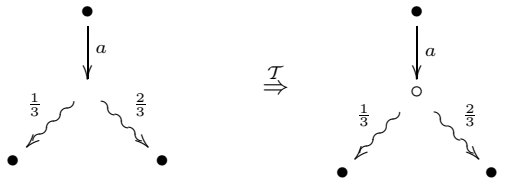
1. $S' \subseteq S$,
2. for any two common states $s, t \in S'$, $s \approx_{\mathcal{A}} t \iff s \approx_{\mathcal{T}(\mathcal{A})} t$, and
3. the translation $\mathcal{T}$ is injective up to bisimilarity of irrelevant states, i.e., if $\mathcal{A}_1 = \langle S_1, A, \alpha_1 \rangle$ and $\mathcal{A}_2 = \langle S_2, A, \alpha_2 \rangle$ are two automata of the first class such that $\mathcal{T}(\mathcal{A}_1) = \mathcal{T}(\mathcal{A}_2) = \langle S, A, \alpha \rangle$, then
$$s_1 \in S_1 \setminus S \Rightarrow \exists s_2 \in S_2 \setminus S \colon s_1 \approx s_2 \text{ and}$$
$$s_2 \in S_2 \setminus S \Rightarrow \exists s_1 \in S_1 \setminus S \colon s_1 \approx s_2.$$

Theorem 2. *The following embeddings compare the classes of strictly alternating, bundle and simple Segala probabilistic automata.*

Fig. 2. Strictly alternating models as models with a structured transition relation

Instead of a complete proof, we give the translations needed in each of the embeddings.

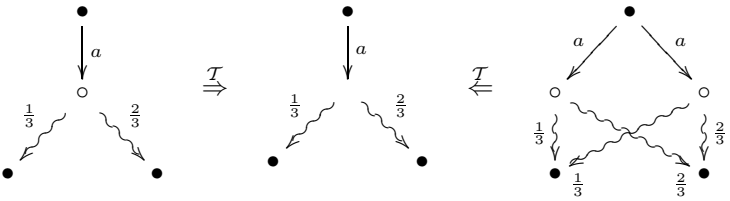
SSeg $\twoheadrightarrow$ $\mathbf{SA}_n$:



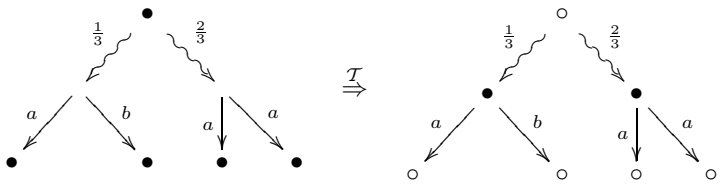
a simple Segala automaton as a $\mathbf{SA}_n$ automaton

Let $\mathcal{A} = \langle \mathcal{S}, \mathcal{A}, \alpha \rangle$ be a simple Segala automaton. The translation function translates it to $\mathcal{T}(\mathcal{A}) = \langle S', A, \alpha' \rangle$ by inserting a new probabilistic state for each transition, i.e., by changing $s \xrightarrow{a}\!\!\rightsquigarrow \mu$ to $s \xrightarrow{a} s_{s,a,\mu} \rightsquigarrow \mu$. Formally, $S' = S + S_p$ where S_p is a set of fresh probabilistic states, such that $S_p = \{s_{s,a,\mu} \mid s \in S, a \in A, \mu \in \mathcal{D}(S), \langle a, \mu \rangle \in \alpha(s)\}$. So, for every outgoing transition of a state there is a corresponding new state added to S_p. The transition function is defined as follows: if $s \in S$, such that $\langle a, \mu \rangle \in \alpha(s)$ and the corresponding new state is $s_{s,a,\mu}$ then and only then $\langle a, s_{s,a,\mu} \rangle \in \alpha'(s)$; if $s_{s,a,\mu} \in S_p$ corresponds to a transition $\langle a, \mu \rangle \in \alpha(s)$ then $\alpha'(s_{s,a,\mu}) = \mu$. The translation is injective and it preserves and reflects bisimilarity on S.

