Science of Computer Programming 80 (2014) 3–24

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

A compositional model to reason about end-to-end QoS in Stochastic Reo connectors

Young-Joo Moon [a,∗], Alexandra Silva [d,b,e], Christian Krause [c], Farhad Arbab [b]

[a] INRIA, Bordeaux, France
[b] Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands
[c] Hasso Plattner Institute (HPI), Potsdam, Germany
[d] Radboud University Nijmegen, Nijmegen, The Netherlands
[e] HASLab/INESC TEC, Universidade do Minho, Braga, Portugal

ARTICLE INFO

Article history:
Received 22 November 2010
Received in revised form 21 November 2011
Accepted 23 November 2011
Available online 1 December 2011

Keywords:
Coordination language
Reo
Continuous-ime Markov Chains
Quality of service
Compositional semantic model

ABSTRACT

In this paper, we present a compositional semantics for the channel-based coordination language Reo that enables the analysis of quality of service (QoS) properties of service compositions. For this purpose, we annotate Reo channels with stochastic delay rates and explicitly model data-arrival rates at the boundary of a connector, to capture its interaction with the services that comprise its environment. We propose Stochastic Reo Automata as an extension of Reo automata, in order to compositionally derive a QoS-aware semantics for Reo. We further present a translation of Stochastic Reo Automata to Continuous-Time Markov Chains (CTMCs). This translation enables us to use third-party CTMC verification tools to do an end-to-end performance analysis of service compositions. In addition, we discuss to what extent Interactive Markov Chains (IMCs) can serve as an alternative semantic model for Stochastic Reo. We show that the semantics of Stochastic Reo cannot be specified compositionally using the product operator provided by IMCs.

1. Introduction

In service-oriented computing (SOC), complex distributed applications are built by composing existing – often third-party – services using additional coordination mechanisms, such as workflow engines, component connectors, or tailor-made glue code. Due to the high degree of heterogeneity and the fact that the owner of the application is not necessarily the owner of its building blocks, issues involving quality of service (QoS) properties become increasingly entangled. Even if the QoS properties of every individual service and connector are known, it is far from trivial to determine and reason about the end-to-end QoS of a composed system in its application context. Yet, the end-to-end QoS of a composed service is often as important as its functional properties in determining its viability in its market.

Reo [1], a channel-based coordination language, supports the composition of services, and typically, its semantics is given in terms of Constraint Automata (CA) [2]. However, CA do not account for the QoS properties and cannot capture the context-dependency [2] of Reo connectors. To capture context-dependency, Reo Automata were introduced in [3]. However, they also provide no means for modeling QoS properties. On the other hand, Quantitative Intentional Automata (QIA) were proposed in [4] to account for the end-to-end QoS properties of Reo connectors. Unfortunately, no formal results are readily available regarding the compositionality of QIA. Thus, in order to overcome the shortcomings of CA and QIA, mentioned above, the design of a new compositional semantic model for Reo connectors was required.

∗ Corresponding author.
E-mail addresses: young-joo.moon@inria.fr (Y.-J. Moon), alexandra@cs.ru.nl (A. Silva), christian.krause@hpi.uni-potsdam.de (C. Krause), farhad@cwi.nl (F. Arbab).

0167-6423/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2011.11.007

For this purpose, in [5], we suggested *Stochastic Reo Automata* as a compositional semantic model for reasoning about the end-to-end QoS properties, as well as handling the context-dependency of Reo connectors. We showed that the compositionality results of Reo Automata extend to Stochastic Reo Automata. We also presented a translation of Stochastic Reo Automata to Continuous-Time Markov Chains (CTMCs). This enabled the use of third-party tools for stochastic analysis. Therefore, [5] shows a compositional approach for constructing Markov Chain (MC) models of complex composite systems, using Stochastic Reo Automata as an intermediate model. Stochastic Reo Automata provide a compositional framework wherein the corresponding CTMC model of a connector can be derived. This approach, thus, enabled us to model the QoS properties of system behavior, where our translation derives a CTMC model for complex systems for subsequent analysis by other tools. This paper is the extended version of [5] together with the contribution mentioned above. In this paper, we provide more examples for Stochastic Reo, its semantic model, and the translation method. We show the proof of the compositionality of Stochastic Reo Automata. In addition, we discuss to what extend Interactive Markov Chains (IMCs) can serve as an alternative semantic model for Stochastic Reo.

2. Overview of Reo

Reo is a channel-based coordination model wherein so-called *connectors* are used to coordinate (i.e., control the communication among) components or services exogenously (from outside of those components and services). In Reo, complex connectors are compositionally built out of basic channels. Channels are atomic connectors with exactly two ends, which can be either *source* or *sink* ends. Source ends accept data into, and sink ends dispense data out of their respective channels. Reo allows channels to be undirected, i.e., to have respectively two source or two sink ends.

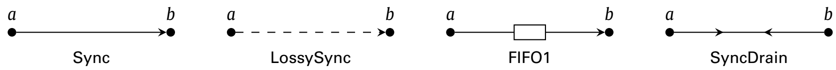


Fig. 1. Some basic Reo channels.

Fig. 1 shows the graphical representations of some basic channel types. The Sync channel is a directed, unbuffered channel that synchronously reads data items from its source end and writes them to its sink end. The LossySync channel behaves similarly, except that it does not block if the party at the sink end is not ready to receive data. Instead, it just loses the data item. FIFO1 is an asynchronous channel with a buffer of size one. The SyncDrain channel differs from the other channels in that it has two source ends (and no sink end). If there is data available at both ends, this channel consumes (and loses) both data items synchronously.

Channels can be joined together using nodes. A node can have one of three types: source, sink or mixed node, depending on whether all ends that coincide on the node are source ends, sink ends or a combination of both. Source and sink nodes, called *boundary nodes*, form the boundary of a connector, allowing interaction with its environment. Source nodes act as synchronous replicators, and sink nodes as mergers. A mixed node combines both behaviors by atomically consuming a data item from one sink end and replicating it to all of its source ends.

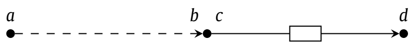


Fig. 2. Example connector: LossyFIFO1.

An example connector is depicted in Fig. 2. It reads a data item from a, buffers it in a FIFO1 and writes it to d. The connector loses data items from a if and only if the FIFO1 buffer is already full. This construct, therefore, behaves as a connector called (overflow) LossyFIFO1.

2.1. Semantics: Reo Automata

In this section, we recall Reo Automata [3], an automata model that provides a compositional operational semantics for Reo connectors. Intuitively, a Reo Automaton is a non-deterministic automaton whose transitions have labels of the form $g|f$, where f a set of nodes that fire synchronously, and g is a *guard* (boolean condition) that represents the presence or the absence of I/O requests at nodes, i.e., the pending status of the nodes. A transition can be taken only when its guard g is true.

We recall some facts about Boolean algebras. Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ be a set of symbols that denote names of connector ports, $\overline{\sigma}$ be the negation of σ, and $\mathcal{B}_\Sigma$ be the free Boolean algebra generated by the following grammar:

$$g \ ::= \ \sigma \in \Sigma \mid \top \mid \bot \mid g \vee g \mid g \wedge g \mid \overline{g}$$

We refer to the elements of the above grammar as *guards* and in its representation we frequently omit $\wedge$ and write g_1g_2 instead of $g_1 \wedge g_2$. Given two guards $g_1, g_2 \in \mathcal{B}_\Sigma$, we define a natural order $\leq$ as $g_1 \leq g_2 \iff g_1 \wedge g_2 = g_1$. The intended interpretation of $\leq$ is logical implication: g_1 implies g_2. An *atom* of $\mathcal{B}_\Sigma$ is a guard $a_1 \ldots a_k$ such that $a_i \in \Sigma \cup \overline{\Sigma}$ with $\overline{\Sigma} = \{\overline{\sigma}_i \mid \sigma_i \in \Sigma\}$, $1 \leq i \leq k$. We can think of an atom as a truth assignment. We denote atoms by Greek letters $\alpha, \beta, \ldots$

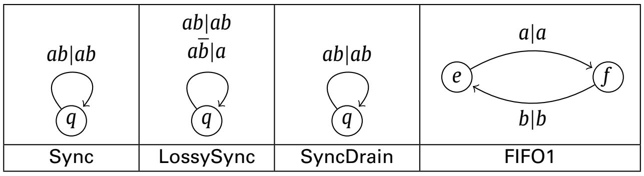


Fig. 3. Automata for the basic Reo channels of Fig. 1.

and the set of all atoms of $\mathcal{B}_\Sigma$ by $\mathbf{At}_\Sigma$. Given $S \subseteq \Sigma$, we define $\widehat{S} \in \mathcal{B}_\Sigma$ as the conjunction of all elements of S. For instance, for $S = \{a, b, c\}$ we have $\widehat{S} = abc$.

Definition 2.1 (*Reo Automaton [3]*)**.** A *Reo Automaton* is a triple (Σ, Q, δ) where Σ is the set of nodes, Q is the set of states, $\delta \subseteq Q \times \mathcal{B}_\Sigma \times 2^\Sigma \times Q$ is the finite transition relation such that for each $\langle q, g, f, q'\rangle \in \delta$, which is represented as $q \xrightarrow{g|f} q' \in \delta$:
(i) $g \leq \widehat{f}$ (reactivity)
(ii) $\forall g \leq g' \leq \widehat{f} \cdot \forall \alpha \leq g' \cdot \exists q \xrightarrow{g''|f} q' \in \delta \cdot \ \alpha \leq g''$ (uniformity)

In Reo Automata, for simplicity we abstract data constraints [2] and assume they are *true*.

Intuitively, a transition $q \xrightarrow{g|f} q'$ in an automaton corresponding to a Reo connector conveys the following notion: if the connector is in state q and the boundary requests present at the moment, encoded by α that is the conjunction of all possible requests presence, are such that $\alpha \leq g$, then the nodes f fire and the connector evolves to state q'. Each transition labeled by $g|f$ satisfies two criteria: (i) *reactivity* — data flow only through those nodes where a request is pending, capturing Reo's interaction model; and (ii) *uniformity* — which captures two properties: (a) the request set corresponding precisely to the firing set is sufficient to cause firing, and (b) removing additional unfired requests from a transition will not affect the (firing) behavior of the connector [3]. In compliance with these criteria, for a firing f, its guard g considers the presence of the least sufficient requests.

In Fig. 3 we depict the Reo Automata for the basic channel types listed in Fig. 1. Note that here and in the remainder of this paper, given transition $q \xrightarrow{g|f} q'$, if there is more than one transition from a state q to the same state q' we often just draw one arrow and separate their labels by commas, and every guard in a transition label in the automata is a conjunction of literals in Σ. Moreover, it is always possible to transform any guard g into this form, by taking its disjunctive normal form (DNF) $g_1 \vee \cdots \vee g_k$ and splitting the transition $g|f$ into the several $g_i|f$, for $i = 1, \ldots, k$. Given a transition relation δ we call $norm(\delta)$ the normalized transition relation obtained from δ by putting all of its guards in DNF and splitting the transitions as explained above.

