Probabilistic Models for Reo Connector Circuits

Christel Baier
Institut für Informatik I, Universität Bonn
Römerstraße 164, D-53117 Bonn, Germany
baier@cs.uni-bonn.de

Abstract: Constraint automata have been used as an operational model for Reo which offers a channel-based framework to compose complex component connectors. In this paper, we introduce a variant of constraint automata with discrete probabilities and nondeterminism, called *probabilistic constraint automata*. These can serve for compositional reasoning about connector components, modelled by Reo circuits with unreliable channels, e.g., that might loose or corrupt messages, or channels with random output values that, e.g., can be helpful to model randomized coordination principles.

Key Words: Probabilistic Constraint Automata, Reo, Coordination, Composition, Markov Decision Process, Bisimulation
Category: F4. Logics and Meaning of Programs, G3. Probability and Statistics, D2. Software Engineering

1 Introduction

Coordination models and languages provide a formalization of the *glue-code* that binds individual components and organizes the communication and cooperation between them. In the past 15 years, various types of coordination models have been developed, including techniques for the design and the analysis of such models. They all have in common that they yield a clear separation between the internal structure of the components and their relationship that arises through the organization of their interactions.

In this paper, we concentrate on the *exogenous coordination language Reo* that has been introduced by Arbab [6]. Reo is a channel-based calculus where complex component connectors are synthesized from channels via certain composition operators. Reo component connectors have a graphical representation as a network of channels, resembling logical circuits used in hardward design, and thus called *Reo circuits*. Despite its simplicity, Reo is a powerful glue-language that has been applied, for instance, in the context of the Cybernetic Incident Management project [18] for composition of web services [22]; to model business processes [45] or biological systems [19]. A formal semantics for Reo was provided in [6] where the possible data flow in the configurations of a Reo circuits is described by means of so-called accept- and offer-predicates. These predicates formalize the conditions under which a node in a Reo circuit accepts to read a certain (input) value from the sink end of a channel and offers to write a certain (output) value at the source end of a channel. The conditions for the

I/O-operations can be data-dependent (e.g., a node might accept to read data item d, but not data item d') and are derived from the semantics of the involved channels. Compositional semantics have been proposed by Arbab and Rutten [9] using relations over timed data streams and in [8] by an automata-model, called *constraint automata*. Constraint automata describe the configurations of Reo component connectors, e.g. contents of FIFO channels, and the stepwise behavior by means of the possible I/O-operations and their effect on the configurations. Constraint automata thus serve as an operational model for Reo circuits and yield the basis for verification algorithms, such as equivalence checking [8] or model checking against temporal logic specifications [7].

Although the Reo coordination model allows for arbitrary channel types, the semantics presented in [9] or [8] for Reo circuits rely on the assumption that the semantics of the involved channel types are formally described in a non-probabilistic way. The goal of this paper is to provide formal models for Reo circuits built out of channels that might behave non-deterministically and probabilistically. Throughout the paper constraint automata will serve as reference model. We will propose two variants thereof that yield *probabilistic operational models* for Reo circuits built out of faulty or randomized channels. These probabilistic automata-models can serve as the basis for a quantitative analysis of connector components that are formalized by Reo circuits. We will study Reo circuits where certain channels are unreliable, e.g. channels that might loose certain messages from their buffer or corrupt written messages with some small probability, and randomized channels that may deliver random data values. In fact, both types of channels appear rather natural in the context of Reo. First, channel connections in the real world are reliable to some large extent, but exceptional behaviors occur with small probability. Second, randomized channels are needed in the Reo framework to model coordination principles relying on coin-tossing actions. For instance, there is a wide range of coordination algorithms for distributed systems that use randomization to break symmetry, see e.g. [30]. To model channels with probabilistic output values we introduce a variant of constraint automata with discrete probabilities for the transitions. While a rather straightforward extension of constraint automata, called simple probabilistic constraint automata (SPCA), suffices if only probabilities for the successor configurations are provided, a more general and difficult model is needed when for a given configuration the potential next steps are given by probabilities for the possible I/O-operations. To support *compositional reasoning*, we provide semantic operators corresponding to Reo's main primitives (join and hiding) to model complex component connectors, and thus obtain a compositional framework to generate (S)PCA from a given Reo circuit. Furthermore, we introduce a notion of *bisimulation equivalence* for probabilistic constraint automata that serves to formalize the equivalence of two Reo circuits by means of the (probabilities for the) observable data flow, but abstracting away from their configurations.

Organization of the paper. Section 2 is a brief introduction to Reo and (non-probabilistic) constraint automata. In Sections 3 and 4, we consider component connectors that behave probabilistically according to the possible faulty behaviors of channels, and introduce SPCA and PCA. Section 5 presents a notion of bisimulation equivalence on (S)PCA. Section 6 concludes the paper.

2 Reo and constraint automata

In this section we provide a brief introduction to Reo which is an exogenous coordination model introduced by Arbab [6] that supports compositional reasoning about complex coordinators for components, called *component connectors*. We present here only a brief summary of Reo's main features and its constraint automata semantics [8]. Further details about Reo and its semantics can be found in [6, 9, 37, 8] or in the extended version of this paper [10]. For larger examples and applications of Reo we refer to [5, 22, 19].

Reo's coordination and cooperation principles are purely *channel-based*. Any component connector is built by a network of channels (called Reo circuit) that serve to provide the protocol that controls and organizes the communication, synchronization and cooperation between the concurrent components. The atomic building blocks in Reo are *channels* that connect two *channel ends*. The channel ends can be seen as ports where data items can enter or leave a channel via the corresponding write or read operations. The enabledness of such I/O-operations at the channel ends and their effect depend on the *type* of the given channel.

2.1 Reo circits

Classical channel types have a *source end* that provides input values through write actions and a *sink end* that takes data items from the channel by performing reading actions. Reo generalizes this traditional concept of channels and allows for channel types with two arbitrary channel ends, either a source and a sink end, or two source ends or two sink ends. Reo places no restriction on the behavior of a channel and allows an open-ended set of different channel types to be used simultaneously together. The only requirement for Reo's channel types is a specification that declares (i) the type of the two channel ends (sink or source end) and (ii) the input/output behavior by means of the relation between their write and read operations at the two channel ends (e.g., by means of a constraint automaton, see Section 2.1).

Figure 1 shows a few examples of channel types. The simplest form of an asynchronous channel type is a *FIFO channel* with one buffer cell, simply called a FIFO1 channel. It has one source end and one sink end. We graphically represent a FIFO1 channel by a small box in the middle of an arrow leading from the source end to the sink end. The buffer is assumed to be initially empty if no data item is shown in the box. A *synchronous channel* has a source and a sink end, but no buffer. It requires that the writing operations at its source end are synchronized with matching read operations at its sink end. Several variants

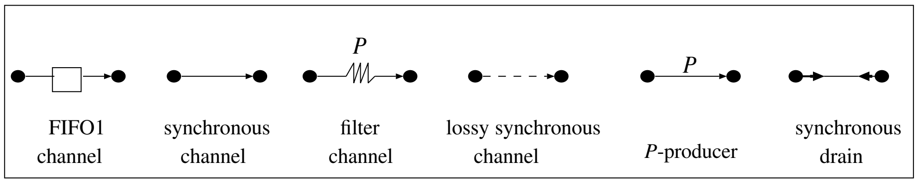


Figure 1: Some basic channel types

of such synchronous channels have been used in the Reo framework. A *P-filter channel* is a data-dependent variant of a synchronous channel. It uses a pattern P for specifying the type of data items that can be transmitted through the channel. For our purposes here, we may assume that P is formalized as a set of data items. For data items $d \in P$, the filter channels behaves as an ordinary synchronous channel. For the data items $d \notin P$, writing d at the source end is always enabled, but the written value d is immediately lost. A P-producer requires synchronous writing and reading at its source and sink end. When an arbitrary data item is written at the source end of a P-producer, it generates (nondeterministically) a data item $d \in P$ which is then taken at its sink end. A *lossy synchronous channel* behaves like a standard synchronous channel except that it is non-blocking for the write-operations at its source end, i.e., writing at its source end is always enabled. If a matching read operation is not available at the sink end then the written data is immediately lost. Reo permits also for more exotic channels like *synchronous drains*. Drains have two source ends, but no sink ends, and hence no data value can ever be obtained from these channels. For a synchronous drain, the write operations at the two source ends have to be synchronized. The written values are irrelevant.

Reo offers a compositional framework to built component connectors by glueing together several channels of arbitary types. The result is a visual representation, a so-called Reo-circuit, which can be formalized as a graph with various additional information. The nodes of this graph represent non-empty *sets of channel ends* that have been combined through a series of join-operations (explained below). The edges of this graph represent the channels that have been established between the nodes. Reo does not impose any restrictions on the used channel types, except that their instances must have two (different) channel ends and an operational semantics has to be provided by the user (we will use constraint automata for this purpose, see Section 2.1). For a given node A of a Reo circuit, $\mathsf{Src}(A)$ denotes the set of source ends that are coincident on A, while $\mathsf{Snk}(A)$ denotes the set of sink ends of A. Node A is called a *source node* if $\mathsf{Src}(A) \neq \emptyset$ and $\mathsf{Snk}(A) = \emptyset$, a *sink node* if $\mathsf{Src}(A) = \emptyset$ and $\mathsf{Snk}(A) \neq \emptyset$, and *mixed node* if $\mathsf{Src}(A) \neq \emptyset$ and $\mathsf{Snk}(A) \neq \emptyset$.

We now explain the intuitive semantics of the nodes which has been formally defined in [6]. Source nodes of a circuit are analogous to input ports, and sink nodes to output ports. Data flow at the source and sink nodes depends on the pending write and read operations of the environment, i.e., components or other component connectors that are linked to the source/sink nodes of a Reo circuit and that might feed a source node of the circuit with a certain input value or take a data item from one of its sink nodes. In our operational model, the environment is assumed to be unknown and the possible interactions via I/O-operations at the sink and source nodes are modelled by nondeterminism.

A component can write data items to a *source node* A of a Reo circuit that it is connected to. A write operation on A succeeds only if *all* (source) channel ends coincident on A accept the data item, in which case the data item is transparently written to every source end coincident on the node. A source node, thus, acts as a *replicator*: the effect of any write operation on A is that the same value is written on all channels that have a source end in A. A component can obtain data items from a *sink node* of a Reo circuit that it is connected to. A read operation at a sink node succeeds if and only if at least *one* of the (sink) channel ends coincident on the node offers a suitable data item. If more than one coincident channel end offers suitable data items, one is selected nondeterministically. In other words, the read operations at the sink ends on A are performed in an interleaved way, but never occur simultaneously. A sink node, thus, acts as a nondeterministic *merger*. The *mixed nodes* are self-contained entities that combine the behavior of a sink node (merger) and a source node (replicator) and perform read and write operations synchronously. A mixed node A takes a suitable data item offered by one of its coincident sink channel ends and replicates it into all of its coincident source channel ends. This requires an enabled read action at one of A's sink ends and that writing the corresponding data item is enabled for all its source ends.

The idea for the compositional construction of Reo circuits (as formal model for a component connector) is as follows. We start with several channels, each of them can be viewed as an atomic Reo circuit consisting of two nodes, one for each channel end, and the given channel. The main operations to create Reo circuits for complex component connectors are join and hiding. We first will explain the join-operation which combines nodes. For the purpose of this paper, we can think of join as an operator that takes as input two disjoint Reo circuits R_1 and R_2 with the same sets of channel types. Then, join creates a new Reo circuit $R = R_1 \bowtie R_2$ that arises from R_1 and R_2 by creating a new node C and combines all channel ends that are coincident on A in R_1 and on B in R_2.

Reo's hiding operator serves to make (parts of) the internal structure of a component connector invisible and unaccessible to its environment. Hiding is depicted by putting a box around certain mixed nodes with certain sink and source nodes at its border. The (mixed) nodes inside the box are then called hidden. In the pictures, their names are not shown. The subcircuit consisting of the hidden nodes and the source and sink nodes at its border can be viewed as a component where the sink and source nodes play the role of its input and output ports. Such components can still be used as building blocks of Reo component connectors, in the same way as channels serve as building blocks.

2.2 Constraint automata

An operational semantics for Reo circuits based on so-called *constraint automata* has been presented in [8]. Constraint automata are variants of labeled transition systems where transitions are augmented with pairs $\langle N, g\rangle$ rather than action labels. The states of a constraint automaton – as an operational model for a Reo circuit – stand for the configurations, e.g., the contents of the buffers for FIFO channels or other buffered channels. The transition labels $\langle N, g\rangle$ can be viewed as *sets of I/O-operations* that will be performed *in parallel*.