$\mathbf{SA}_n \longrightarrow\!\!\!\!\!/_{\approx}\ \mathbf{SSeg}$: The translation for this embedding, is the "inverse" of the case $\mathbf{SSeg} \twoheadrightarrow \mathbf{SA}_n$. If $\mathcal{A} = \langle S, A, \alpha \rangle \in \mathbf{SA}_n$ with a set of probabilistic states P and a set of non-deterministic states N, then $\mathcal{T}(\mathcal{A}) = \langle N, A, \alpha' \rangle$ where for every $s \in N$, $\alpha'(s) = \{\langle a, \mu \rangle \mid \langle a, s' \rangle \in \alpha(s) \text{ and } \alpha(s') = \mu\}$. Hence this translation forgets the probabilistic states, and turns two strictly alternating steps of the $\mathbf{SA}_n$ automaton into one transition of the corresponding Segala automaton. This translation is only injective up to bisimilarity of the disappearing probabilistic states. In fact it is even stricter than that, it is injective up to indistinguishable probabilistic states. The only cases of non-injectivity come as a consequence of having two probabilistic states s_{p1} and s_{p2} such that $\alpha(s_{p1}) = \alpha(s_{p2}) = \mu$. Then if s is a non-deterministic state with $s \xrightarrow{a} s_{p1}$ and $s \xrightarrow{a} s_{p2}$, the two "copies" $s \xrightarrow{a} s_{p1} \rightsquigarrow \mu$ and $s \xrightarrow{a} s_{p2} \rightsquigarrow \mu$ will be mapped to one single transition $s \xrightarrow{a}\!\!\rightsquigarrow \mu$. For example, the following two different $\mathbf{SA}_n$ automata translate to one **SSeg** automaton.



Bun $\twoheadrightarrow \mathbf{SA}_p$:



a bundle automaton as a $\mathbf{SA}_p$ automaton

The translation is as follows: $\mathcal{T}(\langle S, A, \alpha \rangle) = \langle S', A, \alpha' \rangle$ where $S' = S + S_n$, S_n containing the fresh non-deterministic states $S_n = \{s_{s,T} \mid s \in S, T \in spt(\alpha(s))\}$. The transition function is defined by: for $s \in S$ with $\alpha(s) = \mu$, we put $\alpha'(s) = \mu_s \in \mathcal{D}(S_n)$ where $\mu_s(s_{s,T}) = \mu(T)$, and for $s_{s,T} \in S_n$ we put $\alpha'(s_{s,T}) = T$.

$\mathbf{SA}_p \twoheadrightarrow \mathbf{Bun}$: The inverse of the translation $\mathbf{Bun} \twoheadrightarrow \mathbf{SA}_p$ gives us the translation in this case. We forget all the non-deterministic states in a $\mathbf{SA}_p$ automaton, being left with a bundle automaton. For a $\mathbf{SA}_p$ automaton $\mathcal{A} = \langle S, A, \alpha \rangle$ with a set of probabilistic states P and non-deterministic states N we put $\mathcal{T}(\mathcal{A}) = \langle P, A, \alpha' \rangle$ where for $s \in P$ with $\alpha(s) = \mu \in \mathcal{D}(N)$ we define $\alpha'(s) = \mu_s \in \mathcal{D}(\mathcal{P}(A \times S))$ such that $\mu_s(\alpha(s')) = \mu(s')$. Note that the phenomenon of losing "copies" as in $\mathbf{SA}_n \longrightarrow\!\!\!\!\!/_{\approx}\ \mathbf{SSeg}$ does not appear here. Even though states are lost, no transitions are identified, i.e. no arrow is lost.

Remark 7. Restricting to subclasses of the class **SA**, i.e. considering the classes $\mathbf{SA}_n$ and $\mathbf{SA}_p$, is necessary to obtain injectivity (up to bisimilar irrelevant states) of the embedding mappings.

6 Conclusions

In this overview we have presented various types of probabilistic automata, including generative, reactive and stratified ones, strictly alternating and alternating ones, the simple Segala, Segala and Vardi type of probabilistic automata and the bundle, Pnueli-Zuck and general ones.

A major part of our work has been devoted to the comparison of the various classes of probabilistic automata, taking strong bisimilarity for these automata as a starting point, resulting in a hierarchy of probabilistic system types. Additionally, we have discussed the extent of non-determinism that can be modelled in the various types of automata and the operator of parallel composition for them. Classes positioned higher in the map of Figure 1 can be characterized as closures of the simpler classes under parallel composition, which clarifies the need for the more complex models.