2.1.1. Composing Reo connectors

We now model at the automata level the composition of Reo connectors. We define two operations: product, which puts two connectors in parallel, and synchronization, which models the plugging of two nodes. Thus, the product and synchronization operations can be used to obtain the automaton of a Reo connector by composing the automata of its primitive connectors. Later in this section we formally show the compositionality of these operations.

We first define the product operation for Reo Automata. This definition differs from the classical definition of (synchronous) product for automata: our automata have disjoint alphabets and they can either take steps together or independently. In the latter case the composite transition in the product automaton explicitly encodes that one of the two automata cannot perform a step in the current state, using the following notion:

Definition 2.2 (*[3]*)**.** Given a Reo Automaton $\mathcal{A} = (\Sigma, Q, \delta)$ and $q \in Q$ we define

$$q^\sharp = \neg \bigvee \{ g \mid q \xrightarrow{g|f} q' \in \delta \}.$$

This captures precisely the condition under which $\mathcal{A}$ cannot fire in state q.

Definition 2.3 (*Product of Reo Automata [3]*)**.** Given two Reo Automata $\mathcal{A}_1 = (\Sigma_1, Q_1, \delta_1)$ and $\mathcal{A}_2 = (\Sigma_2, Q_2, \delta_2)$ such that $\Sigma_1 \cap \Sigma_2 = \emptyset$, we define the *product* of $\mathcal{A}_1$ and $\mathcal{A}_2$ as $\mathcal{A}_1 \times \mathcal{A}_2 = (\Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, \delta)$ where δ consists of:

$$\begin{aligned} &\{(q,p) \xrightarrow{gg'|ff'} (q',p') \mid q \xrightarrow{g|f} q' \in \delta_1 \wedge p \xrightarrow{g'|f'} p' \in \delta_2\} \\ \cup \quad &\{(q,p) \xrightarrow{gp^\sharp|f} (q',p) \mid q \xrightarrow{g|f} q' \in \delta_1 \wedge p \in Q_2\} \\ \cup \quad &\{(q,p) \xrightarrow{gq^\sharp|f} (q,p') \mid p \xrightarrow{g|f} p' \in \delta_2 \wedge q \in Q_1\} \end{aligned}$$

Here and throughout, we use ff' as a shorthand for $f \cup f'$. The first term in the union, above, applies when both automata fire in parallel. The other terms apply when one automaton fires and the other is unable to (indicated by $p^\sharp$ and $q^\sharp$, respectively). Note that the product operation is closed for Reo Automata, since it preserves reactivity and uniformity [3]. Fig. 4 shows an example of the product of two automata.

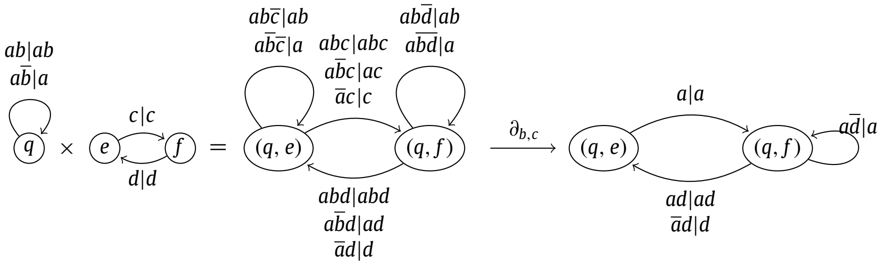


Fig. 4. Product of LossySync and FIFO1 and the synchronization of nodes b and c.

We now define a synchronization operation that corresponds to joining two nodes in a Reo connector. When synchronizing two nodes a and b (which are then made internal), in the resulting automaton, only the transitions where either both a and b or neither a nor b fire are kept, i.e., $a \in f \Leftrightarrow b \in f$ — this is what it means for a and b to synchronize. Moreover, we will only keep transitions whose guards encode that ports a and b are not blocked. That is, transitions labeled by $g|f$ where $g \not\leq \overline{a}\overline{b}$. This condition roughly corresponds to the notion of an internal node acting like a *self-contained pumping station* [1], which implies that an internal node cannot store data nor actively block behavior.

Definition 2.4 (*Synchronization [3]*)**.** Given a Reo Automaton $\mathcal{A} = (\Sigma, Q, \delta)$, we define the *synchronization* for $a, b \in \Sigma$ as $\partial_{a,b}\mathcal{A} = (\Sigma, Q, \delta')$ where

$$\delta' = \{q \xrightarrow{g\backslash_{ab}|f\backslash\{a,b\}} q' \mid q \xrightarrow{g|f} q' \in norm(\delta)\ s.t.\ g \not\leq \overline{a}\overline{b} \text{ and } a \in f \Leftrightarrow b \in f\}$$

Here and throughout, $g\backslash_{ab}$ is the guard obtained from g by deleting all occurrences of a and b. It is worth noting that synchronization preserves reactivity and uniformity.

Fig. 4[1] depicts the product of LossySync and FIFO1, together with the result of synchronizing nodes b and c. This synchronized result provides the semantics for the LossyFIFO1 example in Fig. 2.

2.1.2. Compositionality

Given two Reo Automata $\mathcal{A}_1$ and $\mathcal{A}_2$ over the disjoint alphabets Σ_1 and Σ_2, $\{a_1, \ldots, a_k\} \subseteq \Sigma_1$ and $\{b_1, \ldots, b_k\} \subseteq \Sigma_2$ we construct $\partial_{a_1,b_1}\partial_{a_2,b_2}\cdots\partial_{a_k,b_k}(\mathcal{A}_1 \times \mathcal{A}_2)$ as the automaton corresponding to a connector where node a_i of the first connector is connected to node b_i of the second connector, for all $i \in \{1, \ldots, k\}$. Note that the 'plugging' order does not matter because ∂ can be applied in any order and it interacts well with product. These properties are captured in the following lemma.

Lemma 2.5 (*[3]*)**.** *For the Reo Automata $\mathcal{A}_1 = (\Sigma_1, Q_1, \delta_1)$ and $\mathcal{A}_2 = (\Sigma_2, Q_2, \delta_2)$:*

1. $\partial_{a,b}\partial_{c,d}\mathcal{A}_1 = \partial_{c,d}\partial_{a,b}\mathcal{A}_1$, *if* $a, b, c, d \in \Sigma_1$.
2. $\left(\partial_{a,b}\mathcal{A}_1\right) \times \mathcal{A}_2 \sim \partial_{a,b}(\mathcal{A}_1 \times \mathcal{A}_2)$, *if* $a, b \notin \Sigma_2$

The notion of equivalence $\sim$ used above is bisimilarity, defined as follows.

Definition 2.6 (*Bisimulation [3]*)**.** Given the Reo Automata $\mathcal{A}_1 = (\Sigma, Q_1, \delta_1)$ and $\mathcal{A}_2 = (\Sigma, Q_2, \delta_2)$, we call $R \subseteq Q_1 \times Q_2$ a *bisimulation* iff for all $(q_1, q_2) \in R$:

If $q_1 \xrightarrow{g|f} q_1' \in \delta_1$ and $\alpha \in \mathbf{At}_\Sigma$, $\alpha \leq g$, then there exists a transition $q_2 \xrightarrow{g'|f} q_2' \in \delta_2$ such that $\alpha \leq g'$ and $(q_1', q_2') \in R$ and vice-versa.

We say that two states $q_1 \in Q_1$ and $q_2 \in Q_2$ are bisimilar if there exists a bisimulation relation containing the pair (q_1, q_2) and we write $q_1 \sim q_2$. Two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are bisimilar, written $\mathcal{A}_1 \sim \mathcal{A}_2$, if there exists a bisimulation relation such that every state of one automaton is related to some state of the other automaton.

3. Stochastic Reo

Stochastic Reo [4,5] is an extension of Reo where channel ends and channels are annotated with stochastic values for *data arrival rates* at channel ends and *processing delay rates* at channels. Such rates are non-negative real values and describe how the probability that an event occurs varies with time. Fig. 5 shows the stochastic versions of the basic Reo channels in Fig. 1. Here and throughout, for simplicity, we omit the node names, since they can be inferred from the names of their respective arrival rates: for instance, γa is the arrival rate of node a.

[1] For simplicity, we abstract away data-constrains on firings by assuming them true. Thus, the composition result of a LossySync and a FIFO1 channels, i.e., an overflow LossyFIFO1 connector, becomes indistinguishable from the automaton for a shift LossyFIFO1 [2] connector. However, by reviving data constraints we can distinguish the automata for these two connectors.

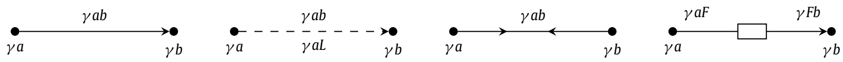


Fig. 5. Basic Stochastic Reo channels for the basic Reo channels of Fig. 1.

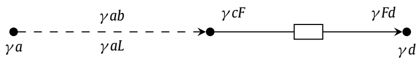


Fig. 6. Stochastic LossyFIFO1 connector.

It should be noted that such an annotation does not affect the functionalities of Reo connectors, thus, when the annotations of rates are neglected, the mapping between the operational semantics of Reo and Stochastic Reo is quite straightforward, i.e., one-to-one mapping.[2]

A processing delay rate represents how long it takes for a channel to perform a certain activity, such as data-flow. For instance, a LossySync has two associated rates γab and γaL for, respectively, successful data-flow from node a to node b, and losing the data item from node a. In a FIFO1 γaF represents the delay for data-flow from its source a into the buffer, and γFb for sending the data from the buffer to the sink b.

Arrival rates describe the time between consecutive arrivals of I/O requests at the source and sink nodes of Reo connectors. For instance, γa and γb in Fig. 5 are the associated arrival rates of write/take requests at the nodes a and b.

Since arrival rates on nodes model their interaction with the environment only, mixed nodes have no associated arrival rates. This is justified by the fact that a mixed node delivers data items instantaneously to the source end(s) of its connected channel(s). Hence, when joining a source with a sink node into a mixed node, their arrival rates are discarded.[3]

A stochastic version of the LossyFIFO1 is depicted in Fig. 6, including its arrival and processing delay rates.

As a more complex Stochastic Reo connector, Fig. 7 shows a *discriminator* which takes the first arriving input value and produces it as its output; it also ensures that an input value arrives on every other input port before the next round.