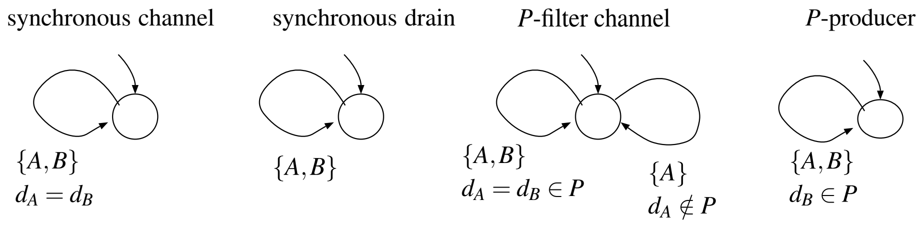
Constraint automata use a finite set $\mathcal{N} = \{A_1, \ldots, A_n\}$ where the A_i's stand for the nodes of a Reo circuit. They are classified into source nodes, sink nodes and mixed nodes. We write $\mathcal{N}^{src}$ for the set of source nodes, $\mathcal{N}^{snk}$ for the set of sink nodes and $\mathcal{N}^{mix}$ for the set of mixed nodes and require that $\mathcal{N} = \mathcal{N}^{src} \cup \mathcal{N}^{snk} \cup \mathcal{N}^{mix}$ and that $\mathcal{N}^{src}$, $\mathcal{N}^{snk}$ and $\mathcal{N}^{mix}$ are pairwise disjoint. For $N \subseteq \mathcal{N}$, let $N^{src} = N \cap \mathcal{N}^{src}$, $N^{snk} = N \cap \mathcal{N}^{snk}$, and $N^{mix} = N \cap \mathcal{N}^{src}$.

Throughout the paper, we assume a fixed, nonempty data domain *Data* consisting of the data items that can be transmitted through the channels. A data assignment for $N \subseteq \mathcal{N}$ means a function $\delta : N \rightarrow \mathit{Data}$. We write $\delta.A$ for the data item assigned to node $A \in N$ under δ. $DA(N)$ denotes the set of all data assignments for node-set N. If $M \subseteq N \subseteq \mathcal{N}$ and $\delta \in DA(N)$ then $\delta|_M$ denotes the data assignment for M that assigns data item $\delta.A$ to any $A \in M$.

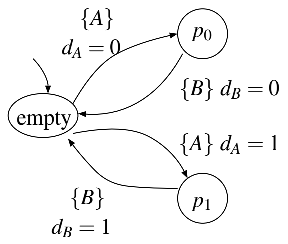
Constraint automata will use a symbolic representation of data assignments by data constraints which mean propositional formulae built from the atoms $d_A = d_B$ or $d_A \in P$ where A B are nodes, d_A is a symbol for the observed data item at node A and $d \in \mathit{Data}$, $P \subseteq \mathit{Data}$. If $P = \{d\}$ is a singelton then we simply write $d_A = d$ rather than $d_A \in \{d\}$. If g is a data constraint and $d \in \mathit{Data}$, then $g[d_A/d]$ denotes the data constraint obtained by syntactically replacing all occurrences of d_A in g with d. For $\emptyset \neq N \subseteq \mathcal{N}$, $DC(N)$ denotes the set of data constraints using only the symbols d_A for $A \in N$, but not d_B for $B \in \mathcal{N} \setminus N$. The symbol $\models$ stands for the obvious satisfaction relation which results from interpreting data constraints over data assignments.

Constraint automata. A constraint automaton is a tuple $\mathcal{A} = (Q, \mathcal{N}, \longrightarrow, Q_0)$ where Q is a set of states, also called configurations, $\mathcal{N}$ a finite set of nodes, disjointly partitioned into $\mathcal{N}^{src}$, $\mathcal{N}^{snk}$, and $\mathcal{N}^{mix}$, $\longrightarrow$ is a subset of $\bigcup_{N \subseteq \mathcal{N}} Q \times \{N\} \times DC(N) \times Q$, and $Q_0 \subseteq Q$ the set of initial states. We refer to $\longrightarrow$ as the transition relation of $\mathcal{A}$. We write $q \xrightarrow{N,g} p$ instead of $(q, N, g, p) \in \longrightarrow$ and refer to N as the node-set and g the guard of the transition. By an instance of $q \xrightarrow{N,g} p$ we mean a transition of the form $q \xrightarrow{N,\delta} p$ where $\delta \in DA(N)$ is a data assignment for the nodes in N with $\delta \models g$.

The intuitive behavior of a constraint automaton is as follows. The automaton starts in an initial state. If the current state is q then an instance $q \xrightarrow{N,\delta} p$ of the outgoing transitions from q is chosen nondeterministically, the corresponding I/O-operations are performed and the next state is p. A formalization of the possible (finite or infinite) observable data flow of a constraint automaton is obtained by the notion of a *run*. Given a constraint automaton $\mathcal{A} = (Q, \mathcal{N}, \longrightarrow, Q_0)$, a run for $\mathcal{A}$ denotes a (finite or infinite) sequence of consecutive transition instances $q_0 \xrightarrow{N_0,\delta_0} q_1 \xrightarrow{N_1,\delta_1} q_2 \xrightarrow{N_2,\delta_2} \ldots$ where $q_0 \in Q_0$. For finite runs we require that the last state q does not have any outgoing transition where the node-set N is contained in $\mathcal{N}^{mix}$. The requirement that runs are either infinite or end in a state that does not have any outgoing transition where the node-set only consists of mixed nodes can be understood as a *maximal progress assumption* for the mixed nodes. Recall that the enabledness of I/O operations through the mixed nodes is totally defined by the circuit.



The picture above shows the constraint automata for a synchronous channel, a P-filter and a P-producer with source node A and sink node B and a synchronous drain. Recall that a single channel can be viewed as a Reo circuit with two nodes representing its channel ends. Since these channels do not have a buffer or any other medium for storing messages, there is only one single state (configuration). Here and in the sequel, we skip the trivial guard true.



The picture above shows a constraint automaton for a FIFO1 channel with source node A and sink node B. We assume here that the data domain consists of two data items 0 and 1. The initial state (empty) stands for the configuration where the buffer is empty, while the states p_0 and p_1 represent the configurations where the buffer is filled with one of the data items.

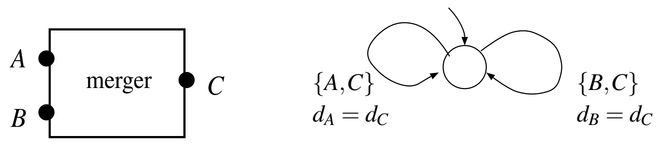
We now briefly summarize how Reo's join operation can be realized on constraint automata. The join of a source node with another (sink, source or mixed) node will be realized by a *product* construction, while joining sink nodes will be modelled with the help of a *merger*. We first consider the join operation for node-pairs $\langle A_i, B_i \rangle$ where in each pair at most one of the nodes is a sink or mixed node, that is, A_i or B_i is a source node. In this case, the effect of join is that all data flow at the nodes A_i and B_i agree. Thus, we may simply rename A_i into B_i and then apply a product construction for the corresponding constraint automata which synchronizes all I/O-operations for the common nodes. The product of $\mathcal{A}_1 = (Q_1, \mathcal{N}_1, \longrightarrow_1, Q_{0,1})$ and $\mathcal{A}_2 = (Q_2, \mathcal{N}_2, \longrightarrow_2, Q_{0,2})$ is $\mathcal{A}_1 \bowtie \mathcal{A}_2 = (Q_1 \times Q_2, \mathcal{N} \longrightarrow, Q_{0,1} \times Q_{0,2})$ where $N = \mathcal{N}_1 \cup \mathcal{N}_2$ and $\longrightarrow$ is defined by the rules:

$$\frac{q_1 \xrightarrow{N_1,g_1}_1 p_1, \quad q_2 \xrightarrow{N_2,g_2}_2 p_2, \quad N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1}{\langle q_1, q_2 \rangle \xrightarrow{N_1 \cup N_2, g_1 \wedge g_2} \langle p_1, p_2 \rangle}$$

$$\frac{q_1 \xrightarrow{N,g}_1 p_1, \quad N \cap \mathcal{N}_2 = \emptyset}{\langle q_1, q_2 \rangle \xrightarrow{N,g} \langle p_1, q_2 \rangle} \qquad \frac{q_2 \xrightarrow{N,g}_2 p_2, \quad N \cap \mathcal{N}_1 = \emptyset}{\langle q_1, q_2 \rangle \xrightarrow{N,g} \langle q_1, p_2 \rangle}$$

The source nodes in the product are the nodes in $\mathcal{N}^{src} = \mathcal{N}_1^{src} \setminus (\mathcal{N}_2^{mix} \cup \mathcal{N}_2^{snk}) \cup \mathcal{N}_2^{src} \setminus (\mathcal{N}_1^{mix} \cup \mathcal{N}_1^{snk})$ where $\mathcal{N}_i^{src}$ denotes the set of source nodes in $\mathcal{A}_i$, $\mathcal{N}_i^{snk}$ the set of sink nodes in $\mathcal{A}_i$ and $\mathcal{N}_i^{mix}$ the set of mixed nodes in $\mathcal{A}_i$. Similarly, the sink nodes in $\mathcal{A}_1 \bowtie \mathcal{A}_2$ are the nodes that are sinks in one of the automata, but not mixed or source nodes of the other automaton. All remaining nodes are mixed in $\mathcal{A}_1 \bowtie \mathcal{A}_2$. The sets of sink and mixed nodes in $\mathcal{A}_1 \bowtie \mathcal{A}_2$ will be denoted by $\mathcal{N}^{snk}$ and $\mathcal{N}^{mix}$, respectively. Note that $\bowtie$ is associative and commutative up to isomorphism.

Let us now consider the case of a join operation that combines two nodes, say A and B, where none of them is a source node. Then, we have to take care about the merge semantics of the resulting new sink or mixed node C, which essentially means that the read operations at A and B have to be interleaved, but they cannot occur at the same moment. For this, we use a *merger* which we join with the constraint automata that contain A and B, respectively. The merger can be viewed as a component connector with two input ports (source nodes) A and B and one output port (sink node) C. Formally, it is modelled by a single-state constraint automaton as illustrated in the picture below.



Note that this merger is inherent in the semantics of sink and mixed nodes and need not be used explicitely when generating component connectors in Reo.

The effect of hiding a mixed node in a Reo circuit is that data flow at that node is no longer observable from outside. We depart here from the approach of [8] and simply remove the hidden nodes from the node-sets of all transitions. Let $\mathcal{A} = (Q, \mathcal{N}, \longrightarrow, Q_0)$ be a constraint automaton and $M \subseteq \mathcal{N}^{mix}$. The constraint automaton $\mathsf{hide}(\mathcal{A}, M)$ is the tuple $\big(Q, \mathcal{N} \setminus M, \longrightarrow_M, Q_{0,M}\big)$ where $q \xrightarrow{\bar{N},\bar{g}}_M p$ iff there exists $q \xrightarrow{N,g} p$ where $\bar{N} = N \setminus M$ and $\bar{g} = \exists M[g]$. Here, $\exists\{A_1, \ldots, A_n\}[g] = \bigvee_{d_1,\ldots,d_n \in Data} g[d_{A_i}/d_i]$. The classification of the nodes in $\mathcal{N} \setminus M$ into sink, source and mixed nodes remains unchanged.

Clearly, $\exists M[\mathcal{P}] = \mathcal{P}$ if none of the nodes $A \in M$ appears in the labels of $\mathcal{P}$'s transitions. There is no general distributive law for the hiding and product operator. If, however, A is in $\mathcal{P}_1$'s node-set, but not in the node-set of $\mathcal{P}_2$ then $\exists\{A\}[\mathcal{P}_1 \bowtie \mathcal{P}_2] = \exists\{A\}[\mathcal{P}_1] \bowtie \mathcal{P}_2$.