The results obtained are briefly presented in Figure 1, Figure 2 and Table 1. From there various conclusions on probabilistic system modelling can be drawn. For instance, if presence of non-determinism and closedness under all variants of parallel composition are desired properties on the one hand, and having as simple a model as possible is needed on the other hand, then whether the choice is for input (reactive) or output (generative) type of systems, the best choice appears to be the simple Segala model and the bundle model, respectively. Different requirements lead to different choices, but we hope the map of probabilistic automata based models will prove to be useful in making a right decision.

References

1. R. Alur and T.A. Henzinger, *Reactive modules*, Formal Methods in System Design **15** (1999), 7–48, A preliminary version appeared in the Proceedings of the 11th Annual Symposium on Logic in Computer Science (LICS), IEEE Computer Society Press, 1996, pp. 207-218.
2. L. de Alfaro, T.A. Henzinger, and R. Jhala, *Compositional methods for probabilistic systems*, CONCUR 2001 - Concurrency Theory: 12th International Conference, Aalborg, Denmark, August 20-25, 2001, LNCS, vol. 2154, Springer-Verlag, 2001, pp. 351–365.

3. L. de Alfaro, *Formal verification of probabilistic systems*, Ph.D. thesis, Stanford University, 1997.
4. ———, *Stochastic transition systems*, International Conference on Concurrency Theory, CONCUR, vol. 1466, LNCS, 1998, pp. 423–438.
5. S. Andova, *Process algebra with probabilistic choice*, Proc. 5th International AMAST Workshop, ARTS'99, Bamberg, Germany (J.-P. Katoen, ed.), LNCS 1601, Springer-Verlag, 1999, pp. 111–129.
6. ———, *Probabilistic process algebra*, Ph.D. thesis, Eindhoven University of Technology, 2002.
7. A. Bianco and L. de Alfaro, *Model checking of probabilistic and nondeterministic systems*, Found. of Software Tech. and Theor. Comp. Sci., LNCS, vol. 1026, Springer-Verlag, 1995.
8. C. Baier, *Polynomial time algorithms for testing probabilistic bisimulation and simulation*, Proc. 8th International Conference on Computer Aided Verification (CAV'96), Lecture Notes in Computer Science, vol. 1102, 1996, pp. 38–49.
9. C. Baier, *On algorithmic verification methods for probabilistic systems*, Habilitationsschrift, FMI, Universitaet Mannheim, 1998.
10. J.C.M. Baeten, J.A. Bergstra, and S.A. Smolka, *Axiomatizing probabilistic processes: ACP with generative probabilities*, Information and Computation **121** (1995), no. 2, 234–255.
11. R. Blute, J. Desharnais, A. Edalat, and P. Panangaden, *Bisimulation for labelled Markov processes*, LICS'97, 1997, pp. 149–158.
12. C. Baier, P.R. D'Argenio, H. Hermanns, and J.-P. Katoen, *How to cook a probabilistic process calculus*, unpublished, 1999.
13. C. Baier, B. Engelen, and M. Majster-Cederbaum, *Deciding bisimilarity and similarity for probabilistic processes*, Journal of Computer and System Sciences **60** (1999), 187–231.
14. M. Bernardo, *Theory and application of extended Markovian process algebra*, Ph.D. thesis, University of Bologna, 1999.
15. M. Bernardo and R. Gorrieri, *A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time*, Theoretical Computer Science **202** (1998), no. 1, 1–54.
16. E. Brinksma and H. Hermanns, *Process algebra and Markov chains*, Lectures on Formal Methods and Performance Analysis, First EEF/Euro Summer School on Trends in Computer Science, Berg en Dal, The Netherlands, July 3–7, 2000 (E. Brinksma, H. Hermanns, and J.-P. Katoen, eds.), LNCS, vol. 2090, Springer-Verlag, 2001, pp. 183–232.
17. C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, *Model checking continuous-time Markov chains by transient analysis*, CAV 2000, vol. 1855, LNCS, Springer-Verlag, 2000, pp. 358–372.
18. S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe, *A theory of communicating sequential processes*, Journal of the ACM **31** (1984), 560–599.
19. J.A. Bergstra and J.W. Klop, *Algebra of communicating processes with abstraction*, Theoretical Computer Science **37** (1985), 77–121.
20. C. Baier and M.Z. Kwiatkowska, *Domain equations for probabilistic processes*, 4th Workshop on Expressiveness in Concurrency (EXPRESS'97), Santa Margherita, vol. 7, Electronic Notes in Theoretical Computer Science, 1997.
21. ———, *Domain equations for probabilistic processes*, Mathematical Structures in Computer Science **10** (2000), 665–717.