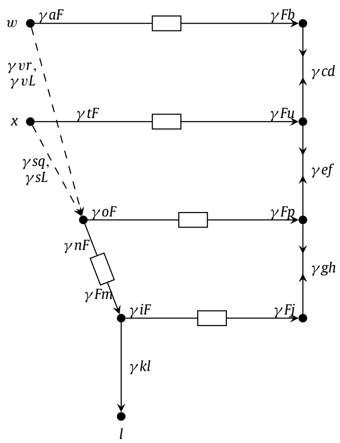


Fig. 7. Stochastic Discriminator with two inputs.

3.1. Semantics: Stochastic Reo Automata

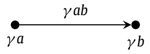
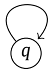
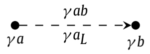
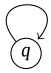
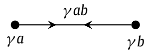
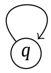
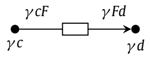
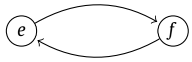
In this section, we provide a compositional semantics for Stochastic Reo connectors, as an extension of Reo Automata with functions that assign stochastic values for data-flows and I/O request arrivals.

Definition 3.1 (*Stochastic Reo Automaton*)**.** A Stochastic Reo Automaton is a triple $(\mathcal{A}, \mathbf{r}, \mathbf{t})$ where $\mathcal{A} = (\Sigma, Q, \delta_{\mathcal{A}})$ is a Reo Automaton and

2 Stochastic Reo is a conservative extension of Reo: if you take a certain Stochastic Reo connector and delete all the rates, what one ends up with is precisely the Reo connector where the rates have also been deleted.

3 For simplicity, we assume ideal nodes whose activity incurs no delay. Any real implementation of a node, of course, induces some processing delay rate. A real node can be modeled as a composition of an ideal node with a Sync channel that manifests the processing delay rate. Thus, we can associate delay distributions with Stochastic Reo nodes and automatically translate them into such "Sync plus ideal node" constructs.

Table 1
Stochastic Reo Automata for the basic Stochastic Reo channels of Fig. 5.

Synchronous Channels			
	$ab\|ab,\ \{(\{a\},\{b\},\gamma ab)\}$ 	a b	**r** γa γb
	$ab\|ab,\ \{(\{a\},\{b\},\gamma ab)\}$ $a\bar{b}\|a,\ \{(\{a\},\emptyset,\gamma a_L)\}$ 	a b	**r** γa γb
	$ab\|ab,\ \{(\{a,b\},\emptyset,\gamma ab)\}$ 	a b	**r** γa γb
Asynchronous Channel			
	$c\|c,\ \{(\{c\},\emptyset,\gamma cF)\}$  $d\|d,\ \{(\emptyset,\{d\},\gamma Fd)\}$	c d	**r** γc γd

- $\mathbf{r} : \Sigma \rightarrow \mathbb{R}^+$ is a function that associates with each node its arrival rate.
- $\mathbf{t} : \delta_{\mathcal{A}} \rightarrow 2^{\Theta}$ is a function that associates with a transition a subset of $\Theta = 2^{\Sigma} \times 2^{\Sigma} \times \mathbb{R}^+$ such that for any $I,\ O \subseteq \Sigma$ and $I \cap O = \emptyset$, each $(I, O, r) \in \Theta$ corresponds to a data-flow where I is a set of input and/or mixed nodes; O is a set of output and/or mixed nodes; and r is a processing delay rate for the data-flow described by I and O, which must satisfy that given two 3-tuples $(I_1, O_1, r_1),\ (I_2, O_2, r_2) \in \Theta$, if $I_1 = I_2 \wedge O_1 = O_2$, then $r_1 = r_2$.

The Stochastic Reo Automata corresponding to the basic Stochastic Reo channels in Fig. 5 are defined by the functions **r** and **t** shown in Table 1. Note that the function **t** is depicted in the transitions, and function **r** is shown inside the tables.

An element of $\theta \in \Theta$ is accessed by projection functions $i : \Theta \rightarrow 2^{\Sigma}$, $o : \Theta \rightarrow 2^{\Sigma}$ and $\upsilon : \Theta \rightarrow \mathbb{R}^+$; $i(\theta)$ and $o(\theta)$ return the respective input and output nodes of a data-flow, and $\upsilon(\theta)$ returns the delay rate of the data-flow through the nodes in $i(\theta)$ and $o(\theta)$.

As mentioned in Section 2.1.2, Reo Automata provide a compositional semantics for Reo connectors. As an extension of Reo Automata, Stochastic Reo Automata also present the composition of Stochastic Reo connectors at the automata level. For this purpose, we define two operations of the product and the synchronization that are used to obtain an automaton of a Stochastic Reo connector by composing the automata of its primitive connectors. The compositionality of these operations is formally proved later in this section.

Definition 3.2 (*Product*). Given two Stochastic Reo Automata $(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1)$ with $\mathcal{A}_1 = (\Sigma_1, Q_1, \delta_1)$ and $(\mathcal{A}_2, \mathbf{r}_2, \mathbf{t}_2)$ with $\mathcal{A}_2 = (\Sigma_2, Q_2, \Sigma_2)$, their product is defined as $(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1) \times (\mathcal{A}_2, \mathbf{r}_2, \mathbf{t}_2) = (\mathcal{A}_1 \times \mathcal{A}_2, \mathbf{r}_1 \cup \mathbf{r}_2, \mathbf{t})$ where

$$\mathbf{t}((q,p) \xrightarrow{gg'|ff'} (q',p')) = \mathbf{t}_1(q \xrightarrow{g|f} q') \cup \mathbf{t}_2(p \xrightarrow{g'|f'} p') \quad \text{where } q \xrightarrow{g|f} q' \in \delta_1 \wedge p \xrightarrow{g'|f'} p' \in \delta_2$$
$$\mathbf{t}((q,p) \xrightarrow{gp^{\sharp}|f} (q',p)) = \mathbf{t}_1(q \xrightarrow{g|f} q') \quad \text{where } q \xrightarrow{g|f} q' \in \delta_1 \wedge p \in Q_2$$
$$\mathbf{t}((q,p) \xrightarrow{gq^{\sharp}|f} (q,p')) = \mathbf{t}_2(p \xrightarrow{g|f} p') \quad \text{where } p \xrightarrow{g|f} p' \in \delta_2 \wedge q \in Q_1$$

Note that we use $\times$ to denote both the product of Reo Automata and the product of Stochastic Reo Automata. Since Stochastic Reo Automata are a conservative extension of Reo Automata with stochastic information, it is easy to derive the product result for a Reo Automaton of a certain connector from the product result of a Stochastic Reo Automaton for the same connector, just by ignoring the second and third components of the Stochastic Reo Automaton.

The set of 3-tuples that **t** associates with a transition m combines the delay rates involved in all data-flows synchronized by the transition m. In order to keep Stochastic Reo Automata generally useful and compositional, and their product commutative, we avoid fixing the precise formal meaning of distribution rates of synchronized transitions composed in a product; instead, we represent the "delay rate" of their composite transition in the product automaton as the union of the delay rates of the synchronizing transitions of the two automata. How exactly these rates combine to yield the composite rate of the transition depends on the different properties of the distributions and their time ranges. For example, in the continuous-time case, no two events can occur at the same time; and the exponential distributions are not closed under taking maximum. In Section 4, we show how to translate a Stochastic Reo Automaton to a CTMC using the union of the rates of the exponential distributions in the continuous-time case.

Definition 3.3 (*Synchronization*)**.** For a Stochastic Reo Automaton $(\mathcal{A}, \mathbf{r}, \mathbf{t})$, the synchronization operation on nodes a and b is defined as $\partial_{a,b}(\mathcal{A}, \mathbf{r}, \mathbf{t}) = (\partial_{a,b}\mathcal{A}, \mathbf{r}', \mathbf{t}')$ where

- $\mathbf{r}'$ is $\mathbf{r}$ restricted to the domain $\Sigma \setminus \{a, b\}$.
- $\mathbf{t}'$ is defined as:

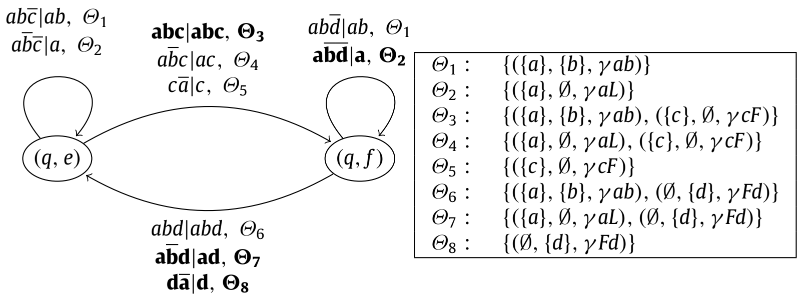
$$\mathbf{t}'(q \xrightarrow{g\backslash_{ab}|f\backslash\{a,b\}} q') = \{(A', B', r) \mid (A, B, r) \in \mathbf{t}(q \xrightarrow{g|f} q'),\\ A' = sync(A, \{a, b\}) \wedge B' = sync(B, \{a, b\})\}$$

where $sync : 2^{\Sigma} \times 2^{\Sigma} \to 2^{\Sigma}$ gathers nodes joined by synchronization, and is defined as:

$$sync(A, B) = \begin{cases} A \cup B & \text{if } A \cap B \neq \emptyset \\ A & \text{otherwise} \end{cases}$$

Note that we use the symbol $\partial_{a,b}$ to denote both the synchronization of Reo Automata and the synchronization of Stochastic Reo Automata. The number of nodes joined by the synchronization is always two, and the sets of joined nodes in multiple synchronization steps are disjoint. That is, given two different synchronization $\partial_{a,b}$ and $\partial_{c,d}$ on a Stochastic Reo automaton, $\{a, b\} \cap \{c, d\} = \emptyset$.

We now revisit the LossyFIFO1 example. Its semantics is given by the triple $(\mathcal{A}_{LossyFIFO1}, \mathbf{r}, \mathbf{t})$, where $\mathcal{A}_{LossyFIFO1}$ is the automaton depicted in Fig. 4 and $\mathbf{r}$ is defined as $\mathbf{r} = \{a \mapsto \gamma a, d \mapsto \gamma d\}$. For $\mathbf{t}$, we first compute $\mathbf{t}_{LossySync \times FIFO1}$:



Above, the labels that correspond to the transitions that will be kept after synchronization appear in **bold**. Thus, the result of joining nodes by synchronization, is shown in Fig. 8 as:

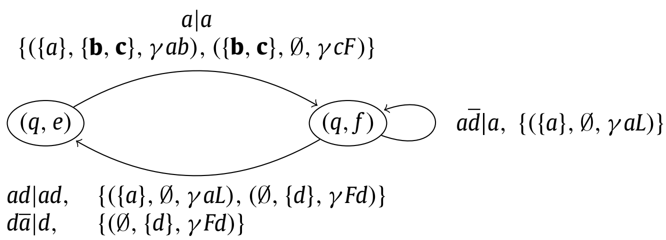


Fig. 8. Stochastic Reo Automaton for LossyFIFO1.