3 Simple probabilistic constraint automata

The basic channel types explained above assume a perfect, totally reliable behavior of the channels. However, certain channels might behave *unreliably*, e.g. they may loose or corrupt messages in an uncontrolled way. In fact, such lossy channels play a crucial role in the context of so-called *lossy channel systems* [24, 17, 4, 3, 38] and probabilistic variants thereof [33, 13, 38, 14, 35, 15, 1, 2]. The behavior of unreliable channels is no longer deterministic. For instance, a FIFO channel might either fail or succeed to store a certain data item d in its buffer when its source end performs the operation $\mathsf{write}(d)$. We assume here that the degree of reliability of the used channel types has been studied through statistical experiments to obtain (approximations of) the probabilities for the faulty behaviors. Our goal is now to provide operational models for Reo circuits built out of perfect and unreliable channels with known failure probabilities, such as:

- a faulty FIFO channel with source node A and sink node B which behaves as a perfect FIFO channel, except that the write operations performed by A fail with probability τ. Thus, the written element will be stored correctly in the buffer with probability $1-\tau$. Such unreliable FIFO channels have been considered in the context of probabilistic lossy channel systems, e.g. in [1] with unbounded capacity or in the modeling language ProbMela [11].
- a lossy FIFO channel that might loose each stored data item with some fixed probability τ in any step. Such channels have been studied in the context of probabilistic lossy channel systems [14, 35, 15, 2].
- an unreliable synchronous channel which – as in the case of a perfect synchronous channel – requires that writing and reading at its channel ends are performed synchronously at the same moment, but the message might be corrupted. More precisely, if A is the source node and B the sink node then A and B are forced to synchronize their write and read operations, but the written data item by A differs from the obtained data item at B with some probability τ. A non-probabilistic variant of such unreliable synchronous channel are used in [25] for modelling the Alternating Bit Protocol.

In the former two examples (the two variants of a faulty FIFO channels) the question for an operational model is simpler since here the possible failures only affect the configurations reached after an I/O-operation, but not the observable data flow at the nodes performing the I/O-operations. The third example, the unreliable synchronous channel, is more difficult to model since its faulty behaviors have to be formalized in a data-dependent way, to reason about the probabilities for observing certain data items. We first consider the case of configuration failures which we model by so-called simple probabilistic constraint automata. These are close to simple probabilistic automata [39], which can be viewed as generalizations of reactive probabilistic systems [28, 42].

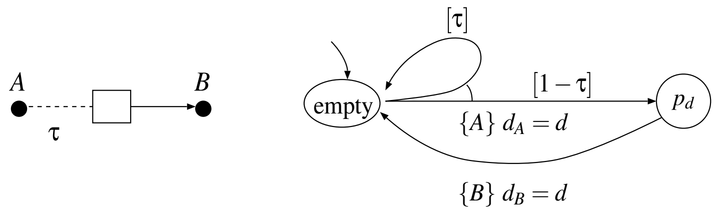
If Q is a countable set then a *distribution* for Q denotes a function $\pi : Q \rightarrow [0,1]$ such that $\sum_{q \in Q} \pi(q) = 1$. $\mathsf{Distr}(Q)$ denotes the set of distributions for Q.

Definition 1 Simple probabilistic constraint automata (SPCA).
A SPCA is a tuple $\mathcal{P} = (Q, \mathcal{N}, \longrightarrow, Q_0)$ where Q is a countable set of states, $Q_0 \subseteq Q$ the set of initial states, $\mathcal{N}$ a finite set of nodes (partitioned into $\mathcal{N}^{snk}$, $\mathcal{N}^{src}$ and $\mathcal{N}^{mix}$) and $\longrightarrow \subseteq \bigcup_{N \subseteq \mathcal{N}} Q \times \{N\} \times DC(N) \times \mathsf{Distr}(Q)$,. $\triangle$

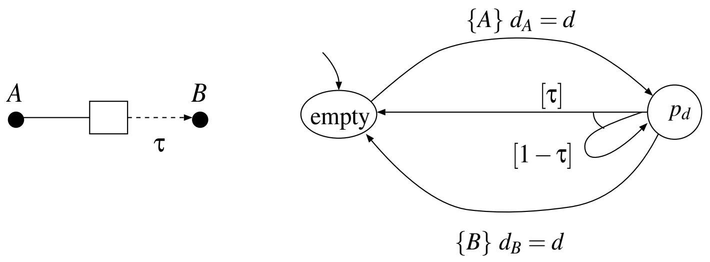
We write $q \xrightarrow{N,g} \pi$ instead of $(q, N, g, \pi) \in \longrightarrow$. Note that ordinary constraint automata can be regarded as special instances of simple probabilistic constraint automata where the distributions π in all transitions assign probability 1 to one successor state. The intuitive behavior of $\mathcal{P}$ in state q is similar as in ordinary constraint automata, the only difference being that the transitions can have a probabilistic effect. If $q \xrightarrow{N,\delta} \pi$ is the taken transition instance then the I/O-operations described by $\langle N, \delta \rangle$ are performed and the next configuration is p with probability $\pi(p)$.

Remark 1. The formalization of the possible infinite behaviors by means of runs is as for ordinary constraint automata. Thus, runs are maximal, consecutive transition instances, starting in an initial state. In the probabilistic setting, runs resolve both nondeterministic choices (which transition instance is selected in the current state) and probabilistic choices (which of the probabilistic alternatives of the selected transition instance is chosen). To reason about probabilities for certain runs, we have to resolve the concept of nondeterminism, but not the probabilistic choices. As it is standard in the theory of Markov decision processes, this is done by means of so-called *schedulers*, also often called policies, adversaries or strategies. We skip the details here which are irrelevant for the purpose of this paper and can be found e.g. in [34, 39]. A scheduler simply means an instance that resolves the nondeterminism, and thus, describes a possible behavior of a SPCA as a discrete-time stochastic process. (The notion discrete-time means stepwise behavior, but still in a time-abstract setting.) Any scheduler for an SPCA induces an infinite-state Markov chain where the states are finite prefixes of runs. The maximal paths in this Markov chain are the runs that can be generated by the scheduler. Using the standard probability measure of the paths in Markov chains, any scheduler induces a probabilitity measure on its runs and allows to speak about the probabilities for conditions on the runs, e.g., to reach eventually a certain configuration, or that infinitely often nodes A and B will synchronize. For verifying quantitative properties, such as *the probability to reach configuration q is at least 0.95*, one typically quantifies over all schedulers. This corresponds to a worst-case analysis where a certain condition is required to hold even under the worst scheduler. As for the analysis of non-probabilistic distributed systems, the verification of liveness properties often requires appropriate fairness conditions. In the probabilistic setting, these can be formalized by means of certain fair schedulers that, e.g., only generate runs where no continuously enabled transition is ignored forever. △

Example 1 SPCA for faulty FIFO1 channel. A faulty FIFO1 channel that might loose messages while inserting them into the buffer can be modelled by a simple probabilistic constraint automaton as illustrated in the following picture.



For simplicity, the picture only shows the case where $Data = \{d\}$. For larger data domains, there is a state p_d for any data item d. Here and in the sequel, we write the probabilities in brackets $[\ldots]$ and skip the probabilities from deterministic transitions, i.e., transitions where the corresponding distribution assigns probability 1 for one successor state. (This is the case for the transition from p_d to state empty.) Since all probabilistic branches of a SPCA-transition have the same label (node-set and data-assignment) only one of the probabilistic branches is labeled in the pictures of SPCA. The idea is that in the initial state empty any write-operation on A might fail with probability τ in which case the buffer remains empty or might be successful with probability $1 - \tau$.



The above picture shows a faulty FIFO1 channel that might loose messages from its buffer, but works perfectly for the write operation. In state p_d we have two transitions: one with the node-set $\{B\}$ and guard $d_B = d$ representing the case where B takes the element d from the buffer and one transition labelled with the empty node-set representing the case where the message in the buffer will be lost with probability τ and stays unchanged in the buffer with probability $1 - \tau$. The SPCA-semantics assumes a discrete-time domain where the loss of a stored message occurs with probability τ in each step (unless B takes the message). Thus, if A writes and B waits very long to take the message (i.e., we consider a scheduler that schedules the $\{A\}$-transition in state empty and then several times the transition labelled with the empty set in state p_d) then it is very likely that the message will be lost. For the extreme case where B never performs the take-operation the message will be lost with probability $\tau + (1 - \tau)\tau + (1 - \tau)^2\tau + (1 - \tau)^3\tau + \ldots \ = \ 1$. Assuming the $\{B\}$-transition is scheduled latest for the fourth step after state p_d has been entered, then the probability that a stored message will be eventually consumed by B is bounded by $1 - (\tau + (1 - \tau)\tau + (1 - \tau^2)\tau)) = (1 - \tau)^3$. The value $\tau + (1 - \tau)\tau + (1 - \tau^2)\tau$ stands for the probability to loose the message in the first three steps where the transition with the empty node-set has been scheduled in state p_d. △

We now address the question of how to construct the automaton (SPCA) for a given Reo circuit where the meaning of the channel types is provided by SPCA. Since ordinary constraint automata are special instances of SPCA, the (nonprobabilistic) perfect channel types discussed in Section 2 can be used in addition to unreliable channel types with a SPCA-semantics as in Example 1. To capture the meaning of Reo's join operator we need a product operator for SPCA which we then may use in combination with an appropriate node-renaming and mergers as explained in Section 2.1. Thus, we may assume that the common nodes are the nodes where data flow has to be synchronized and that each common node is a sink or mixed node in at most one of the given SPCA.

Definition 2 Product of SPCA. The product-automaton of the two SPCA $\mathcal{P}_1 = (Q_1, \mathcal{N}_1, \longrightarrow_1, Q_{0,1})$ and $\mathcal{P}_2 = (Q_2, \mathcal{N}_2, \longrightarrow_2, Q_{0,2})$ where $\mathcal{N}_1 \cap \mathcal{N}_2 \subseteq \mathcal{N}_1^{src} \cup \mathcal{N}_2^{src}$ is given by: $\mathcal{P}_1 \bowtie \mathcal{P}_2 = (Q_1 \times Q_2, \mathcal{N}_1 \cup \mathcal{N}_2, \longrightarrow, Q_{0,1} \times Q_{0,2})$ where $\longrightarrow$ is defined by the following rules:

$$\frac{q_1 \xrightarrow{N_1, g_1}_1 \pi_1, \quad q_2 \xrightarrow{N_2, g_2}_2 \pi_2, \quad N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1}{\langle q_1, q_2 \rangle \xrightarrow{N_1 \cup N_2, g_1 \wedge g_2} \pi_1 * \pi_2}$$

where $(\pi_1 * \pi_2)(\langle p_1, p_2 \rangle) = \pi_1(p_1) \cdot \pi_2(p_2)$ and

$$\frac{q_1 \xrightarrow{N,g}_1 \pi_1, \quad N \cap \mathcal{N}_2 = \emptyset}{\langle q_1, q_2 \rangle \xrightarrow{N,g} \pi_1'} \quad \text{where} \quad \pi_1'(\langle p_1, p_2 \rangle) = \begin{cases} \pi_1(p_1) & : \text{if } q_2 = p_2 \\ 0 & : \text{if } q_2 \neq p_2 \end{cases}$$

and latter's symmetric rule. The classification of the nodes into sink, source and mixed nodes in $\mathcal{P}_1 \bowtie \mathcal{P}_2$ is as in the non-probabilistic case (see Section 2.1). $\triangle$

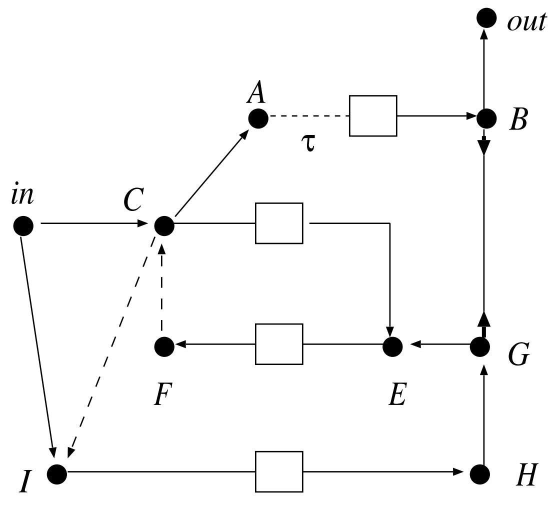
The product $\bowtie$ for SPCA is a conservative extension of the product for ordinary (non-probabilistic) constraint automata. Hence, for Reo circuits that are built from perfect and probabilistic channels, the automaton for the subcircuits consisting of perfect channels can be created by the join operator for ordinary constraint automata. Moreover, $\bowtie$ on SPCA is associative.

The hiding operator for SPCA can be defined as in ordinary constraint automata by just skipping the hidden nodes in the node-sets of the transitions and adapting the guards.