22. A. Benveniste, B. C. Levy, E. Fabre, and P. Le Guernic, *A calculus of stochastic systems for the specification, simulation, and hidden state estimation of mixed stochastic/non-stochastic systems*, Theoretical Computer Science **152** (1995), 171–217.
23. B. Bloom and A. R. Meyer, *A remark on bisimulation between probabilistic processes*, Foundations of Software Technology and Theoretical Computer Science, LNCS, vol. 363, Springer-Verlag, 1989, pp. 26–40.
24. C. Baier and M.I.A. Stoelinga, *Norm fuctions for probabilistic bisimulations with delays*, Proceedings of 3rd International Conference on Foundations of Science and Computation Structures (FOSSACS), Berlin, Germany, March 2000 (J. Tiuryn, ed.), LNCS, vol. 1784, "Springer-Verlag", 2000, pp. 1–16.
25. E. Bandini and R. Segala, *Axiomatizations for probabilistic bisimulation*, Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP) 2001, Crete, LNCS 2076, 2001, pp. 370–381.
26. F. Bartels, A. Sokolova, and E.P. de Vink, *A hierarchy of probabilistic system types*, Electronic Notes in Theoretical Computer Science (H. Peter Gumm, ed.), vol. 82, Elsevier, 2003.
27. P. Buchholz, *Markovian process algebra: Composition and equivalence*, in Proc. of PAPM '94, Erlangen (Germany), 1994, pp. 11–30.
28. L. Christoff and I. Christoff, *Efficient algorithms for verification of equivalences for probabilistic processes*, Proc. Workshop on Computer Aided Verification 1991 (K. Larsen and A. Skou, eds.), LNCS, vol. 575, 1991.
29. I. Christoff, *Testing equivalences and fully abstract models for probabilistic processes*, Proceedings of CONCUR'90 (J.C.M. Baeten and J.W. Klop, eds.), LNCS 458, Springer-Verlag, 1990, pp. 126–140.
30. R. Cleaveland, S.A. Smolka, and A. Zwarico, *Testing preorders for probabilistic processes*, Automata, Languages and Programming (ICALP '92), Vienna, LNCS, vol. 623, Springer-Verlag, 1992, pp. 708–719.
31. C. Courcoubetis and M. Yannakakis, *The complexity of probabilistic verification*, Journal of the ACM (JACM) **42** (1995), 857–907.
32. P.R. D'Argenio, *Algebras and automata for timed and stochastic system*, Ph.D. thesis, University of Twente, 1999.
33. J. Desharnais, A. Edalat, and P. Panangaden, *A logical characterization of bisimulation for labeled Markov processes*, Proc. LICS'98 (Indianapolis), 1998, pp. 478–487.
34. C. Derman, *Finite state Markovian decision proceses*, Academic Press, 1970.
35. P. D'Argenio, H. Hermanns, and J.-P. Katoen, *On generative parallel composition*, Proc. PROBMIV'98, ENTCS 22, 1998, pp. 105–122.
36. P.R. D'Argenio, B. Jeannet, H.E. Jensen, and K.G. Larsen, *Reachability analysis of probabilistic systems by successive refinements*, PAPM-PROBMIV 2001, Aachen, Germany (L. de Alfaro and S. Gilmore, eds.), LNCS, vol. 2165, Springer-Verlag, 2001, pp. 29–56.
37. ———, *Reduction and refinement strategies for probabilistic analysis*, PAPM-PROBMIV 2002, Copenhagen, Denmark (H. Hermanns and R. Segala, eds.), LNCS, Springer-Verlag, 2002.
38. A. Giacalone, C. Jou, and S. Smolka, *Algebraic reasoning for probabilistic concurrent systems*, Proc. of the Working Conf. on Programming Concepts and Methods, 1990. (M. Broy and C.B. Jones, eds.), North Holland, 1990, pp. 443–458.
39. R.J. van Glabbeek, S.A. Smolka, and B. Steffen, *Reactive, generative, and stratified models of probabilistic processes*, Information and Computation **121** (1995), 59–80.