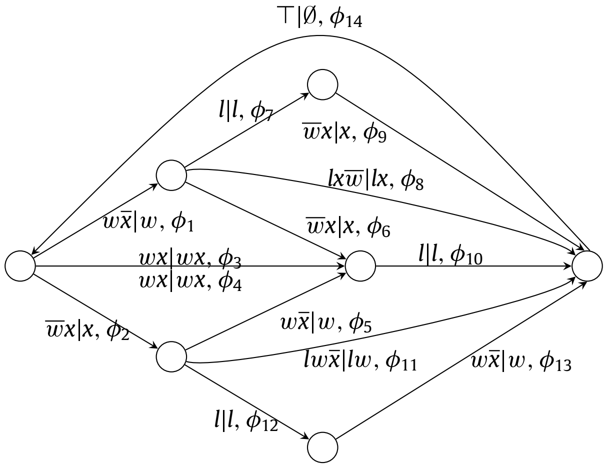
Note that the port names that appear in **bold** represent the synchronization of nodes b and c.

In this way, we can carry in the semantic model of Reo connectors, given as Reo automata, stochastic information, i.e., arrival rates and processing delay rates that pertain to its QoS.

As a more complex example of such composition, Fig. 9 shows a Stochastic Reo Automaton for the discriminator in Fig. 7. Definition 3.1 shows that our extension of Reo Automata deals with such stochastic information separately, apart from the underlying Reo Automaton. Thus, our extended model retains the properties of Reo Automata, i.e., the compositionality result presented in Section 2.1.2 can be extended to Stochastic Reo Automata.

Given two Stochastic Reo Automata $(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1)$ and $(\mathcal{A}_2, \mathbf{r}_2, \mathbf{t}_2)$ with $\mathcal{A}_1 = (\Sigma_1, Q_1, \delta_1)$ and $\mathcal{A}_2 = (\Sigma_2, Q_2, \delta_2)$ over the disjoint alphabets Σ_1 and Σ_2, $\{a_1, \ldots, a_k\} \subseteq \Sigma_1$ and $\{b_1, \ldots, b_k\} \subseteq \Sigma_2$, we construct $\partial_{a_1,b_1}\partial_{a_2,b_2}\cdots\partial_{a_k,b_k}(\mathcal{A}_1 \times \mathcal{A}_2)$ as the automaton corresponding to a connector where node a_i of the first connector is connected to node b_i of the second connector, for all $i \in \{1, \ldots, k\}$. Note that the 'plugging' order does not matter because ∂ can be applied in any order and it interacts well with product. These properties are captured in the following lemma.

Lemma 3.4 (*Compositionality*)**.** *Given two disjoint Stochastic Reo Automata* $(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1)$ *and* $(\mathcal{A}_2, \mathbf{r}_2, \mathbf{t}_2)$ *with* $\mathcal{A}_1 = (\Sigma_1, Q_1, \delta_1)$ *and* $\mathcal{A}_2 = (\Sigma_2, Q_2, \delta_2)$*,*



$$
\begin{array}{lll}
\phi_1 & = \{ & (\{w\}, \{a, v\}, \gamma avw),\ (\{a, v\}, \{r\}, \gamma vr),\ (\{a, v\}, \emptyset, \gamma aF), \\
 & & (\{r\}, \{n, o\}, \gamma nor), (\{n, o\}, \emptyset, \gamma oF),\ (\{n, o\}, \emptyset, \gamma nF)\ \} \\
\phi_2 & = \{ & (\{x\}, \{s, t\}, \gamma stx),\ (\{s, t\}, \{q\}, \gamma sq),\ (\{s, t\}, \emptyset, \gamma tF), \\
 & & (\{q\}, \{n, o\}, \gamma noq),\ (\{n, o\}, \emptyset, \gamma oF),\ (\{n, o\}, \emptyset, \gamma nF)\ \} \\
\phi_3 & = & \phi_2 \cup \phi_5 \\
\phi_4 & = & \phi_1 \cup \phi_6 \\
\phi_5 & = \{ & (\{w\}, \{a, v\}, \gamma avw),\ (\{a, v\}, \emptyset, \gamma aF),\ (\{a, v\}, \emptyset, \gamma vL)\ \} \\
\phi_6 & = \{ & (\{x\}, \{s, t\}, \gamma stx)\ , (\{s, t\}, \emptyset, \gamma tF),\ (\{s, t\}, \emptyset, \gamma sL)\ \} \\
\phi_7 & = \{ & (\emptyset, \{m\}, \gamma Fm),\ (\{m\}, \{i, k\}, \gamma ikm),\ (\{i, k\}, \{l\}, \gamma kl), \\
 & & (\{i, k\}, \emptyset, \gamma iF)\ \} \\
\phi_8 & = & \phi_7 \cup \phi_6 \\
\phi_9 & = & \phi_6 \\
\phi_{10} & = & \phi_{12}\ =\ \phi_7 \\
\phi_{11} & = & \phi_7 \cup \phi_5 \\
\phi_{13} & = & \phi_5 \\
\phi_{14} & = \{ & (\emptyset, \{b, c\}, \gamma Fb),\ (\{b, c, d, e\}, \emptyset, \gamma cd),\ (\emptyset, \{u\}, \gamma Fu) \\
 & & (\{u\}, \{d, e\}, \gamma deu),\ (\{d, e, f, g\}, \emptyset, \gamma ef),\ (\emptyset, \{p\}, \gamma Fp), \\
 & & (\{p\}, \{f, g\}, \gamma fgp),\ (\{f, g, h, j\}, \emptyset, \gamma gh), (\emptyset, \{h, j\}, \gamma Fj)\ \}
\end{array}
$$

Fig. 9. Stochastic Reo Automaton for discriminator in Fig. 7.

1. $\partial_{a,b}\partial_{c,d}(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1) = \partial_{c,d}\partial_{a,b}(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1)$, *if* $a, b, c, d \in \Sigma_1$
2. $(\partial_{a,b}(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1)) \times (\mathcal{A}_2, \mathbf{r}_2, \mathbf{t}_2) \sim \partial_{a,b}((\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1) \times (\mathcal{A}_2, \mathbf{r}_2, \mathbf{t}_2))$, *if* $a, b \notin \Sigma_2$

Here $(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1) \sim (\mathcal{A}_2, \mathbf{r}_2, \mathbf{t}_2)$ if and only if $\mathcal{A}_1 \sim \mathcal{A}_2$, $\mathbf{r}_1 = \mathbf{r}_2$ and $\mathbf{t}_1 = \mathbf{t}_2$.

Proof. Let

- $\partial_{a,b}\partial_{c,d}(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1)\ =\ (\partial_{a,b}\partial_{c,d}\mathcal{A}_1, \mathbf{r}_1', \mathbf{t}_1')$ and
- $\partial_{c,d}\partial_{a,b}(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1)\ =\ (\partial_{c,d}\partial_{a,b}\mathcal{A}, \mathbf{r}_1'', \mathbf{t}_1'')$

By Lemma 4.13 in [3] which is the analogue result for Reo Automata, we know that $\partial_{a,b}\partial_{c,d}\mathcal{A}_1\ =\ \partial_{c,d}\partial_{a,b}\mathcal{A}_1$. Using basic set theory, we also have that

$$
\begin{aligned}
\mathbf{r}_1' &= \mathbf{r} \mid (\Sigma \setminus \{a, b\}) \setminus \{c, d\} \\
&= \mathbf{r} \mid (\Sigma \setminus \{c, d\}) \setminus \{a, b\} \\
&= \mathbf{r}_1''
\end{aligned}
$$

where for $v \subseteq \Sigma$, $\mathbf{r}|v$ is the restriction of $\mathbf{r}$ to v.

Before moving to the fact that $\mathbf{t}_1'\ =\ \mathbf{t}_1''$, we show that the order of applying the synchronization is irrelevant to the synchronization result, i.e., given three node sets A, $\{a, b\}$, and $\{c, d\}$,

$$
sync(sync(A, \{a, b\}), \{c, d\}) = sync(sync(A, \{c, d\}), \{a, b\})
$$

because, given three node sets A, B, and C with $B \cap C = \emptyset$,

$$sync(sync(A,B),C) = \begin{cases} A \cup B \cup C & \text{if } A \cap B \neq \emptyset \ \wedge \ A \cap C \neq \emptyset \\ A \cup B & \text{if } A \cap B \neq \emptyset \ \wedge \ A \cap C = \emptyset \\ A \cup C & \text{if } A \cap B = \emptyset \ \wedge \ A \cap C \neq \emptyset \\ A & \text{otherwise} \end{cases}$$

and the set union operation $\cup$ is commutative.

$$\begin{aligned} & \mathbf{t}_1'(q \xrightarrow{g\backslash abcd \ | \ (f\backslash\{a,b\})\backslash\{c,d\}} q') \\ &= \{(A',B',r) \mid (A,B,r) \in \mathbf{t}_1(q \xrightarrow{g|f} q'), \\ &\qquad A^1 = sync(A,\{a,b\}) \ \wedge \ B^1 = sync(B,\{a,b\}) \ \wedge \\ &\qquad A' = sync(A^1,\{c,d\}) \ \wedge \ B' = sync(B^1,\{c,d\})\} \\ &= \{(A',B',r) \mid (A,B,r) \in \mathbf{t}_1(q \xrightarrow{g|f} q'), \\ &\qquad A^1 = sync(A,\{c,d\}) \ \wedge \ B^1 = sync(B,\{c,d\}) \ \wedge \\ &\qquad A' = sync(A^1,\{a,b\}) \ \wedge \ B' = sync(B^1,\{a,b\})\} \\ &= \mathbf{t}''(q \xrightarrow{g\backslash cdab \ | \ (f\backslash\{c,d\})\backslash\{a,b\}} q') \end{aligned}$$

For the second proposition, let

- $(\partial_{a,b}(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1)) \times (\mathcal{A}_2, \mathbf{r}_2, \mathbf{t}_2) = (\partial_{a,b}(\mathcal{A}_1) \times \mathcal{A}_2, \mathbf{r}, \mathbf{t})$ and
- $\partial_{a,b}((\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1) \times (\mathcal{A}_2, \mathbf{r}_2, \mathbf{t}_2)) = (\partial_{a,b}(\mathcal{A}_1 \times \mathcal{A}_2), \mathbf{r}', \mathbf{t}')$

By [3, Lemma 4.13], we know that $\partial_{a,b}(\mathcal{A}_1) \times \mathcal{A}_2 = \partial_{a,b}(\mathcal{A}_1 \times \mathcal{A}_2)$ if $a, b \notin \Sigma_2$. It remains to prove that $\mathbf{r} = \mathbf{r}'$ and $\mathbf{t} = \mathbf{t}'$.