Definition 3 Hiding for SPCA. Let $\mathcal{P} = (Q, \mathcal{N}, \longrightarrow, Q_0)$ be a SPCA and $M \subseteq \mathcal{N}^{mix}$. The constraint automaton $\mathsf{hide}(\mathcal{A}, M)$ is the tuple $\big(Q, \mathcal{N} \setminus M, \longrightarrow_M, Q_{0,M}\big)$ where the transition relation $\longrightarrow_M$ is given by:

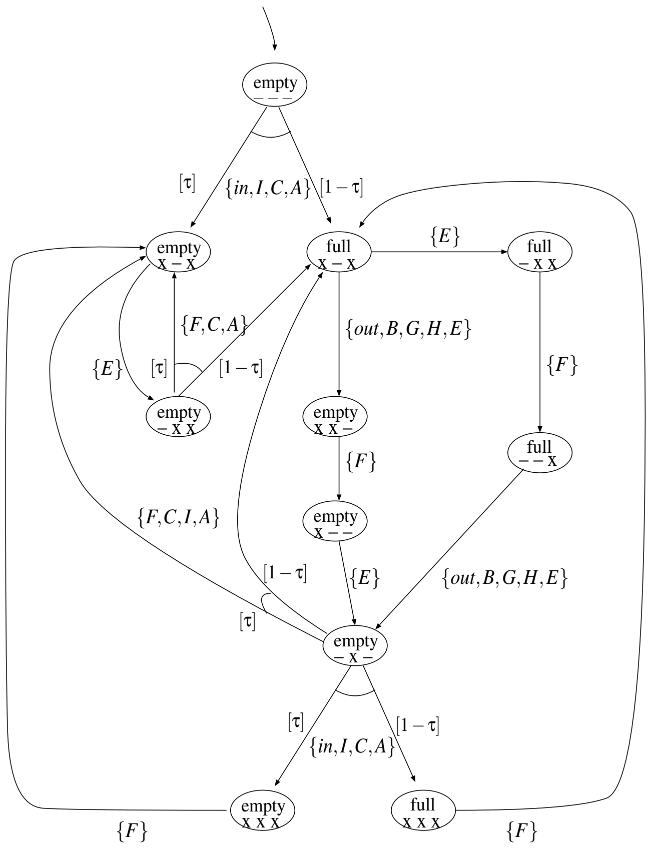
$$\frac{q \xrightarrow{N,g} \pi, \quad \bar{N} = N \setminus M, \quad \bar{g} = \exists M[g]}{q \xrightarrow{\bar{N}, \bar{g}}_M p}$$

Example 2 Distributed components that communicate via faulty channels.
Let us assume that we are given components Comp_1 and Comp_2 in a distributed system where messages from component Comp_1 to Comp_2 have to be transmitted along a FIFO1 channel with faulty write operation.



The picture shows a Reo circuit that organizes several further channels to obtain a component connector which repeats Comp_1's message as often as necessary, to ensure that almost surely Comp_2 will obtain the message. Comp_1 can deliver its messages to the circuit via connecting to node *in*, while Comp_2 might connect to node *out* to obtain the message from the circuit. To keep the example small, we assume here for simplicity that the receipt of the message can be acknowledged by sending a signal along the reliable synchronous drain BG. FIFO1 channels CE and EF in the middle and lossy synchronous channel FC serve to resend the message obtained via *in* along faulty FIFO channel AB, until the message has been stored in which case A is no longer enabled to take a message from channel CA. The lossy synchronous channel FC serves to empty the buffer between E and F if both buffers in the middle are filled. The lossy synchronous channel CI is needed to refill the lower buffer. Channels BG, GE and HG are used to stop data flow through the two buffers in the middle if the message has been taken at node *out*.

The SPCA for the above circuit is obtained by joining the automata for all involved channels and mergers to model the semantics of nodes E and I:



Data constraints have been skipped to simplify the picture. We used triples $uvw \in \{–, x\}^3$ for the names of states. E.g., state x – – means the configuration where the buffer between C and E is filled (the symbol x), while the two lower buffers are empty (symbol –).

Let us assume that Comp_2 is continuously willing to take a message from *out*. The quantitative property stating that any message from Comp_1 (that has been submitted to the circuit via port *in*) will be delivered with probability at least $1 - \tau^n$ after n repetitions of the message along channel AB holds for all schedulers. We now look for the property stating that any message sent by Comp_1 via port *in* will eventually be consumed by Comp_2. This does not hold for any run, since channel AB might always fail to write the message in the buffer. However, almost surely the writing operation at A will be successful after sufficiently many repetitions. Assuming a fair scheduler which rules out the possibility that B and *out* are waiting forever to take the message stored in the upper buffer by ignoring the corresponding transitions (in which case the data element runs continuously through the two buffers in the middle), the property stating that Comp_2 will obtain any message from Comp_1 with probability 1 can be established. △

4 Probabilistic constraint automata

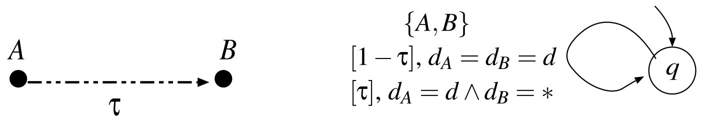
The definition of simple probabilistic constraint automata only treats probabilistic choices over configurations. SPCA are mostly appropriate to model various kinds of unreliable FIFO channels and connectors that are built from them and perfect channels. On the other hand, SPCA fail to model two other important aspects. First, SPCA cannot describe channels where synchronous I/O-operations might fail with some probability. For instance, a synchronous channel might fail to deliver the message written at the source end, instead a corrupted messages will be received at its sink end, although the channel still guarantees that any write operation at its source end is synchronized with a read operation at its sink end. In this case, failure of an I/O-operation does not mean that an unexpected configuration is reached, instead it affects the observable data flow at the sink end. Second, SPCA cannot model coin tossing actions which are required for a wide range of algorithms for resolving concurrency problems in distributed systems [29, 30], algorithms used in robotics [41] or in security protocols [36, 43]. To model such randomized coordination principles in the Reo framework, we need channels that provide random output values.

In this section, we provide the formal definition of a more general notion of a probabilistic constraint automaton that captures the meaning of channels where the observable I/O-operations occur with certain probabilities. This model is close to Segala's general probabilistic automata [39]. Transitions of probabilistic constraint automata have the form $q \longrightarrow \Pi$ where $\Pi \in \mathsf{Distr}\big(\bigcup_{N \subseteq \mathcal{N}} \{N\} \times DA(N) \times Q\big)$. That is, the effect of any step from q is formalized by a distribution that specifies probabilities for the I/O-operations and the successor configurations. The intuitive meaning of a transition $q \longrightarrow \Pi$ is that there is an agreement of all nodes A in $\mathcal{N}$ such that $\Pi(N, \delta, p) > 0$ for some triple (N, δ, p) with $A \in N$, to synchronize via the I/O-operations that occur under Π with positive probability and that involve A. An additional requirement is that for any transition $q \longrightarrow \Pi$ the source nodes and the data items to be written at them are fixed. This assumption is needed for technical reasons. The intuition is that the write operations determine the probabilistic effect of the transitions.

Definition 4 Probabilistic constraint automata (PCA). A PCA is a tuple $\mathcal{P} = (Q, \mathcal{N}, \longrightarrow, Q_0)$ where Q is a countable set of states, $Q_0 \subseteq Q$ the set of initial states, $\mathcal{N}$ is a finite set of nodes disjointly partitioned into $\mathcal{N}^{src}$, $\mathcal{N}^{snk}$, and $\mathcal{N}^{mix}$, and $\longrightarrow \subseteq Q \times \mathsf{Distr}\big(\bigcup_{N \subseteq \mathcal{N}} \{N\} \times DA(N) \times Q\big)$. We require that for any transition $q \longrightarrow \Pi$ we have: If $\Pi(N, \delta, p) > 0$ and $\Pi(M, \sigma, r) > 0$ then $N \cap \mathcal{N}^{src} = M \cap \mathcal{N}^{src}$ and $\delta.A = \sigma.A$ for all source nodes $A \in N \cap \mathcal{N}^{src}$. △

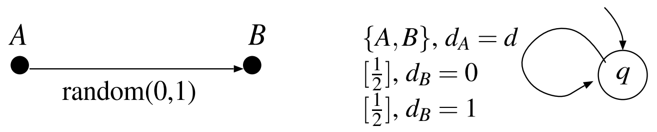
Any simple PCA can be regarded as a PCA. For this, each SPCA-transition $q \xrightarrow{N,g} \pi$ has to be split into the PCA-transitions $q \longrightarrow \mathit{\Pi}_{\langle N,\delta,\pi\rangle}$ for any $\delta \in DA(N)$ with $\delta \models g$. The distribution $\mathit{\Pi}_{\langle N,\delta,\pi\rangle}$ is given by $\mathit{\Pi}_{\langle N,\delta,\pi\rangle}(N,\delta,p) = \pi(p)$ and $\mathit{\Pi}_{\langle N,\delta,\pi\rangle}(\cdot) = 0$ in all other cases. Since ordinary constraint automata are special instances of SPCA, the general model of PCA also covers ordinary constraint automata. In particular, when modeling Reo circuits by probabilistic constraint automata all channel types that can be formalized with ordinary constraint automata or simple PCA can be used in conjunction with the channel types that have a (non-simple) PCA-semantics.

Example 3 A message-corrupting synchronous channel.
The picture below shows the probabilistic constraint automaton for a synchronous channel with source node A and sink node B where the delivered message is corrupted with probability τ. (As before on the left we suggest a Reo notation for this channel. The value τ serves as parameter for this channel type.) That is, if A writes data item d then with probability $1-\tau$ the correct value d is obtained at B, but with probability τ, B reads the symbol * that we use to denote a corrupted message.


A
B
τ
$\{A,B\}$
$[1-\tau]$, $d_A = d_B = d$
$[\tau]$, $d_A = d \wedge d_B = *$
q

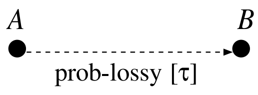
In the picture, d ranges over all data items. Formally, the PCA has for each $d \in \mathit{Data}$ a transition $q \longrightarrow \mathit{\Pi}_d$ where $\mathit{\Pi}_d(\{A,B\},[d_A = d, d_B = d], q) = 1-\tau$ and $\mathit{\Pi}_d(\{A,B\},[d_A = d, d_B = *], q) = \tau$. For all other pairs (N,δ) the probability for (N,δ,q) under $\mathit{\Pi}_d$ is 0. $\triangle$

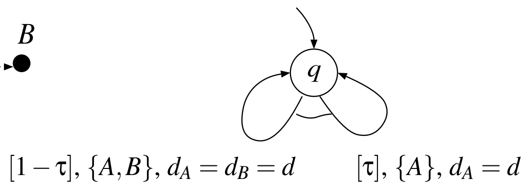
Example 4 Randomized synchronous channel. The Reo notation for a randomized synchronous channel and and its PCA-semantics is shown in the following picture.



When activated through an arbitrary writing action at its source A, the randomized synchronous channel generates a random number $b \in \{0,1\}$ which is synchronously taken by the sink B. In the picture, d ranges over all data items. That is, the sketched automaton has for each $d \in \mathit{Data}$ a transition $q \longrightarrow \mathit{\Pi}_d$ where $\mathit{\Pi}_d(\{A,B\},[d_A = d, d_B = 0], q) = \mathit{\Pi}_d(\{A,B\},[d_A = d, d_B = 1], q) = \frac{1}{2}$. $\triangle$

Example 5 Probabilistic lossy synchronous channel. In the previous two examples, the PCA-semantics only has transitions $q \longrightarrow \Pi$ where Π selects a unique node-set. An example for a channel with a PCA where different node-sets are selected with non-zero probabilities is a *probabilistic lossy synchronous channel.*



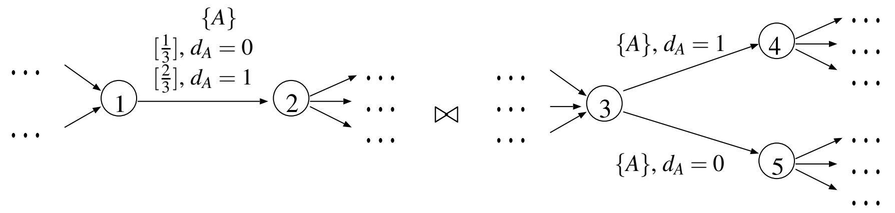


As an ordinary synchronous channels it has a sink end and a source end and requires both channel ends to be available to synchronize. However, the transmission of the message fails with a certain probability τ, while the correct message passing occurs with probability $1 - \tau$. This channel type has not to be confused with a non-probabilistic lossy synchronous channel (depicted by a dashed line without any subscript). The source end A of a non-probabilistic lossy synchronous channel can always write, while the probabilistic lossy synchronous channel requires A and B to agree on the potential handshaking. $\triangle$

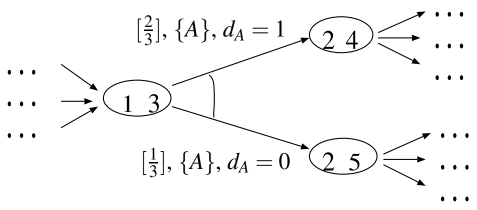
We now turn to the definition of the composition operators for PCA that capture the meaning of Reo's join and hiding operator. Let us start with the hiding operator. The following definition of the hiding operator on PCA is the obvious extension of the corresponding definition for ordinary or SPCA:

Definition 5 Hiding for PCA. Let $\mathcal{P} = (Q, \mathcal{N}, \longrightarrow, Q_0)$ be a PCA and $M \subseteq \mathcal{N}^{mix}$. Then, $\mathsf{hide}(\mathcal{P}, M) = \big(Q, \mathcal{N} \setminus M, \longrightarrow_M, Q_{0,M}\big)$ where $\longrightarrow_M$ is given by: $q \longrightarrow_M \Pi'$ iff $q \longrightarrow \Pi$ where $\Pi'(N', \delta', p) = \sum_{N,\delta} \Pi(N, \delta, p)$ with (N, δ) ranging over pairs where $N \setminus M = N'$ and $\delta|_{N'} = \delta'$. $\triangle$

The remainder of this section addresses the problem of defining an appropriate notion of the product of two PCA that covers the semantics Reo's join operator. Again, we retain the preprocessing by doing some renaming and the use of mergers for joining two mixed or sink nodes. Thus, we can concentrate on the case where we are given two PCA $\mathcal{P}_1 = (Q_1, \mathcal{N}_1, \longrightarrow_1, Q_{0,1})$ and $\mathcal{P}_2 = (Q_2, \mathcal{N}_2, \longrightarrow_2, Q_{0,2})$ that have to synchronize the I/O-operations for all common nodes $A \in \mathcal{N}_1 \cap \mathcal{N}_2$ and where each common node $A \in \mathcal{N}_1 \cap \mathcal{N}_2$ is a source node in $\mathcal{P}_1$ or $\mathcal{P}_2$. Unfortunately, the product for PCA is more complicated than for SPCA. The difficulty is that $\mathcal{P}_1$ and $\mathcal{P}_2$ might have to combine several transitions to complement each others transitions with common nodes (not just two as it is the case for SPCA). Look at the PCA-fragments shown in the picture below, where A is a sink or mixed node in the automaton $\mathcal{P}_1$ on the left and a source node in the automaton $\mathcal{P}_2$ on the right. Transition $1 \longrightarrow_1 \Pi_1$ of $\mathcal{P}_1$ cannot be matched with a single transition of $\mathcal{P}_2$.



However, $\mathcal{P}_1$ and $\mathcal{P}_2$ can agree to have data flow at A: $\mathcal{P}_1$ provides the probabilities for the concrete data item, while $\mathcal{P}_2$ reacts on this choice made by $\mathcal{P}_1$ and chooses the lower or upper transition depending on the outcome of $\mathcal{P}_1$'s probabilistic choice. This is consistent with the meaning of joining sink/mixed node A of $\mathcal{P}_1$ with source node A in $\mathcal{P}_2$: the data items obtained from a channel with sink end on A are transmitted via all channels with a source end on A.



$[\frac{2}{3}], \{A\}, d_A = 1$
$[\frac{1}{3}], \{A\}, d_A = 0$
1 3
2 4
2 5

In fact, this product-PCA arises by the concatenation operator of Def. 6 which relies on the assumption that the common nodes are exactly the source nodes in $\mathcal{P}_2$. Under these assumptions, joining $\mathcal{P}_1$ and $\mathcal{P}_2$ can be viewed as a concatenation operation since $\mathcal{P}_1$ provides the inputs for $\mathcal{P}_2$.

Definition 6 Concatenation of PCA (special case of PCA-product). Let $\mathcal{P}_i = (Q_i, \mathcal{N}_i, \longrightarrow_i, Q_{0,i})$, $i = 1, 2$, be two PCA such that $\mathcal{N}_1 \cap \mathcal{N}_2$ agrees with the set of source nodes in $\mathcal{P}_2$. Let $q_1 \in Q_1$ and $q_2 \in Q_2$. Then, a combined distribution of $\langle q_1, q_2 \rangle$ is a function Π which is obtained as follows. For $N \subseteq \mathcal{N}_1 \cup \mathcal{N}_2$, $\delta \in DA(N)$ and $p_1 \in Q_1$, $p_2 \in Q_2$, $\Pi(N, \delta, \langle p_1, p_2 \rangle)$ is given by

$$\Pi_2^{N_1 \cap \mathcal{N}_2^{src}, \delta|_{N_1 \cap \mathcal{N}_2^{src}}}(N \cap \mathcal{N}_2, \delta|_{N \cap \mathcal{N}_2}, p_2)$$

where $q_1 \longrightarrow \Pi_1$ is a transition on $\mathcal{P}_1$ and, for each pair $(M, \sigma) = (N_1 \cap \mathcal{N}_2, \delta|_{N_1 \cap \mathcal{N}_2})$ where $\Pi_1(N_1, \delta_1, p_1) > 0$ for some $p_1 \in Q_1$, $\Pi_2^{M,\sigma}$ is a distribution on $\bigcup_{N_2 \subseteq \mathcal{N}_2} \{N_2\} \times DA(N_2) \times Q_2$ such that one of the following two conditions holds:

(i) $q_2 \longrightarrow_2 \Pi_2^{M,\sigma}$ is a transition in $\mathcal{P}_2$ such that $\Pi_2^{M,\sigma}(N_2, \delta_2, p_2) > 0$ implies $M = N_2 \cap \mathcal{N}_1$ and $\sigma.A = \delta_2.A$ for all $A \in M$.

(ii) $M = \emptyset$ and $\Pi_2^{M,\sigma} = Idle_{q_2}$ where $Idle_{q_2}$ denotes the distribution that assigns probability 1 to the triple $(\emptyset, \emptyset, q_2)$ and probability 0 to all remaining triples. ($Idle_{q_2}$ is called the idling distribution for q_2.)

The concatenation of $\mathcal{P}_1$ and $\mathcal{P}_2$ is the PCA $\mathcal{P}_1;\mathcal{P}_2 = (Q_1 \times Q_2, \mathcal{N}_1 \cup \mathcal{N}_2, \longrightarrow, Q_{0,1} \times Q_{0,2})$ where the transition relation is defined by the following two rules:

$$\frac{\Pi \text{ is a combined distribution of } \langle q_1, q_2\rangle \in Q_1 \times Q_2}{\langle q_1, q_2\rangle \longrightarrow \Pi}$$

and

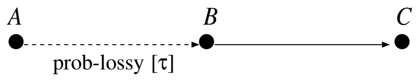
$$\frac{q_2 \longrightarrow_2 \Pi_2 \;\wedge\; (\Pi_2(N_2, \delta_2, p_2) > 0 \Rightarrow N_2 \cap \mathcal{N}_1 = \emptyset)}{\langle q_1, q_2\rangle \longrightarrow \Pi_2'}$$

In the latter rule Π_2' means the composition of the idling distribution for q_1 and distribution Π_2, that is, $\Pi_2'(N, \delta, \langle p_1, p_2\rangle) = \Pi_2(N, \delta, p_2)$ if $N \subseteq \mathcal{N}_2 \setminus \mathcal{N}_1$ and $p_1 = q_1$, and $\Pi_2'(N, \delta, \langle p_1, p_2\rangle) = 0$ in all other cases. $\triangle$

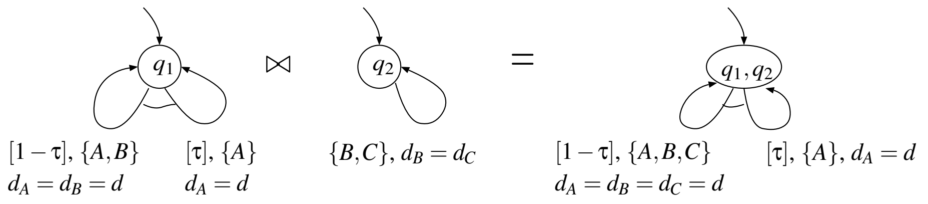
In Def. 6 the combined transitions arise by combining a transition $q_1 \longrightarrow_1 \Pi_1$ in $\mathcal{P}_1$ with several proper transitions in $\mathcal{P}_2$ (case (i) in Def. 6) or idling step in $\mathcal{P}_2$ (case (ii) in Def. 6), depending on the input values for the source nodes in $\mathcal{P}_2$ provided by $\mathcal{P}_1$ in the corresponding probabilistic branch of Π_1. The source nodes in $\mathcal{P}_2$ that are sink or mixed nodes in $\mathcal{P}_1$ are mixed in $\mathcal{P}_1 \bowtie \mathcal{P}_2$. The written values at the source ends of such a node A (modelled by transitions in $\mathcal{P}_2$) are obtained from the sink ends on A (modelled by a transition in $\mathcal{P}_1$). This explains the definition of combined distributions where the node-set $N_1 = N \cap \mathcal{N}_1$ and data assignment $\delta|_{N_1}$ determines the read-operations of $\mathcal{P}_1$'s sink or mixed nodes $A \in N_1 \cap \mathcal{N}_2^{src} \subseteq \mathcal{N}_1^{snk} \cup \mathcal{N}_1^{mix}$, while $\mathcal{P}_2$ has to provide matching write-operations at its source nodes $A \in N_1 \cap \mathcal{N}_2^{src}$ by means of a transition $q_2 \longrightarrow_2 \Pi_2^{N_1 \cap \mathcal{N}_2, \delta|_{N_1 \cap \mathcal{N}_2}}$. In particular, if $q_1 \longrightarrow_1 \Pi_1$ is a transition in $\mathcal{P}_1$ such that for some triple (N_1, δ_1, p_1) we have $\Pi_1(N_1, \delta_1, p_1) > 0$ but there is no matching transition in $\mathcal{P}_2$ according to condition (2), then there is no combined transition that relies on the combination of $q_1 \longrightarrow_1 \Pi_1$ with transitions in $\mathcal{P}_2$. This, however, corresponds to the join-operator for SPCA (or ordinary constraint automata) which also discards transitions of one automaton that involves common nodes and cannot be matched with a transition of the other automaton.

The second transition type $\langle q_1, q_2\rangle \longrightarrow \Pi_2'$ in Def. 6 covers the case where $\mathcal{P}_2$ performs transitions where none of the common nodes $A \in \mathcal{N}_1 \cap \mathcal{N}_2$ is involved. In this case, $\mathcal{P}_2$ performs its transition $q_2 \longrightarrow_2 \Pi$ without synchronizing with $\mathcal{P}_1$. The symmetric case where $\mathcal{P}_1$ performs a transition without interacting with $\mathcal{P}_2$ is covered by the combined transitions. It can be shown that any combined distribution Π is in fact a distribution (i.e., its values sum up to 1) and that the concatenation operator is associative Moreover, the concatenation operator of Def. 6 applied to SPCA (viewed as PCA) yields the product operator for SPCA (Def. 2). (See [10] for the proofs.)

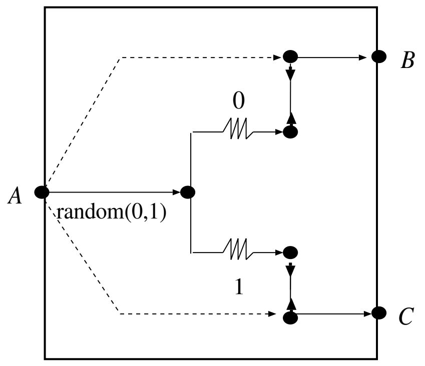
Example 6. To illustrate how the concatenation-operator can serve to model Reo's join operator, we consider a simple Reo circuit that arises by joining the sink node of a probabilistic lossy synchronous channel (cf. Example 5) with the source node of an ordinary synchronous channel.



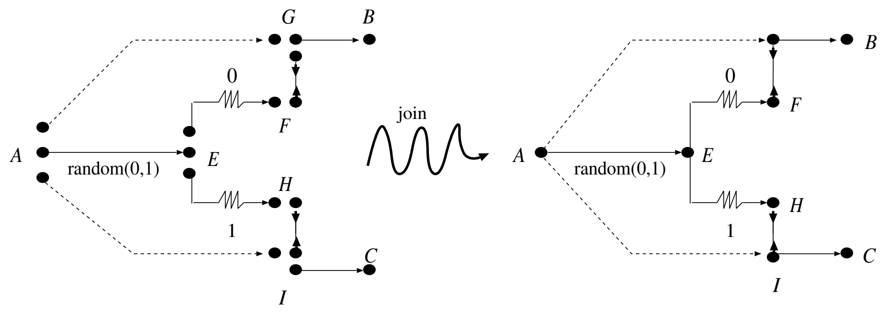
The PCA that results from joining the PCA for the probabilistic lossy synchronous channel AB and the synchronous channel BC via the concatenation-operator is shown in the picture below.