40. R. J. van Glabbeek, S. A. Smolka, B. Steffen, and C. M. N. Tofts, *Reactive, generative, and stratified models of probabilistic processes*, Logic in Computer Science, 1990, pp. 130–141.
41. H. A. Hansson, *Time and probability in formal design of distributed systems*, Ph.D. thesis, Uppsala University, Department of Computer Systems, 1991, Also appeared in Real-Time Safety Critical Systems, vol. 1, Elsevier, 1994.
42. J.I. den Hartog, *Probabilistic extensions of semantical models*, Ph.D. thesis, Vrije Universiteit Amsterdam, 2002.
43. B. R. Haverkort, *Markovian models for performance and dependability evaluation*, Lectures on Formal Methods and Performance Analysis, First EEF/Euro Summer School on Trends in Computer Science, Berg en Dal, The Netherlands, July 3–7, 2000 (E. Brinksma, H. Hermanns, and J.-P. Katoen, eds.), LNCS, vol. 2090, Springer-Verlag, 2001, pp. 38–84.
44. H. Hermanns, *Interactive Markov chains*, Ph.D. thesis, Universiät Erlangen-Nürnberg, 1998, Revised version appeared as Interactive Markov Chains And the Quest for Quantified Quality, LNCS 2428, 2002.
45. J. Hillston, *A compositional approach to performance modelling*, Ph.D. thesis, University of Edinburgh, 1994, Also appeared in the CPHC/BCS Distinguished Dissertation Series, Cambridge University Press, 1996.
46. H. Hansson and B. Jonsson, *A logic for reasoning about time and reliability*, Formal Aspects of Computing **6** (1994), 512–535.
47. C.A.R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.
48. R. A. Howard, *Dynamic probabilistic systems*, John Wiley & Sons, Inc., New York, 1971.
49. J.I. den Hartog and E.P. de Vink, *Mixing up nondeterminism and probability: A preliminary report*, Proc. PROBMIV'98 (C. Baier, M. Huth, M. Kwiatkowska, and M. Ryan, eds.), ENTCS 22, 1998.
50. ———, *Verifying probabilistic programs using a Hoare-like logic*, International Journal of Foundations of Computer Science **13** (2002), 315–340.
51. B. Jonsson and K.G. Larsen, *Specification and refinement of probabilistic processes*, Proceedings of Sixth Annual IEEE Symposium on Logic in Computer Science, 1991. LICS '91., IEEE, 1991.
52. B. Jonsson, K.G. Larsen, and W. Yi, *Probabilistic extensions of process algebras*, Handbook of Process Algebras, Elsevier, North Holland, 2001.
53. B.P.F. Jacobs and J.J.M.M. Rutten, *A tutorial on (co)algebras and (co)induction*, Bulletin of the EATCS **62** (1996), 222–259.
54. C.-C. Jou and S.A. Smolka, *Equivalences, congruences and complete axiomatizations for probabilistic processes*, Proceedings of CONCUR'90 (J.C.M. Baeten and J.W. Klop, eds.), Springer-Verlag, 1990, pp. 367–383.
55. B. Jonnson and W. Yi, *Testing preorders for probabilistic processes can be characterized by simulations*, Theoretical Computer Science **282** (2002), 33–51.
56. M.Z. Kwiatkowska and G.J. Norman, *A testing equivalence for reactive probabilistic processes*, EXPRESS '98 Fifth International Workshop on Expressiveness in Concurrency, ENTCS 16(2), 1998.
57. J. G. Kemeny and J. L. Snell, *Finite Markov Chains*, Springer-Verlag, New York, 1976.
58. N. López and M. Núñez, *An overview of probabilistic process algebras and their equivalences*, 2003, this volume.
59. G. Lowe, *Probabilistic and prioritized models of timed CSP*, Theoretical Computer Science **138** (1995), 315–352.