To prove $\mathbf{r} = \mathbf{r}'$, we easily calculate, $\forall p \in (\Sigma_1 \setminus \{a,b\}) \cup \Sigma_2$:

$$\begin{aligned} \mathbf{r}(p) &= \begin{cases} \mathbf{r}_1(p) & \text{if } p \in \Sigma_1 \setminus \{a,b\} \\ \mathbf{r}_2(p) & \text{if } p \in \Sigma_2 \end{cases} \\ &= \mathbf{r}'(p) \end{aligned}$$

To prove $\mathbf{t} = \mathbf{t}'$, consider transitions $(q_1, q_2) \xrightarrow{(g_1\backslash ab)g_2|(f_1\backslash\{a,b\})f_2} (p_1, p_2)$ in $\partial_{a,b}(\mathcal{A}_1) \times \mathcal{A}_2$ and $(q_1, q_2) \xrightarrow{(g_1g_2)\backslash ab|(f_1f_2)\backslash\{a,b\}} (p_1, p_2)$ in $\partial_{a,b}(\mathcal{A}_1 \times \mathcal{A}_2)$ with $g_i \in \mathcal{B}_{\Sigma_i}$ and $f_i \in 2^{\Sigma_i}$ for $i = 1, 2$, which includes joined nodes a and b. Then,

$$\begin{aligned} & \mathbf{t}((q_1,q_2) \xrightarrow{(g_1\backslash ab)g_2|(f_1\backslash\{a,b\})f_2} (p_1,p_2)) \\ &= \{(A',B',r) \ \mid \ (A,B,r) \in \mathbf{t}_1(q_1 \xrightarrow{g_1|f_1} p_1), \\ &\qquad A' = sync(A,\{a,b\}) \ \wedge \ B' = sync(B,\{a,b\})\} \\ &\cup \ \{(A,B,r) \ \mid \ (A,B,r) \in \mathbf{t}_2(q_2 \xrightarrow{g_2|f_2} p_2)\} \\ &= \{(A',B',r) \ \mid \ (A,B,r) \in \mathbf{t}_1(q_1 \xrightarrow{g_1|f_1} p_1) \cup \mathbf{t}_2(q_2 \xrightarrow{g_2|f_2} p_2), \\ &\qquad A' = sync(A,\{a,b\}) \ \wedge \ B' = sync(B,\{a,b\})\} \\ &= \mathbf{t}'((q_1,q_2) \xrightarrow{(g_1g_2)\backslash ab|(f_1f_2)\backslash\{a,b\}} (p_1,p_2)) \end{aligned}$$

Since $sync(C, D) = C$ if $C \cap D = \emptyset$, the equation above holds without a need to consider if $ab \leq g_1$ or $\{a, b\} \subseteq f_1$. This also implies that $\mathbf{t} = \mathbf{t}'$ holds for transitions $(q, p) \xrightarrow{g|f} (q', p)$ and $(q, p) \xrightarrow{g'|f'} (q, p')$, which do not include the joined nodes, in $\partial_{a,b}(\mathcal{A}_1) \times \mathcal{A}_2$ (equivalently, in $\partial_{a,b}(\mathcal{A}_1 \times \mathcal{A}_2)$). $\square$

4. Translation to CTMC

In this section, we show how to translate a Stochastic Reo Automaton into a homogeneous CTMC model. A homogeneous CTMC is a stochastic process with (1) discrete state space, (2) Markov property, (3) memoryless property, and (4) homogeneity in the continuous-time domain [6]. That is, in a homogeneous CTMC, (1) — the state space is countable; (2) — the state changes depend on the current state, not on the trace of state changes; (3) — the remaining time before exiting a current state is independent of the time already spent in that state; and (4) — the probability of state changes does not depend on a time instance, i.e., the occurrence of stochastic event can take place at any time instance. These properties yield efficient methodologies [7] for numerical analysis. In the continuous-time domain, the exponential distribution is the only one that satisfies the memoryless property. Therefore, for the translation, we assume that the rates of data-arrivals and data-flows are exponentially distributed.

A CTMC model derived from a Stochastic Reo Automaton $(\mathcal{A}, \mathbf{r}, \mathbf{t})$ with $\mathcal{A} = (\Sigma, Q, \delta_{\mathcal{A}})$ is a pair (S, δ) where $S = S_A \cup S_M$ is the set of states. S_A represents the configurations of the system derived from its Stochastic Reo Automaton and the pending status of I/O requests; S_M is the set of states that result from the micro-step division of synchronous actions (see below). $\delta = \delta_{Arr} \cup \delta_{Proc} \subseteq S \times \mathbb{R}^+ \times S$, explained below, is the set of transitions, each labeled with a stochastic value specifying the arrival or the processing delay rate of the transition. δ_{Arr} and δ_{Proc} are defined in Section 4.3.

A state in S models a configuration of the connector, including the presence of the I/O requests pending on its boundary nodes, if any. Data-arrivals change system configuration only by changing the pending status of their respective boundary nodes. Data-flows corresponding to a transition of a Reo Automaton change the system configuration, and release the pending I/O requests on their involved boundary nodes.

In a CTMC model, the probability that two events (e.g., the arrival of an I/O request, the transfer of a data item, a processing step, etc.) happen at the same time is *zero*: only a single event occurs at a time. In compliance with this requirement, for a Stochastic Reo Automaton $(\mathcal{A}, \mathbf{r}, \mathbf{t})$ with $\mathcal{A} = (\Sigma, Q, \delta_{\mathcal{A}})$ and a set of boundary nodes $\Sigma' \subseteq \Sigma$, we define its set of data-arrival transitions, δ_{Arr}, in several steps. The set S_A and the preliminary set[4] of data-arrival transitions of the CTMC derived for $(\mathcal{A}, \mathbf{r}, \mathbf{t})$ are defined as:

$$\begin{aligned} S_A &= \{(q, R) \mid q \in Q,\ R \subseteq \Sigma'\} \\ \delta'_{Arr} &= \{(q, R) \xrightarrow{\mathbf{r}(c)} (q, R \cup \{c\}) \mid (q, R),\ (q, R \cup \{c\}) \in S_A,\ c \notin R\} \end{aligned}$$

The set δ'_{Arr} is used in Section 4.3 to define δ_{Arr}.

As an example of obtaining S_A and δ'_{Arr}, let us recall the Stochastic Reo Automaton for the LossyFIFO1 connector in Fig. 8. It has two states of (q, e) and (q, f), and its boundary nodes set Σ' is $\{a, d\}$. Therefore,

$$\begin{aligned} S_A &= \{\ ((q, e), \emptyset),\ ((q, e), \{a\}),\ ((q, e), \{d\}),\ ((q, e), \{a, d\}) \\ &\qquad ((q, f), \emptyset),\ ((q, f), \{a\}),\ ((q, f), \{d\}), ((q, f), \{a, d\})\ \} \\ \delta'_{Arr} &= \{\ ((q, e), \emptyset) \xrightarrow{\mathbf{r}(a)} ((q, e), \{a\}),\ ((q, e), \emptyset) \xrightarrow{\mathbf{r}(d)} ((q, e), \{d\}), \\ &\quad ((q, e), \{a\}) \xrightarrow{\mathbf{r}(d)} ((q, e), \{a, d\}),\ ((q, e), \{d\}) \xrightarrow{\mathbf{r}(a)} ((q, e), \{a, d\}), \\ &\quad ((q, f), \emptyset) \xrightarrow{\mathbf{r}(a)} ((q, f), \{a\}),\ ((q, f), \emptyset) \xrightarrow{\mathbf{r}(d)} ((q, f), \{d\}), \\ &\quad ((q, f), \{a\}) \xrightarrow{\mathbf{r}(d)} ((q, f), \{a, d\}), ((q, e), \{d\}) \xrightarrow{\mathbf{r}(a)} ((q, f), \{a, d\})\ \} \end{aligned}$$

In the remainder of this section, for simplicity, we abbreviate the configurations of states. For instance, in this example, (q, e) and (q, f) are represented as e and f. Thus, the diagram of S_A and δ'_{Arr} are represented as follows:

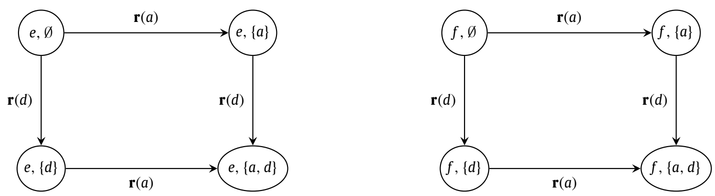


Fig. 10. State diagram for data-arrivals.

4.1. Micro-step transitions

The CTMC transitions associated with data-flows are more complicated because groups of synchronized data-flows are modeled as a single transition in a Reo Automaton, abstracting away their precise occurrence order. Therefore, we need to divide such synchronized data-flows into so-called micro-step transitions,[5] respecting the connection information, i.e., the topology of a Reo connector, through which the data-flow occurs.

The connection information can be recovered from the 3-tuples associated with each transition in a Reo Automaton, since the first and the second elements of a 3-tuple describe the input and the output nodes, respectively, involved in the data-flow of its transition, and the data-flow in the transition occurs from its input to its output nodes.

For example, the transition $(q, e) \xrightarrow{a|a} (q, f)$ in the Reo Automaton of the LossyFIFO1 example in Fig. 8 has $\{(\{a\}, \{b, c\}, \gamma ab), (\{b, c\}, \emptyset, \gamma cF)\}$ as a set of 3-tuples. The connection information inferred from this set states that data-flow occurs from a to the buffer through b and c. The transition is thus divided into two consecutive micro-step transitions $(\{a\}, \{b, c\}, \gamma ab)$ and $(\{b, c\}, \emptyset, \gamma cF)$.

Such data-flow information on each transition in a Stochastic Reo Automaton is formalized by a *delay-sequence* defined by the following grammar:

$$\Lambda \ni \lambda ::= \epsilon \ \mid\ \theta\ \mid\ \lambda|\lambda\ \mid\ \lambda; \lambda$$

where ϵ is the empty sequence and θ is a 3-tuple (I, O, r) for a basic Reo channel. $\lambda|\lambda$ denotes parallel composition, and $\lambda; \lambda$ denotes sequential composition. The empty sequence ϵ is an identity element for | and ;, | is commutative, associative, and idempotent, ; is associative and distributes over |. Most of properties of these compositional operators are intuitive, except for the distributivity of ;. The delay-sequence λ extracted by the Algorithm 4.2.1 is in the format $\lambda = \lambda_1|\lambda_1| \dots |\lambda_n$. Consider

4 In the process of generating CTMCs, some macro-step events (e.g., synchronized data-flows) are divided into several micro-step events. After that, independent events (e.g., data-arrivals) are considered as preemptive events between any two micro-step events. Before this division, we need to specify the transitions for respective synchronized data-flows. For this purpose, S_A is obtained to describe source and target states of these transitions. The preliminary set of data-arrivals includes the transitions that connect the states in S_A, each of which corresponds to all possible data-arrivals at every connector configurations.