Example 7 Constructing a randomized router. We use the Reo framework with random synchronous channels to model a randomized router. This means a component with one source node A and two sinks B and C which randomly choses B or C (both with probability $\frac{1}{2}$) to obtain a data item written at A.



Beside two filter channels, two synchronous drains and the randomized synchronous channel, we use two lossy synchronous channels (depicted by dashed lines). As mentioned before, a lossy synchronous channel behaves as an ordinary synchronous channel but writing at its source end is always enabled. If the sink end is not available to take the written data item then the data item is lost. If A actives the randomized synchronous channel then, with equal probability, data items 0 or 1 serve as input for the two filter channels. Exactly one of the two filter channels delivers the randomly generated bit, while the other loses the written bit. On the other hand, the read operation at B has to be synchronized with data flow at the upper synchronous drain and the upper lossy synchronous channel. Moreover, data flow at the upper synchronous drain is possible only if the upper filter channel delivers 0. The same holds for C and the lower synchronous drain, lossy synchronous channel and data item 1. Hence, with equal probability A delivers its message to either B or C, provided that B and C are enabled to obtain a data item.



The above picture illustrates the construction that preceeds the hiding operator on the circuit-level. We then mimick these steps on the automata-level by first applying the join operator (product) to the involved non-probabilistic channels. The resulting automaton is then joined with the PCA for the randomized channel. Finally, we hide all mixed nodes which yields a formalization of the behavior described above. These steps are shown at the end of the paper (Fig. 2). $\triangle$

In the sequel, we discuss the general case where two PCA $\mathcal{P}_1$, $\mathcal{P}_2$ (where all common nodes are source nodes in $\mathcal{P}_1$ or $\mathcal{P}_2$) have to be combined via a product-construction that is consistent with Reo's join operator. To obtain a perfect match of the I/O-operations at the common nodes, the need to combine one transition in one automaton with several transitions in the other automaton can become cyclic when transitions are considered that involve common nodes $A \in \mathcal{N}_1^{snk} \cap \mathcal{N}_2^{src}$ and $B \in \mathcal{N}_1^{src} \cap \mathcal{N}_2^{src}$. In fact, we do not see any appropriate way to assign probabilities for the values obtained and written at nodes A and B. In our opinion, the most reasonable semantics for the composite circuit does not allow for any data flow since there is no possibility for the underlying subcircuits to agree on I/O-operations at their common nodes A and B and resolving the probabilistic choices in randomized channels independently.

In the sequel, we introduce a product-construction for PCA which relies on a combination of the concepts of the SPCA-product and the concatenation operator for PCA. The idea is that any step in the composite circuit (product-PCA on the semantic level) arises either by combining simple transitions where both circuits agree on the joint I/O-operations in a deterministic way or by combining a single transition of one circuit R_i where none of the sink or mixed nodes of the other circuit R_j is involved with several matching transitions of the other circuit R_j. Both types of transitions in the product can be interpreted as transitions that are initialized by one automaton, say $\mathcal{P}_1$, which offers to synchronize on certain common nodes by fixing I/O-operations for them. The other automaton $\mathcal{P}_2$ can react on the suggested synchronization by providing matching I/O-operations for the common nodes. If $\mathcal{P}_1$ offers I/O-operations with a probabilistic effect for some sink or moxed node in $\mathcal{P}_1$ that is a source node in $\mathcal{P}_2$ then $\mathcal{P}_2$'s can react on the outcome of $\mathcal{P}_1$'s probabilistic choice. Since PCA-transitions require unique write-operations for the source nodes, the only possibility for $\mathcal{P}_2$'s reaction on a transition offered by $\mathcal{P}_1$ where nodes in $\mathcal{N}_1^{src} \cap (\mathcal{N}_2^{snk} \cup \mathcal{N}_2^{mix})$ are involved is to provide a transition in $\mathcal{P}_2$ where the written values for the involved nodes $A \in \mathcal{N}_1^{src} \cap (\mathcal{N}_2^{snk} \cup \mathcal{N}_2^{mix})$ are deterministic.

Notation 7 I/O-determinism, simple combined transitions. Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two PCA as before where any common node $A \in \mathcal{N}_1 \cap \mathcal{N}_2$ is a source node in $\mathcal{P}_1$ or $\mathcal{P}_2$ (or both). Let $q_1 \longrightarrow_1 \varPi_1$ be a transition in $\mathcal{P}_1$. Then, $q_1 \longrightarrow_1 \varPi_1$ is called I/O-deterministic for $\mathcal{P}_2$ if $\varPi_1(N_1, \delta_1, p_1) > 0$ and $\varPi_1(N_1', \delta_1', p_1') > 0$ implies $N_1 \cap \mathcal{N}_2 = N_1' \cap \mathcal{N}_2$ and $\delta_1.A = \delta_1'.A$ for all $A \in N_1 \cap \mathcal{N}_2$. If $q_1 \longrightarrow_1 \varPi_1$ is I/O-deterministic for $\mathcal{P}_2$ then a matching transition for $q_1 \longrightarrow_1 \varPi_1$ in $\varPi_2$ is a transition $q_2 \longrightarrow_2 \varPi_2$ in $\mathcal{P}_2$ such that $\varPi_1(N_1, \delta_1, p_1) > 0$ and $\varPi_2(N_2, \delta_2, p_2) > 0$ implies $N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1$ and $\delta_1.A = \delta_2.A$ for all $A \in N_1 \cap \mathcal{N}_2$. If $q_1 \longrightarrow_1 \varPi_1$ is I/O-deterministic for $\mathcal{P}_2$ and $q_2 \longrightarrow_2 \varPi_2$ a matching transition then the transition $\langle q_1, q_2 \rangle \longrightarrow \varPi$ where

$$\varPi(N, \delta, \langle p_1, p_2 \rangle) = \varPi_1(N \cap \mathcal{N}_1, \delta|_{N \cap \mathcal{N}_1}, p_1) \cdot \varPi_2(N \cap \mathcal{N}_2, \delta|_{N \cap \mathcal{N}_2}, p_2)$$

is called a simple combined transition of $\langle q_1, q_2 \rangle$. $\triangle$

Note that if $q_1 \longrightarrow_1 \varPi_1$ is I/O-deterministic for $\mathcal{P}_2$ and $\varPi_2$ is as in in Notation 7 then $q_2 \longrightarrow_2 \varPi_2$ is I/O-deterministic for $\mathcal{P}_1$. From this, we can derive that $\varPi$ in Notation 7 is in fact a distribution. Moreover, any source node $A \in \mathcal{N}^{src}$ either has a fixed value to be written at A in $\varPi$'s probabilistic branches or A does not occur in any probabilistic branch of $\varPi$. Thus, simple combined transitions fulfill the requirements of PCA-transitions.

Notation 8 Input-independence, non-simple combined transitions. Let $\mathcal{P}_1$, $\mathcal{P}_2$ be two PCA as before where any common node $A \in \mathcal{N}_1 \cap \mathcal{N}_2$ is a source node in $\mathcal{P}_1$ or $\mathcal{P}_2$ (or both). Let $q_1 \longrightarrow_1 \varPi_1$ be a transition in $\mathcal{P}_1$. Then, $q_1 \longrightarrow_1 \varPi_1$ is called input-independent on $\mathcal{P}_2$ if $\varPi_1(N_1, \delta_1, p_1) > 0$ implies $N_1 \cap \mathcal{N}_1^{src} \cap (\mathcal{N}_2^{snk} \cup \mathcal{N}_2^{mix}) = \emptyset$. Let $q_1 \longrightarrow_1 \varPi_1$ is input-independent on $\mathcal{P}_2$ and $q_2 \in Q_2$. Furthermore, let $L \subseteq \mathcal{N}_2^{src} \setminus \mathcal{N}_1$ and $\eta \in DA(L)$. For any pair $(M, \sigma) = (N_1 \cap \mathcal{N}_2, \delta_1|_{N_1 \cap \mathcal{N}_2})$ for some triple (N_1, δ_1, p_1) where $\varPi_1(N_1, \delta_1, p_1) > 0$, let $\varPi_2^{M,\sigma}$ be a distribution such that one of the conditions (i) or (ii) holds:

(i) $q_2 \longrightarrow_2 \varPi_2^{M,\sigma}$ is a transition in $\mathcal{P}_2$ such that $\varPi_2^{M,\sigma}(N_2, \delta_2, p_2) > 0$ implies $N_2 \cap \mathcal{N}_2^{src} = (N_1 \cap \mathcal{N}_2^{src}) \cup L$, $\delta_2.A = \eta.A$ for all $A \in L$, $N_2 \cap \mathcal{N}_1 = M$ and $\delta_2.A = \sigma.A$ for all $A \in M$.

(ii) $M = L = \emptyset$ and $\mathit{\Pi}_2^{M,\sigma} = \mathit{Idle}_{q_2}$

Then, $\langle q_1, q_2\rangle \longrightarrow \mathit{\Pi}$ is called a combined transition of $\langle q_1, q_2\rangle$ initialized by q_1 where $\mathit{\Pi}$ is defined as follows. For $N \subseteq \mathcal{N}_1 \cup \mathcal{N}_2$, $\delta \in DA(N)$, $p_1 \in Q_1$, $p_2 \in Q_2$, $\mathit{\Pi}(N, \delta, \langle p_1, p_2\rangle)$ is $\mathit{\Pi}_1(N_1, \delta_1, p_1) \cdot \mathit{\Pi}_2^{N_1 \cap \mathcal{N}_2, \delta_1|_{N_1 \cap \mathcal{N}_2}}(N_2, \delta_2, p_2)$ where $N_i = N \cap \mathcal{N}_i$, $\delta_i = \delta|_{N_i}$, $i = 1, 2$. The combined transition $\langle q_1, q_2\rangle \longrightarrow \mathit{\Pi}$ is called non-simple if $L \neq \emptyset$ or $q_1 \longrightarrow_1 \mathit{\Pi}_1$ is not I/O-deterministic for $\mathcal{P}_2$. (Non-simple) combined transitions of $\langle q_1, q_2\rangle$ initialized by q_2 are defined in the symmetric way. A combined transition of $\langle q_1, q_2\rangle$ means a non-simple combined transition of $\langle q_1, q_2\rangle$ initialized by q_1 or q_2 or a simple combined transition of $\langle q_1, q_2\rangle$. $\triangle$

Since non-simple combined transitions rely on a fixed transition in $\mathcal{P}_1$ and a pair (L, η) that determines the source nodes in $\mathcal{N}_2^{src} \setminus \mathcal{N}_1$ at which values have to be written the write-operations at the source nodes in $\mathcal{N}^{src} = (\mathcal{N}_1^{src} \setminus (\mathcal{N}_2^{snk} \cup \mathcal{N}_2^{mix})) \cup (\mathcal{N}_2^{src} \setminus (\mathcal{N}_1^{snk} \cup \mathcal{N}_1^{mix}))$ are deterministic in $\mathit{\Pi}$, i.e., do not depend on the probabilistic branches of $\mathit{\Pi}$. Hence, combined transitions can serve as transitions in the product.

Definition 9 Product of PCA. Let $\mathcal{P}_1$, $\mathcal{P}_2$ be two PCA as before. The product of $\mathcal{P}_1$ and $\mathcal{P}_2$ is the PCA $\mathcal{P}_1 \bowtie \mathcal{P}_2 = (Q_1 \times Q_2, \mathcal{N}_1 \cup \mathcal{N}_2, \longrightarrow, Q_{0,1} \times Q_{0,2})$ where the outgoing transitions from state $\langle q_1, q_2\rangle$ in $\mathcal{P}_1 \bowtie \mathcal{P}_2$ are the combined transitions $\langle q_1, q_2\rangle \longrightarrow \mathit{\Pi}$ (defined in Notation 7 and 8). $\triangle$

The product on PCA is a conservative extension of the product on simple PCA and the concatenation operator on PCA.