60. L. R. Lewis and C. H. Papadimitriou, *Elements of the theory of computation*, Prentice-Hall, Englewood Cliffs, NJ, 1981.
61. K. G. Larsen and A. Skou, *Bisimulation through probabilistic testing*, Information and Computation **94** (1991), 1–28.
62. K.G. Larsen and A. Skou, *Compositional verification of probabilistic processes*, CONCUR '92, Third International Conference on Concurrency Theory, Stony Brook, NY, USA (R. Cleaveland, ed.), LNCS, vol. 630, Springer-Verlag, 1992, pp. 456–471.
63. N. A. Lynch and M. Tuttle, *Hierarchical completeness proofs for distributed algorithms*, Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing, 1987.
64. S. MacLane, *Categories for the working mathematician*, Springer-Verlag, 1971.
65. R. Milner, *Calculi for synchrony and asynchrony*, Theoretical Computer Science **25** (1983), 267–310.
66. ———, *Communication and Concurrency*, Prentice-Hall, 1989.
67. C. Morgan, A. McIver, K. Seidel, and J.W. Sanders, *Refinement oriented probability for CSP*, Formal aspects of computing **8** (1996), 617–647.
68. L.S. Moss, *Coalgebraic logic*, Annals of Pure and Applied Logic **96** (1999), 277–317.
69. G. Norman, *Metric semantics for reactive probabilistic processes*, Ph.D. thesis, School of Computer Science, University of Birmingham, 1997.
70. B. Plateau and K. Atif, *Stochastic automata network for modeling parallel systems*, IEEE Trans. on Software Engineering **17** (1991), 1093–1108.
71. A. Pnueli and L. Zuck, *Verification of multiprocess probabilistic protocols*, Distributed Computing **1** (1986), no. 1, 53–72.
72. ———, *Probabilistic verification*, Information and Computation **103** (1993), 1–29.
73. M.O. Rabin, *Probabilistic automata*, Information and Control **6** (1963), 230–245.
74. J.J.M.M. Rutten, *Universal coalgebra: A theory of systems*, Theoretical Computer Science **249** (2000), 3–80.
75. E.W. Stark, R. Cleaveland, and S.A. Smolka, *A process-algebraic language for probabilistic I/O automata*, Proc. CONCUR'03 (R. Amadio and D. Lugiez, eds.), LNCS, vol. 2761, Springer, 2003, pp. 193–207.
76. R. Segala, *Modeling and verification of randomized distributed real-time systems*, Ph.D. thesis, MIT, 1995.
77. K. Seidel, *Probabilistic communicating processes*, Theoretical Computer Science **152** (1995), 219–249.
78. R. Segala and N.A. Lynch, *Probabilistic simulations for probabilistic processes*, Proc. Concur'94, LNCS 836, 1994, pp. 481–496.
79. S. A. Smolka and B.U. Steffen, *Priority as extremal probability*, Proceedings of CONCUR'90 (J.C.M. Baeten and J.W. Klop, eds.), LNCS, vol. 458, Springer-Verlag, 1990, pp. 456–466.
80. M.I.A. Stoelinga, *Alea jacta est: verification of probabilistic, real-time and parametric systems*, Ph.D. thesis, University of Nijmegen, the Netherlands, 2002.
81. ———, *An introduction to probabilistic automata*, EATCS bulletin, vol. 78, 2002.
82. M.I.A. Stoelinga and F.W. Vaandrager, *Root contention in IEEE 1394*, Proc. 5th International AMAST Workshop, ARTS'99, Bamberg, Germany (J.-P. Katoen, ed.), LNCS, vol. 1601, Springer-Verlag, 1999, pp. 53–75.
83. ———, *A testing scenario for probabilistic automata*, Proceedings of the 30th International colloquium on automata, languages and programming (ICALP'03) Eindhoven, the Netherlands, June 2003, LNCS, vol. 2719, Springer-verlag, 2003, pp. 464–477.

84. M.Y. Vardi, *Automatic verification of probabilistic concurrent finite state programs*, Proc. FOCS'95 (Portland, Oregon), IEEE Computer Society Press, 1985, pp. 327–338.
85. E.P. de Vink, *On a functor for probabilistic bisimulation and the preservation of weak pullbacks*, Tech. Report IR–444, Vrije Universiteit Amsterdam, 1998.
86. E.P. de Vink and J.J.M.M. Rutten, *Bisimulation for probabilistic transition systems: a coalgebraic approach*, Theoretical Computer Science **221** (1999), 271–293.
87. S.-H. Wu, S. A. Smolka, and E. W. Stark, *Composition and behaviors of probabilistic I/O automata*, Theoretical Computer Science **176** (1997), 1–38.
88. W. Yi and K.G. Larsen, *Testing preorders for probabilistic and non-deterministic processes*, Protocol Specification, Testing and Verification (Florida, USA), vol. 12, 1992, pp. 47–61.