5 This division delineates multiple synchronized data-flows, not each data-flow itself.

$\lambda = \lambda_1|\lambda_2 = (\theta_1; \theta_2)|(\theta_3; \theta_2)$.[6] Distributivity, that is, the property $(\theta_1; \theta_2)|(\theta_3; \theta_2) = (\theta_1|\theta_3); \theta_2$ is justified by the fact that θ_2 is the delay of the same action and the other actions θ_1 and θ_3 in the composed delays $(\theta_1; \theta_2)$ and $(\theta_3; \theta_2)$ need to finish before the action corresponding to θ_2 occurs. We use this distributivity law to generate compacter delay-sequences from the delay-sequences extracted in Section 4.2. For example, recall the delay-sequence $\lambda = (\theta_1|\theta_2)|(\theta_3; \theta_2)$. Then, λ becomes $(\theta_1|\theta_3); \theta_2$ and it still preserves the sequential precedence of θ_1 and θ_3 over θ_2 and shows the undetermined order between θ_1 and θ_3.

4.2. Extracting delay-sequences

The delay-sequence corresponding to a set of 3-tuples associated with a transition in a Stochastic Reo Automaton is obtained by Algorithm 4.2.1. Note that if the parameter of the function **Ext** is a singleton, then $\mathbf{Ext}(\{\theta\}) = \theta$ since $i(\theta) \cap o(\theta) = \emptyset$ based on Definition 3.1.

Algorithm 4.2.1 Extraction of a delay-sequence out of a set Θ of 3-tuples

Ext(Θ) where $\Theta = \mathbf{t}(p \xrightarrow{g|f} q)$
 $S = \epsilon,\ toGo = \Theta,\ Init := \{\theta \in \Theta \mid i(\theta) \cap o(\theta') = \emptyset \text{ for all } \theta' \in \Theta\}$
 for $\theta \in Init$ **do**
  $\lambda_\theta := \theta$
  $Pre := \{\theta\}$
  $toGo := toGo \setminus Pre$
  $Post = \{\theta \in toGo \mid \exists \theta' \in Pre\ s.t.\ o(\theta') \cap i(\theta) \neq \emptyset\}$
  while $Post \neq \emptyset$ **do**
   $\lambda' := (\theta_1| \cdots |\theta_k)$ where $Post = \{\theta_1, \ldots, \theta_k\}$
   $\lambda_\theta := \lambda_\theta; \lambda'$
   $Pre := Post$
   $toGo := toGo \setminus Pre$
   $Post := \{\theta \in toGo \mid \exists \theta' \in Pre\ s.t.\ o(\theta') \cap i(\theta) \neq \emptyset\}$
  end while
  $S := S|(\lambda_\theta)$
 end for
 return S

Intuitively, the **Ext** function delineates the set of activities that – at the level of a Stochastic Reo Automaton – must happen synchronously/atomically, into its corresponding delay-sequences. If a certain data-flow associated with a 3-tuple θ_1 explicitly precedes another one θ_2, then θ_1 is sequenced before θ_2, i.e., encoded as $\theta_1; \theta_2$. Otherwise, they can occur in any order, encoded as $\theta_1|\theta_2$.

Applying Algorithm 4.2.1 to the LossyFIFO1 example of Fig. 8 yields the following result:

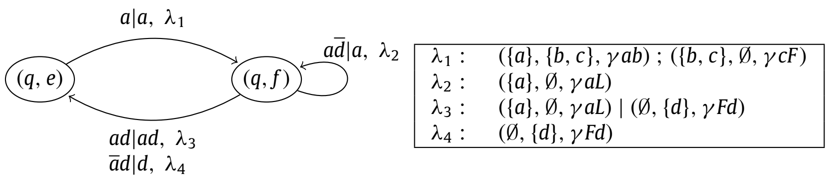


Fig. 11. Extracting delay-sequences of LossyFIFO1.

The parameter Θ of Algorithm 4.2.1 is a finite set of 3-tuples, and *Init*, *Post* and *toGo*, subsets of Θ, are also finite. Moreover, *Post* becomes eventually $\emptyset$ since *toGo* decreases during the procedure. Thus, we can conclude that Algorithm 4.2.1 always terminates.

A resulting delay-sequence S extracted by Algorithm 4.2.1 is generated by the parallel composition of λ_θ. The order of selecting θ from the set *Init* is not deterministic, thus, the resulting delay-sequence for the same input can be syntactically different, for example, $\lambda_\theta|\lambda_{\theta'}$ and $\lambda_{\theta'}|\lambda_\theta$ with $Init = \{\theta, \theta'\}$. However, the parallel composition operator | is commutative, thus, the composition order of | does not matter.

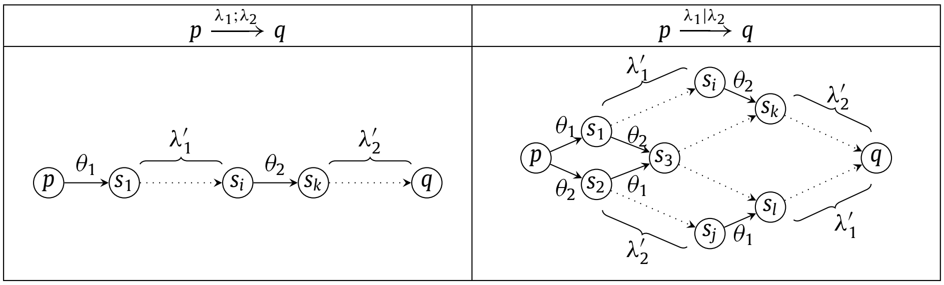
4.3. Deriving the CTMC

We now show how to derive the transitions in the CTMC model from the transitions in a Stochastic Reo Automaton. We do this in two steps:

[6] In general, the operators inside '()' have the highest order. Here and in the remainder of this paper, we also follow this standard order without explicit mention.

1. For each transition $p \xrightarrow{g|f} q \in \delta_{\mathcal{A}}$, we derive transitions $(p, R) \xrightarrow{\lambda} (q, R \setminus f)$ for every set of pending requests R that suffices to activate the guard g ($\widehat{R} \leq g \setminus \widehat{\widehat{\Sigma}}$), where λ is the delay-sequence associated with the set of 3-tuples $\mathbf{t}(p \xrightarrow{g|f} q)$. This set of derived transitions is defined below as δ_{Macro}.
2. We divide a transition in δ_{Macro} labeled by λ into a combination of micro-step transitions, each of which corresponds to a single event.

The following figure briefly illustrates the procedure mentioned above, for two transitions $p \xrightarrow{\lambda_1;\lambda_2} q$ and $p \xrightarrow{\lambda_1|\lambda_2} q$ where $\lambda_1 = \theta_1; \lambda'_1$ and $\lambda_2 = \theta_2; \lambda'_2$:



A sequential delay-sequence $\lambda_1; \lambda_2$ allows for the events corresponding to λ_1 to occur before the ones corresponding to λ_2. For a parallel delay-sequence $\lambda_1|\lambda_2$, events corresponding to λ_1 and λ_2 occur interleaving each other, while they preserve their respective order of occurrence in λ_1 and λ_2. All indexed states s_n are included in S_M which consists of the states derived from the division of the synchronized data-flows into micro-step transitions. The formal description of dealing with these two delay-sequences is presented in the definition of a *div* function below, in which handling the respective delay-sequences correspond to the second and the third conditions of the *div* function.

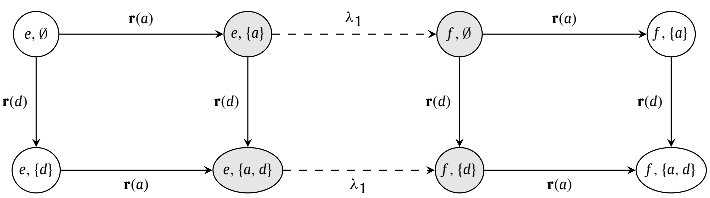
Given a Stochastic Reo Automaton $(\mathcal{A}, \mathbf{r}, \mathbf{t})$ with $\mathcal{A} = (\Sigma, Q, \delta_{\mathcal{A}})$ and a set of boundary nodes Σ', a macro-step transition relation for the synchronized data-flows is defined as:

$$\delta_{Macro} = \{(p, R) \xrightarrow{\lambda} (q, R \setminus f) \mid p \xrightarrow{g|f} q \in \delta_{\mathcal{A}},\ R \subseteq \Sigma',\ \widehat{R} \leq g \setminus \widehat{\widehat{\Sigma}},\ \lambda = \mathbf{Ext}(\mathbf{t}(p \xrightarrow{g|f} q))\}$$

As an example of obtaining a macro-step transition relation, let us recall a transition $(q, e) \xrightarrow{a|a,\lambda_1} (q, f)$ with $\lambda_1 = (\{a\}, \{b, c\}, \gamma ab)\ ;\ (\{b, c\}, \emptyset, \gamma cF)$ in Fig. 11. Given the guard $g = a$ and the set of boundary nodes $\Sigma' = \{a, d\}$, $g \setminus \widehat{\widehat{\Sigma}} = a \setminus \overline{a}\overline{b}\overline{c}\overline{d} = a$, and R is $\emptyset$, $\{a\}$, $\{d\}$, or $\{a, d\}$. Thus,

$$\widehat{R} = \begin{cases} a & \text{if } R = \{a\} \\ d & \text{if } R = \{d\} \\ ad & \text{if } R = \{a, d\} \\ \top & \text{otherwise} \end{cases}$$

Then, $\widehat{R} \leq g \setminus \widehat{\widehat{\Sigma}}$ is satisfied when R is either $\{a\}$ or $\{a, d\}$, i.e., $a \leq a$ and $ad \leq a$. This generates the following macro-step transitions $((q, e), \{a\}) \xrightarrow{\lambda_1} ((q, f), \emptyset)$ and $((q, e), \{a, d\}) \xrightarrow{\lambda_1} ((q, f), \{d\})$, and these transitions are represented as dashed transitions in the state diagram in Fig. 10 as follows:



We explicate a macro-step transition with a number of micro-step transitions, each of which corresponds to a single data-flow. This refinement yields auxiliary states between the source and the target states of the macro-step transition. Let (p, R) be a source state for a data-flow corresponding to a 3-tuple θ. Then the generated auxiliary states are defined as $(p_\theta, R \setminus nodes(\theta))$ where p_θ is just a label denoting that data-flows corresponding to θ have occurred, and the function $nodes : \Lambda \rightarrow 2^{\Sigma}$ is defined as:

$$nodes(\lambda) = \begin{cases} i(\theta) \cup o(\theta) & \text{if } \lambda = \theta \\ nodes(\lambda_1) \cup nodes(\lambda_2) & \text{if } \lambda = \lambda_1; \lambda_2 \ \vee \lambda = \lambda_1|\lambda_2 \end{cases}$$