Since the products treats $\mathcal{P}_1$ and $\mathcal{P}_2$ in a symmetric way, the product operator $\bowtie$ on PCA is commutative up to isomorphism. Unfortunately, $\bowtie$ is not associative. The reason is that if we are given three PCA $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ then $\mathcal{P}_1 \bowtie (\mathcal{P}_2 \bowtie \mathcal{P}_3)$ might contain a non-simple combined transition $\langle q_1, q_2, q_3\rangle \longrightarrow \mathit{\Pi}$ initialized by q_1 which relies on the combination of a transition in $\mathcal{P}_1$ that is input-independent on $\mathcal{P}_2$ with one or more non-simple combined transitions in $\mathcal{P}_2 \bowtie \mathcal{P}_3$ initialized by q_3. It might be the case that $\langle q_1, q_2, q_3\rangle \longrightarrow \mathit{\Pi}$ is not a transition in $(\mathcal{P}_1 \bowtie \mathcal{P}_2) \bowtie \mathcal{P}_3$. However, all other transitions in $\mathcal{P}_1 \bowtie (\mathcal{P}_2 \bowtie \mathcal{P}_3)$ are also transitions in $(\mathcal{P}_1 \bowtie \mathcal{P}_2) \bowtie \mathcal{P}_3$. Thus, the PCA-semantics of a Reo-circuit depends on the order in which the PCA for its subcircuits are generated. To provide a sensible PCA-semantics for arbitrary Reo circuits with perfect and probabilistic channels we suggest an extension of the binary product-operator $\bowtie$ for three or more PCA $\mathcal{P}_1, \ldots, \mathcal{P}_n$. The idea is that in $\bowtie_{1 \leq i \leq n} \mathcal{P}_i$ the outgoing combined transitions from state $\langle q_1, \ldots, q_n\rangle$ are obtained by fixing a permutation $i_1, \ldots, i_n$ of $1, \ldots, n$ and combining

– a single transition $q_{i_1} \longrightarrow_{i_1} \mathit{\Pi}_{i_1}$ in $\mathcal{P}_{i_1}$ which is input-independent on or I/O-deterministic for $\mathcal{P}_{i_2}, \ldots, \mathcal{P}_{i_n}$,

– several transitions $q_{i_2} \Longrightarrow_{i_2} \mathit{\Pi}_{i_2}^{M_1,\sigma_1}$ in $\mathcal{P}_{i_2}$ that are input-independent on or I/O-deterministic for $\mathcal{P}_{i_3}, \ldots, \mathcal{P}_{i_n}$ and that yield a perfect match for the I/O-operations provided by $\mathit{\Pi}_{i_1}$'s probabilistic choices,

– several transitions $q_{i_3} \Longrightarrow_{i_3} \mathit{\Pi}_{i_3}^{M_1,\sigma_1,M_2,\sigma_2}$ in $\mathcal{P}_{i_3}$ that are input-independent on or I/O-deterministic for $\mathcal{P}_{i_4}, \ldots, \mathcal{P}_{i_n}$ and that yield a perfect match for the I/O-operations provided by the $\mathit{\Pi}_{i_1}$'s and $\mathit{\Pi}_{i_2}$'s probabilistic choices,

and so on. Here, $\Longrightarrow_i$ extends $\longrightarrow_i$ by the idling steps $q \Longrightarrow_i \mathit{Idle}_q$. The precise definition of combined transitions follows the same pattern as in the binary case and is omitted here. This operator extends the binary operator $\bowtie$ defined in Def. 9 in the sense that $\bowtie_{i=1,2} \mathcal{P}_i = \mathcal{P}_1 \bowtie \mathcal{P}_2$, and hence, it fails to be associative. However, if a Reo circuit with n channels $c_1, \ldots, c_n$ is given then the PCA obtained by the product for the PCA for $c_1, \ldots, c_n$ and the required mergers yields a reasonable operational semantics for the circuit.

Similar difficulties for the definition of products of other probabilistic models are known in the context of parallel composition for Segala's general probabilistic automata [39]. One solution in this setting is to require simple transitions for all input actions as in I/O-automata [31, 44]. In Segala's approach, a simple transitions means a transition with a unique action label for all probabilistic branches. This is not an appropriate solution in the context of Reo, since we do not simply have one action label per transition, but node-sets that stand for several input/output actions.

5 Bisimulation on PCA

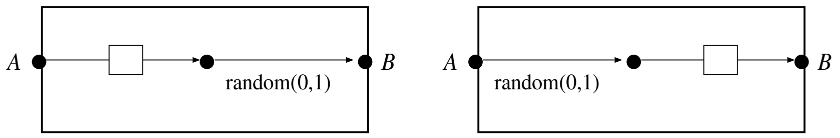
Formal notions of equivalence, such as bisimulation equivalence, provide a precise definition of what is meant that two probabilistic constraint automata, and thus two Reo circuits with perfect and probabilistic channels, have the same behavior. They play a crucial role for design purposes; for instance, when replacing a large (expensive) Reo circuit with a simpler (cheaper) one, the equivalence of the two circuits has to be ensured. Another example is the situation where the desired behavior of a component connector to be constructed is specified by a probabilistic constraint automata. We then may show the correctness of an implementation, given by a Reo circuit with probabilistic channels, by proving that the specification-PCA and the probabilistic constraint automaton for the constructed Reo circuit are bisimulation equivalent. The following definition of bisimulation equivalence for PCA is a slight variant of probabilistic bisimulation à la Larsen and Skou [28] and Segala and Lynch [40].

Definition 10 Bisimulation for PCA. Let $\mathcal{P}$ be a probabilistic constraint automaton as in Def. 4. A bisimulation on $\mathcal{P}$ is an equivalence relation $\mathcal{R}$ on the state space Q such that for all $(q_1, q_2) \in \mathcal{R}$ we have: For any transition

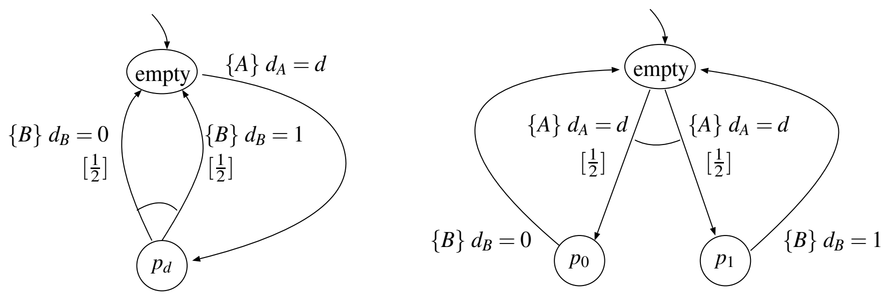
$q_1 \longrightarrow \Pi_1$ there is a transition $q_2 \longrightarrow \Pi_2$ such that $\Pi_1(N,\delta,P) = \Pi_2(N,\delta,P)$ for all $N \subseteq \mathcal{N}$, $\delta \in DA(N)$ and all equivalence classes $P \in Q/\mathcal{R}$. Here, we write $\Pi_i(N,\delta,P)$ for the sum $\sum_{p\in P} \Pi_i(N,\delta,p)$. Two states q_1, q_2 are called bisimilar, denoted $q_1 \sim q_2$, if (q_1,q_2) is contained in some bisimulation. $\triangle$

As usual for bisimulation-like relations, this notion of bisimulation equivalence for the states of one automaton can be adapted to a relation that compares two automata. If we are given two probabilistic constraint automata $\mathcal{P}_1$ and $\mathcal{P}_2$ with the same node-set then we consider the probabilistic constraint automaton $\mathcal{P}$ that arises through the disjoint union of $\mathcal{P}_1$ and $\mathcal{P}_2$. Then, $\mathcal{P}_1$ and $\mathcal{P}_2$ are called bisimulation equivalent if any bisimulation equivalence class of $\mathcal{P}$ either contains initial states of both automata or no initial states. We write $\mathcal{P}_1 \sim \mathcal{P}_2$ if $\mathcal{P}_1$ and $\mathcal{P}_2$ are bisimulation equivalent. As shown for bisimulation relation for other types of probabilistic automata (see e.g. [39]), it can be shown that if $(\mathcal{R}_i)_{i\in I}$ is a family of bisimulation relations on a PCA $\mathcal{P}$ then the $(\bigcup_i \mathcal{R}_i)^*$ is again a bisimulation on $\mathcal{P}$. This yields that $\sim$ as an relation on the state-space of $\mathcal{P}$ is the coarsest bisimulation on $\mathcal{P}$.

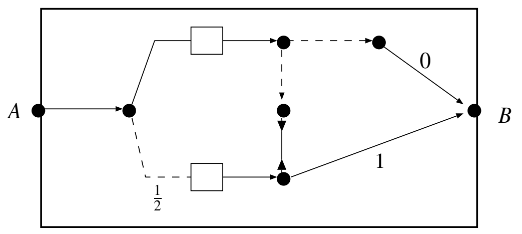
Example 8. We consider the two Reo circuits R_1 (left) and R_2 (right) shown in the picture below. For both, source node A is connected with sink node B via a randomized channel and a perfect FIFO1 channel. One might expect that R_1 and R_2 have the same meaning, since in either case the written data item at A is irrelevant for the obtained value at B which is 0 or 1 with equal probability.



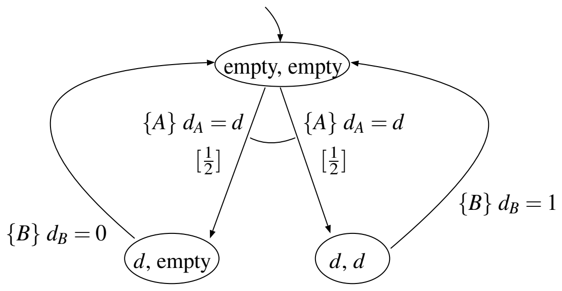
However, the PCA for R_1 and R_2 are not bisimulation equivalent, since in R_1 the $\{A\}$-transition is deterministic, while the $\{B\}$-transition is randomized, but the converse holds for R_2. The picture below shows the PCA $\mathcal{P}_1$ and $\mathcal{P}_2$ for R_1 and R_2, respectively. In the picture for $\mathcal{P}_1$, there is one state p_d and one outgoing transition from the initial state for any $d \in Data$.



Although all states p_d in $\mathcal{P}_1$ are bisimulation equivalent, they are not bisimulation equivalent to state p_0 or p_1 in $\mathcal{P}_2$ because of the data assignments for B in their outgoing transitions. Thus, $\mathcal{P}_1 \not\sim \mathcal{P}_2$.



Let us now regard the Reo circuit R_3 shown above. Any written value by A is simultaneously delivered to the (upper) perfect FIFO1 channel and the (lower) FIFO1 channel that loses the written value with probability $\frac{1}{2}$. In the configuration where both buffers are filled (which is reached with probability $\frac{1}{2}$), the element in the lower buffer enters the lower 1-producer channels and delivers data item 1 at node B, while the data element from the upper buffer enters the two lossy synchronous channels and will be lost in the upper one. In the configuration where only the upper buffer is filled (which is reached with probability $\frac{1}{2}$ when A provides its input), the data item enters both lossy channels, will be lost in the lower one, but will be delivered in the upper one and enters from there the 0-producer channel. Thus, B obtains the value 0.



The sketched behavior is formalized by the PCA for R_3, which can be obtained by applying the concatenation and hiding operator for PCA. The picture for $\mathcal{P}_3$ is parametric in d, that is, for all data items $d \in \mathit{Data}$, there are two states $\langle d, \text{empty}\rangle$ and $\langle d, d\rangle$ and there is one outgoing transition from the initial state with the label $\{A\}, d_A = d$. However, the states $\langle d, \text{empty}\rangle$, $d \in \mathit{Data}$, fall into the same bisimulation equivalence class. The same holds for the states $\langle d, d\rangle$, $d \in \mathit{Data}$. Hence, $\mathcal{P}_3$ and $\mathcal{P}_2$ are bisimulation equivalent. $\triangle$

Lemma 11 Compositionality of join and hiding for PCA. *If $\mathcal{P}$, $\mathcal{P}'$, $\mathcal{P}_1$, $\mathcal{P}_1'$, $\mathcal{P}_2$, $\mathcal{P}_2'$ are probabilistic constraint automata with the same node-set then:*

(a) $\mathcal{P}_1 \sim \mathcal{P}_1'$ *and* $\mathcal{P}_2 \sim \mathcal{P}_2'$ *implies* $\mathcal{P}_1 \bowtie \mathcal{P}_2 \ \sim\ \mathcal{P}_1' \bowtie \mathcal{P}_2'$.

(b) $\mathcal{P} \sim \mathcal{P}'$ *implies* $\mathsf{hide}(\mathcal{P}, M) \sim \mathsf{hide}(\mathcal{P}', M)$.

The proof for Lemma 11 can be provided using similar arguments as in the non-probabilistic setting [8] (see the extended version of this paper [10]).

6 Conclusion

The goal of the paper was to provide an operational semantics for probabilistic component connectors modelled by a Reo circuits. For this purpose, we introduced simple and general probabilistic constraint automata, presented operators for modelling Reo's join and hiding operations and provided a notion of bisimulation equivalence. While the join-operator for simple general probabilistic constraint automata is a natural extension of the join for non-probabilistic constraint automata, it causes difficulties for the general case. We introduced a product-construction that is appropriate for modeling join in non-cyclic probabilistic Reo circuits and thus can be viewed as a concatenation operator. While these operators are associative, and thus appropriate for compositional reasoning, their generalization for PCA fails to be associative. The problem with the join operator for PCA are cyclic input-dependencies via channels with a non-simple PCA-semantics. However, we suggested a product operator for two or more PCA that generalizes the SPCA-product and concatenation operator and can serve to provide a reasonable meaning for arbitrary Reo circuits with perfect and probabilistic channels.