The set of such auxiliary states is obtained as $S_M = states((p, R) \xrightarrow{\lambda} (q, R'))$ where

$$states((p, R) \xrightarrow{\lambda} (q, R')) = \begin{cases} \{(p, R), (q, R')\} & \text{if } \lambda = \theta \\ \bigcup states(m)\ \forall m \in div((p, R) \xrightarrow{\lambda} (q, R')) & \text{otherwise} \end{cases}$$

The function $div : \delta_{Macro} \rightarrow 2^{\delta_{Macro}}$ is defined as:

$$div((p, R) \xrightarrow{\lambda} (q, R')) = \begin{cases} \{(p, R) \xrightarrow{\theta} (q, R')\} & \text{if } \lambda = \theta \ \wedge\ \nexists (p, R) \xrightarrow{\theta} (p', R') \in \delta_{Macro} \\ div((p, R) \xrightarrow{\lambda_1} (p_{\lambda_1}, R'')) \cup div((p_{\lambda_1}, R'') \xrightarrow{\lambda_2} (q, R')) & \\ & \text{if } \lambda = \lambda_1; \lambda_2 \text{ where } R'' = R \setminus nodes(\lambda_1) \\ \{m_1 \bowtie m_2 \mid m_i \in div((p, R) \xrightarrow{\lambda_i} (p_{\lambda_i}, R'')),\ i \in \{1, 2\}\} & \\ & \text{if } \lambda = \lambda_1 | \lambda_2 \text{ where } R'' = R \setminus nodes(\lambda_i) \\ \emptyset & \text{otherwise} \end{cases}$$

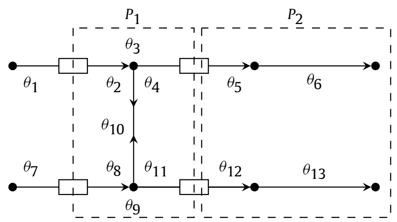
where the function $\bowtie: \delta_{Macro} \times \delta_{Macro} \rightarrow 2^{\delta_{Macro}}$ computes all interleaving compositions of the two transitions as follows. For a transition $(p, R) \xrightarrow{\lambda_1|\lambda_2} (q, R') \in \delta_{Macro}$, $(p, R) \xrightarrow{\lambda_1} (p_{\lambda_1}, R \setminus nodes(\lambda_1))$ and $(p, R) \xrightarrow{\lambda_2} (p_{\lambda_2}, R \setminus nodes(\lambda_2))$ correspond to, respectively, m_1 and m_2 of the third condition in the definition of the div function. While m_1 and m_2 are handled by the div function recursively, some auxiliary states, i.e., $states(m_1)$ and $states(m_2)$, are generated. In the interleaving composition, m_1 can occur at any states that are generated by $states(m_2)$, and vice-versa. This interleaving composition of m_1 and m_2 is represented as:

$$\begin{aligned} m_1 \bowtie m_2 = \{\ & div\big((p_1, R_1) \xrightarrow{\lambda_2} (p_{(1,\lambda_2)}, R \setminus nodes(\lambda_2))\big), \\ & div\big((p_2, R_2) \xrightarrow{\lambda_1} (p_{(2,\lambda_1)}, R \setminus nodes(\lambda_1))\big) \ \mid \\ & \quad (p_1, R_1) \in states(m_1) \text{ and } (p_2, R_2) \in states(m_2)\ \} \end{aligned}$$

The following example shows the application of the function div to a non-trivial delay-sequence, which contains a combination of sequential and parallel compositions.

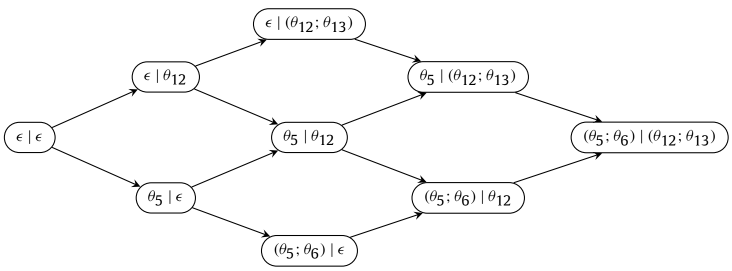
Example 4.1. Consider the Stochastic Reo connector shown below. Every indexed θ is a rate for its respective processing activity, e.g., θ_2 is the rate at which the top-left FIFO1 dispenses data through its sink end; θ_3 is the rate at which the node replicates its incoming data, etc. Data-flows contained in boxed regions marked as P_1 and P_2 appear in δ_{Macro}, derived from the Stochastic Reo Automaton of this connector, as two transitions with the delay-sequences of λ_1 and λ_2 where:

- from P_1: $\lambda_1 = ((\theta_2; \theta_3)|(\theta_8; \theta_9))\ ;\ (\theta_4|\theta_{10}|\theta_{11})$
- from P_2: $\lambda_2 = (\theta_5; \theta_6)\ |\ (\theta_{12}; \theta_{13})$



To derive a CTMC, λ_1 and λ_2 must be divided into micro-step transitions. We exemplify a few of these divisions. For λ_1, the division of $(\theta_4|\theta_{10}|\theta_{11})$ is trivial since it contains only simple parallel composition. This division result is then appended to the division result of $(\theta_2; \theta_3)|(\theta_8; \theta_9)$, which has the same structure as that of λ_2. Thus, we show below the division result of λ_2 only.

In the following CTMC fragment, to depict which events have occurred up to a current state, the name of each state consists of the delays of all the events that have occurred up to that state. The delay for a newly occurring event is appended at the end of its respective segment in the current state name.



This example shows that when a delay-sequence λ is generated by parallel composition, the events in one of the sub-delay-sequences of λ occur independently of the events in other sub-delay-sequences. Still events preserve their occurrence order within the sub-delay-sequence that they belong to. ■

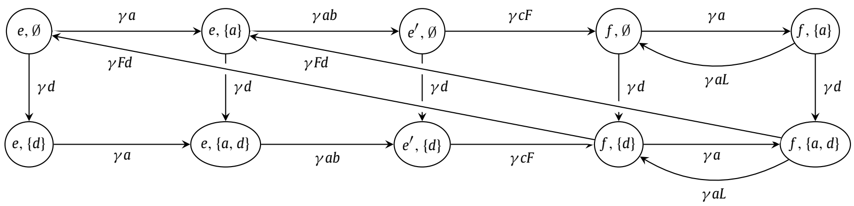


Fig. 12. Derived CTMC of LossyFIFO1.

The division into micro-step transitions ensures that each transition has a single 3-tuple in its label. Thus, the micro-step transitions can be extracted as:

$$\delta_{Proc} = \{(p, R) \xrightarrow{v(\theta)} (p', R') \mid (p, R) \xrightarrow{\theta} (p', R') \in div(t) \text{ for all } t \in \delta_{Macro}\}$$

Synchronized data-flows in Stochastic Reo Automata are considered atomic, hence other events cannot interfere with them. However, splitting these data-flows allows non-interfering events to interleave with their micro-steps, disregarding the strict sense of their atomicity. For example, a certain boundary node unrelated to a group of synchronized data-flows can accept a data item between any two micro-steps. Since we want to allow such interleaving, we must explicitly add such data-arrivals. For a Stochastic Reo Automaton $(\mathcal{A}, \mathbf{r}, \mathbf{t})$ with $\mathcal{A} = (\Sigma, Q, \delta_{\mathcal{A}})$ and a set of micro-step states S_M, its full set of data-arrival transitions, including its preliminary data-arrival set δ'_{Arr}, is defined as:

$$\delta_{Arr} = \delta'_{Arr} \cup \{(p, R) \xrightarrow{\mathbf{r}(d)} (p, R \cup \{d\}) \mid (p, R), (p, R \cup \{d\}) \in S_M,\ d \in \Sigma,\ d \notin R\}$$

The derived CTMC model can be used for stochastic analysis. For instance, Fig. 13 is obtained from PRISM[7] [8] using the CTMC model (see Fig. 12) derived from the Stochastic Reo connector of the LossyFIFO1 example in Fig. 6. Fig. 13 shows how the probability of data loss varies as the arrival rate at node a increases.

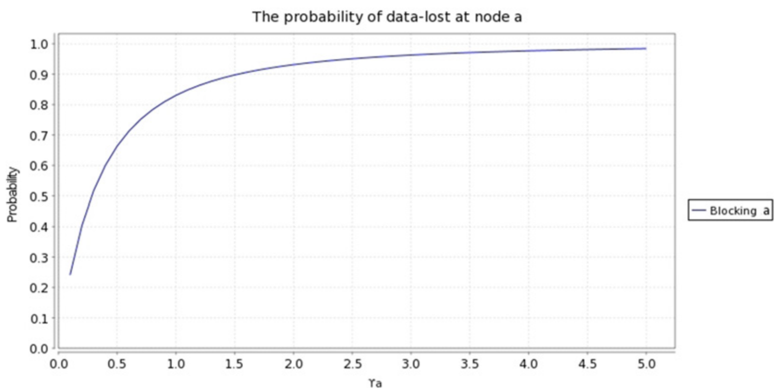
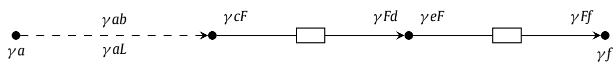


Fig. 13. Probability of data lost at node a.

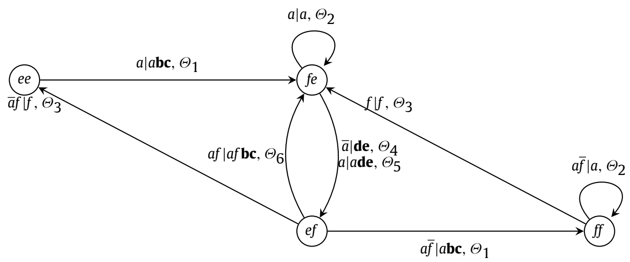
5. Example

As an example for the whole process mentioned in the previous sections, we use the Stochastic Reo connector of a task queue with minimum capacity of 2:



7 http://www.prismmodelchecker.org/.

The Stochastic Reo Automaton corresponding to this Stochastic Reo connector is given below. In Stochastic Reo Automata, the nodes joined by synchronization disappear, but, to facilitate the understanding of the behavior of the queue, we keep the joined nodes in the labels of their respective firing transitions and highlight them in bold. An indexed Θ represents the composite delay information relevant for its respective firing transition.