Probabilistic constraint automata are slight variants of Markov decision processes, a well-known mathematical model for probabilistic systems. The techniques suggested in this paper provide the basis for applying known methods for a quantitative analysis of component connectors with unreliable or randomized channels. Model checking for temporal logic specifications and simulation techniques for probabilistic systems which are supported by various tools, e.g. PRISM [26], Möbius [20], Modest [16] or LiQuor [12], serve as examples.

Several extensions and variants of the presented concepts will have to be studied in future work. This includes the discussion of reward models to reason about costs, the combination of the probabilistic models discussed here with hard real-time constraints as in timed automata (e.g., in the style of probabilistic timed automata of [27]), an investigation of the relations of timed data streams (as it has been done in [9] for non-probabilistic Reo) in the probabilistic case and the connection to stochastic relations as discussed e.g. in [23] and the development of special models and logics to reason about Reo's dynamic features.

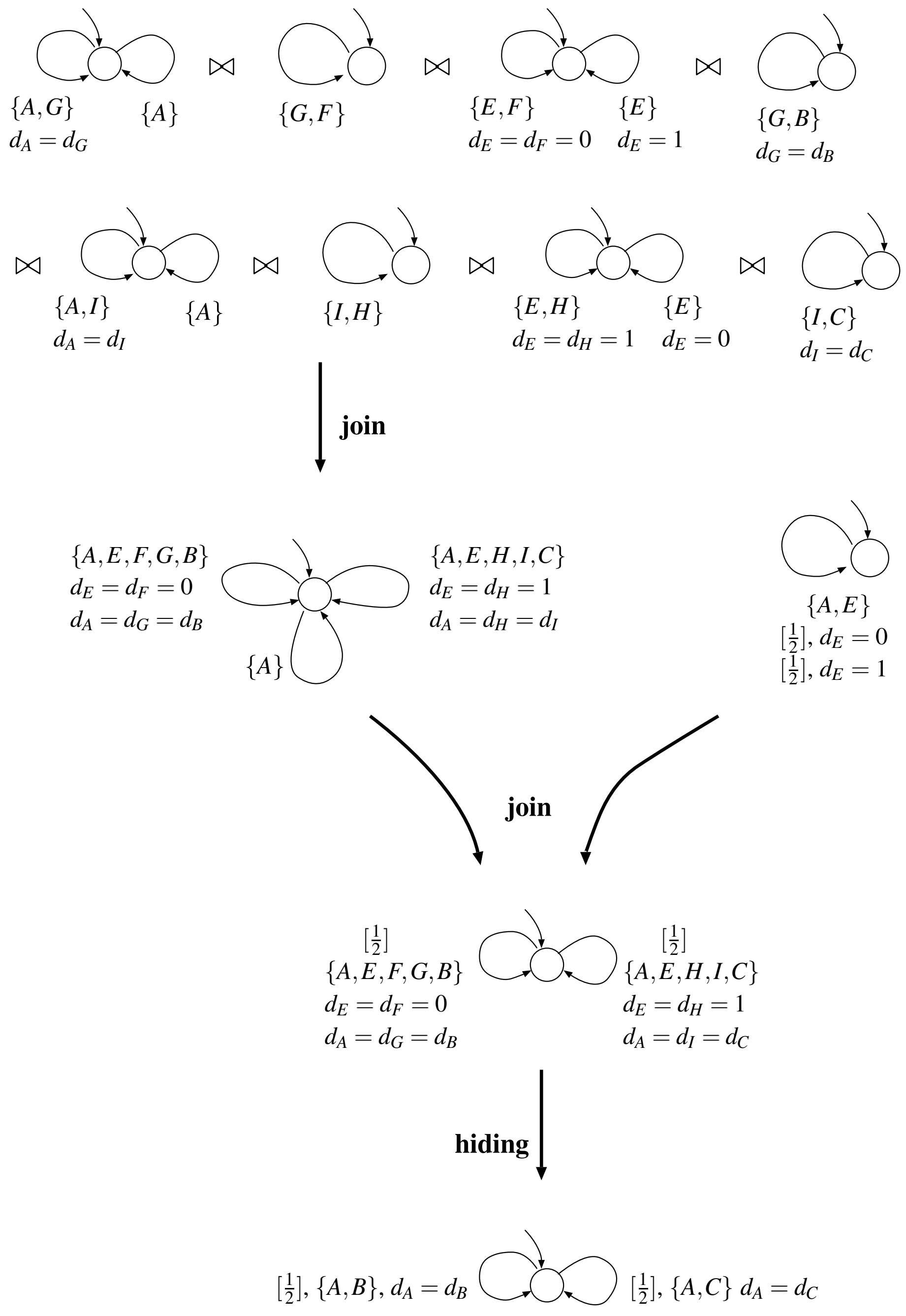

$\{A,G\}$ $d_A = d_G$
$\{A\}$
$\bowtie$
$\{G,F\}$
$\bowtie$
$\{E,F\}$ $d_E = d_F = 0$
$\{E\}$ $d_E = 1$
$\bowtie$
$\{G,B\}$ $d_G = d_B$
$\bowtie$
$\{A,I\}$ $d_A = d_I$
$\{A\}$
$\bowtie$
$\{I,H\}$
$\bowtie$
$\{E,H\}$ $d_E = d_H = 1$
$\{E\}$ $d_E = 0$
$\bowtie$
$\{I,C\}$ $d_I = d_C$
join
$\{A,E,F,G,B\}$ $d_E = d_F = 0$ $d_A = d_G = d_B$
$\{A,E,H,I,C\}$ $d_E = d_H = 1$ $d_A = d_H = d_I$
$\{A\}$
$\{A,E\}$ $[\frac{1}{2}]$, $d_E = 0$ $[\frac{1}{2}]$, $d_E = 1$
join
$[\frac{1}{2}]$ $\{A,E,F,G,B\}$ $d_E = d_F = 0$ $d_A = d_G = d_B$
$[\frac{1}{2}]$ $\{A,E,H,I,C\}$ $d_E = d_H = 1$ $d_A = d_I = d_C$
hiding
$[\frac{1}{2}]$, $\{A,B\}$, $d_A = d_B$
$[\frac{1}{2}]$, $\{A,C\}$ $d_A = d_C$

Figure 2: Compositional construction of the PCA for the randomized router

References

1. P. Abdulla, C. Baier, S. Purushothaman Iyer, and B. Jonsson. Simulating perfect channels with probabilistic lossy channels. *Inf. and Comp.*, 197(1–2):22–40, 2005.
2. P. Abdulla, N. Bertrand, A. Rabinovich, and P. Schnoebelen. Verification of probabilistic systems with faulty communication. submitted for publication.
3. P. A. Abdulla and B. Jonsson. Undecidable verification problems for programs with unreliable channels. *Information and Computation*, 130(1):71–90, 1996.
4. P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.
5. F. Arbab. Abstract behavior types: A foundation model for components and their composition. In *[21]*, pages 33–70, 2003.
6. F. Arbab. Reo: A channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):1–38, 2004.
7. F. Arbab, C. Baier, F. de Boer, and J. Rutten. Models and temporal logics for timed component connectors. In *Proc. SEFM'04*, 2004.
8. F. Arbab, C. Baier, J.J.M.M. Rutten, and M. Sirjani. Modeling component connectors in Reo by constraint automata. *Sc. of Comp. Progr.*, to appear, 2005.
9. F. Arbab and J.J.M.M. Rutten. A coinductive calculus of component connectors. In *Recent Trends in Algebraic Development Techniques, Proc. 16th WADT*, LNCS 2755, pages 35–56, 2003.
10. C. Baier. Probabilistic models for Reo connector circuits. Extended version, 2005. see http://web.informatik.uni-bonn.de/I/baier/publikationen.html.
11. C. Baier, F. Ciesinski, and M. Größer. ProbMela: a modeling language for communicating probabilistic systems. In *Proc. MEMOCODE*, pages 57–66, 2004.
12. C. Baier, F. Ciesinski, and M. Größer. ProbMela and Model Checking Markov decision processes. *Sig. Performance Evaluation Review* 32(4), pages 22-27, 2005.
13. C. Baier and B. Engelen. Establishing qualitative properties for probabilistic lossy channel systems: An algorithmic approach. In *Proc. ARTS*, LNCS 1601, pages 34–52, 1999.
14. N. Bertrand and Ph. Schnoebelen. Model checking lossy channels systems is probably decidable. In *Proc. FOSSACS*, LNCS 2620, pages 120–135, 2003.
15. N. Bertrand and Ph. Schnoebelen. Verifying nondeterministic channel systems with probabilistic message losses. In *Proc. Automated Verification of Infinite-State Systems (AVIS 2004)*, Electronic Notes in Theor. Comp. Sci., 2004. To appear.
16. H. Bohnenkamp, H. Hermanns, J.-P. Katoen, and R. Klaren. The modest modeling tool and its implementation. *Computer Performance Evaluation/TOOLS*, pages 116–133, 2003.
17. G. Cécé, A. Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, 1996.
18. CIM. http://www.almende.com/cim/.
19. D. Clarke, D. Costa, and F. Arbab. Modeling coordination in biological systems. In *Proc. of the Int. Symposium on Leveraging Applications of Formal Methods (ISoLA 2004)*, 2004.
20. D. Daly, D. D. Deavours, J. M. Doyle, A. J. Stillman, and P. G. Webster. Möbius: an extensible tool for performance and dependability modeling. In *Proc. FTCS*, pages 15–16, 1999.
21. F.S. de Boer, M.M. Bonsangue, S. Graf, and W.-P. de Roever, editors. *Formal Methods for Components and Objects*, LNCS 2852. Springer, 2003.
22. N. Diakov and F. Arbab. Compositional construction of web services using Reo. In *Proc. ICEIS*, Porto, 2004.
23. E.-E. Doberkat. The converse of a stochastic relation. *Journal of Logic and Algebraic Programming*, 62(1):133–154, 2005.
24. A. Finkel. Decidability of the termination problem for completely specificied protocols. *Distributed Computing*, 7(3):129–135, 1994.

25. W. Fokkink. *Introduction to Process Algebra.* Texts in Theoretical Computer Science, An EATCS Series. Springer-Verlag, 1999.
26. M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 2.0: A tool for probabilistic model checking. In *Proc. QEST*, pages 322–323, 2004.
27. M. Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282(1):101–150, 2002.
28. K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.
29. D. Lehmann and M. O. Rabin. On the advantage of free choice: A symmetric and fully distributed solution to the Dining Philosophers problem (extended abstract). In *Proc. POPL*, pages 133–138, 1981.
30. N. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.
31. N. Lynch and M.R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, 1989.
32. P. Panangaden and F. van Breugel, editors. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems.* CRM Monograph Series. American Mathematical Society, 2004.
33. S. Purushothaman Iyer and M. Narasimha. Probabilistic lossy channel systems. In *Proc. TAPSOFT* LNCS 1214, pages 667–681, 1997.
34. M. L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., New York, 1994.
35. A. Rabinovich. Quantitative analysis of probabilistic lossy channel systems. In *Proc. ICALP*, LNCS 2719, pages 1008–1021, 2003.
36. M. Reiter and A. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.
37. J.J.M.M. Rutten. Component connectors. In *[32]*, chapter 5, pages 73–87. American Mathematical Society, 2004.
38. Ph. Schnoebelen. The verification of probabilistic lossy channel systems. In *Validation of Stochastic Systems – A Guide to Current Research*, LNCS 2925, pages 445–465, 2004.
39. R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, Massachusetts Institute of Technology, 1995.
40. R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.
41. S. Thrun. Probabilistic algorithms in robotics. *AI Magazine*, 21(4):93–109, 2000.
42. R. van Glabbeek, S. Smolka, B. Steffen, and C. Tofts. Reactive, generative, and stratified models of probabilistic processes. In *Proc. LICS*, pages 130–141, 1990.
43. D. Volpano and G. Smith. Probabilistic noninterference in a concurrent language. In *Proc. PCSFW*, pages 34–43, 1998.
44. S.-H. Wu, S. Smolka, and E. Stark. Composition and behaviors of probabilistic I/O automata. *Theoretical Computer Science*, 176(1–2):1–38, 1997.
45. Z. Zlatev, N. Diakov, and S. Pokraev. Construction of negotiation protocols for E-Commerce applications. *ACM SIGecom Exchanges*, 5(2):11–22, November 2004.