$\Theta_1: \{(\{a\}, \{b, c\}, \gamma ab), (\{b, c\}, \emptyset, \gamma cF)\}$
$\Theta_2: \{(\{a\}, \emptyset, \gamma aL)\}$ $\quad \Theta_3: \{(\emptyset, \{f\}, \gamma Ff)\}$
$\Theta_4: \{(\emptyset, \{d, e\}, \gamma Fd), (\{d, e\}, \emptyset, \gamma eF)\}$
$\Theta_5: \Theta_2 \cup \Theta_4$ $\quad \Theta_6: \Theta_1 \cup \Theta_3$

5.1. Stochastic analysis

The CTMC model derived from the Stochastic Reo Automaton corresponding to the Stochastic Reo connector of the task queue appears in Fig. 14. This figure shows the derived CTMC model as input to the PRISM tool for stochastic analysis.

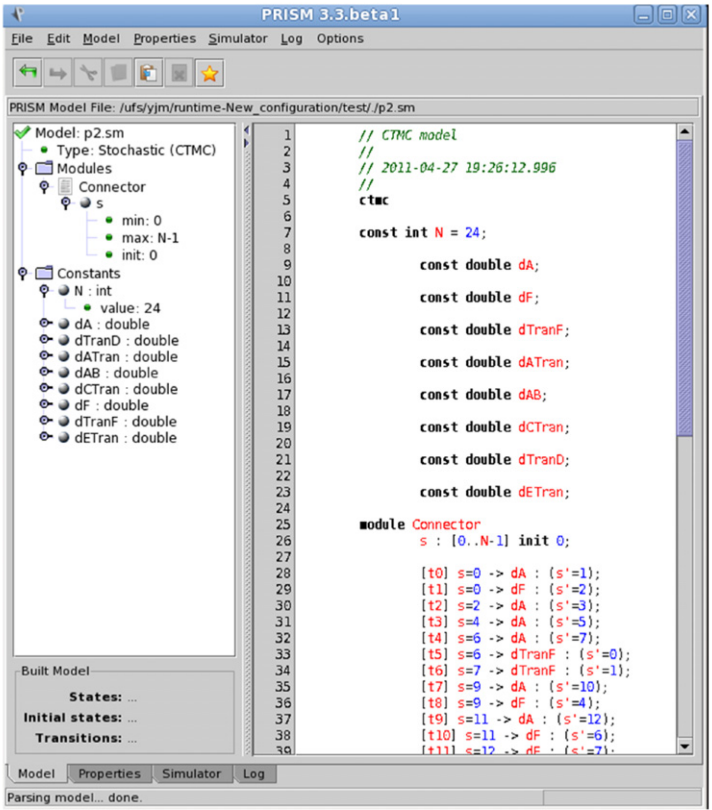


Fig. 14. CTMC for a task queue in PRISM.

In PRISM, properties of models are expressed using operations such as P and S operators: the P operator is used to reason about the probability of the occurrence of a certain event; the S operator is used to reason about the steady-state behavior of a model. In addition, labels are used to concisely express the formulas representing the properties of a model. The following labels are used to express some properties later.

- *Qsize* represents how many tasks are in the task queue.
- *MaxSize* is the capacity of the task queue, i.e., 2 in this example.
- *DataLost* represents how many tasks are lost in the task queue.
- *MaxDataLost* is a fixed maximum for losing data items in the task queue.

We have analyzed the derived CTMC model with the following properties of the queue:

1. $S = ?$ [$(Qsize/MaxSize) > 0.5$]
This formula is a PRISM query asking the steady-state probability that the queue is more than 50% full (i.e., contains at least one task). As seen in Fig. 15, when the task-arrival rate at the queue is twice the rate at which tasks are handled, the answer is 0.438.

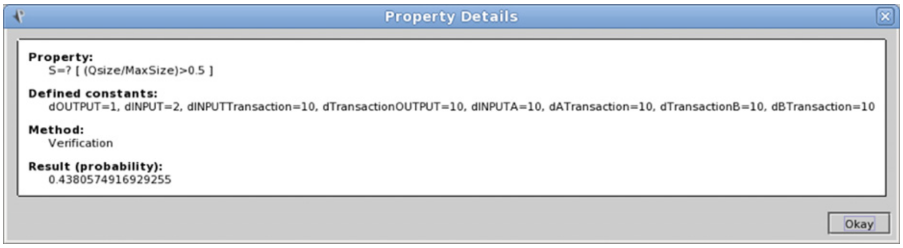


Fig. 15. Analysis result of Property 1.

2. $S = ?$ [$DataLost = MaxDataLost$]
Fig. 16 shows the variation of the steady-state probability that the queue loses new incoming tasks because it reached its capacity. Here, we fix the task-arrival rate and vary the task-handling rate.

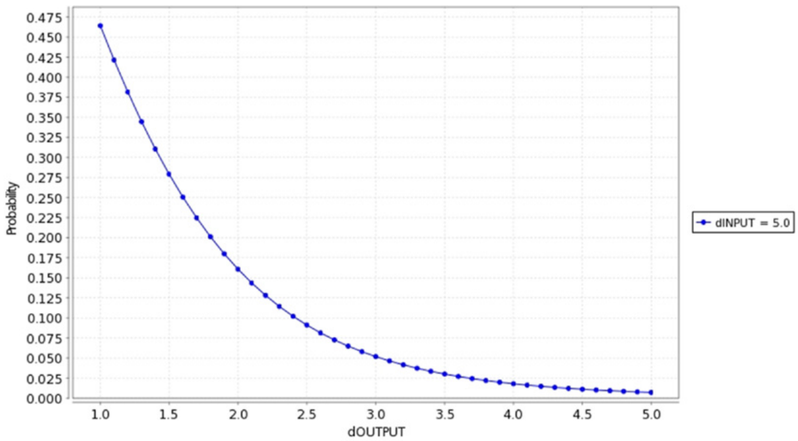


Fig. 16. Analysis result of Property 2.

This example shows the analysis of the task queue with variable rates. If we put the actual arrival and processing rates of a real deployed system in this derived CTMC model, we can determine, e.g., whether or not the number of available server components is sufficient to process the incoming tasks efficiently.

6. Interactive Markov Chains

Interactive Markov Chains (IMCs) are a compositional stochastic model [6] which can be used to provide quantitative semantics to concurrent systems. In IMC models, delays can be represented by combinations of exponential delay transitions, it allows to accommodate non-exponential distributions within the models. That is, it can represent delays from the larger class of phase-type distributions [9,10] which can approximate general continuous distributions. This enables a more general usage of Stochastic Reo Automata, if IMCs are used instead of CTMCs as the translation target of Stochastic Reo Automata models.

In this section, we discuss to what extent IMCs are an appropriate semantic model for Stochastic Reo, instead of Stochastic Reo Automata. In addition, we provide a translation from Stochastic Reo into IMCs, which enables the use of the latter as an alternative target stochastic model.

An IMC specifies a reactive system and is formally described as a tuple $(S, Act, \rightarrow, \Rightarrow, s_0)$ where S is a finite set of states; Act is a set of actions; s_0 is an initial state in S; $\rightarrow$ and $\Rightarrow$ are two types of transition relations:

- $\rightarrow \subseteq S \times Act \times S$ for *interactive* transitions and
- $\Rightarrow \subseteq S \times \mathbb{R}^+ \times S$ for *Markovian* transitions.

Thus, an IMC is a Labeled Transition System (LTS) if $\Rightarrow = \emptyset$ and $\rightarrow \neq \emptyset$, and is a CTMC if $\Rightarrow \neq \emptyset$ and $\rightarrow = \emptyset$.

Compared to other stochastic models such as CTMCs, the main strength of IMCs is their compositionality. Thus, one can generate a complex IMC as the composition of relevant simple IMCs, which enables compositional specification of complex systems.

Definition 6.1 (*Product [6]*). Given two IMCs $\mathcal{I}_1 = (S_1, Act_1, \rightarrow_1, \Rightarrow_1, s_{(1,0)})$ and $\mathcal{I}_2 = (S_2, Act_2, \rightarrow_2, \Rightarrow_2, s_{(2,0)})$, the composition of $\mathcal{I}_1$ and $\mathcal{I}_2$ over a set of actions A is defined as $\mathcal{I}_1 \times \mathcal{I}_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, \Rightarrow, s_{(1,0)} \times s_{(2,0)})$ where $\rightarrow$ and $\Rightarrow$ are defined as:

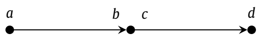
$$\begin{aligned}
\rightarrow \; = \; & \{(s_1, s_2) \xrightarrow{\alpha} (s_1', s_2') \mid \alpha \in A,\ s_1 \xrightarrow{\alpha}_1 s_1' \ \wedge\ s_2 \xrightarrow{\alpha}_2 s_2'\} \\
\cup \; & \{(s_1, s_2) \xrightarrow{\alpha} (s_1', s_2) \mid \alpha \notin A,\ s_2 \in S_2,\ s_1 \xrightarrow{\alpha}_1 s_1'\} \\
\cup \; & \{(s_1, s_2) \xrightarrow{\alpha} (s_1, s_2') \mid \alpha \notin A,\ s_1 \in S_1,\ s_2 \xrightarrow{\alpha}_2 s_2'\} \\
\Rightarrow \; = \; & \{(s_1, s_2) \overset{\lambda}{\Rightarrow} (s_1', s_2) \mid s_2 \in S_2\ ,\ s_1 \overset{\lambda}{\Rightarrow}_1 s_1'\} \\
\cup \; & \{(s_1, s_2) \overset{\lambda}{\Rightarrow} (s_1, s_2') \mid s_1 \in S_1\ ,\ s_2 \overset{\lambda}{\Rightarrow}_2 s_2'\}
\end{aligned}$$

The product of interactive transitions is similar to ordinary automaton product, which includes interleaving and synchronized compositions of interactive transitions. The product of Markovian transitions consists only of interleaved transitions.

We now discuss IMCs from two different perspectives:

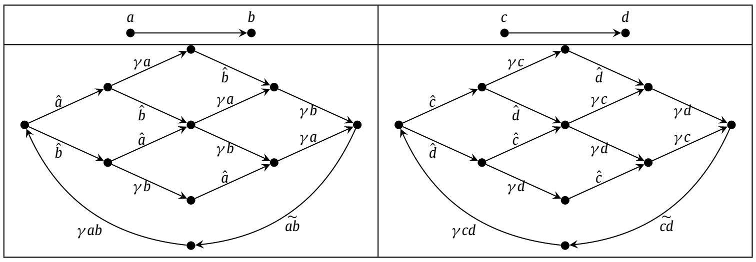
1. as a semantic model for Stochastic Reo: translating basic Stochastic Reo channels into IMCs and then composing the derived IMCs using the product operation defined above; or
2. as an alternative translation target model: composing the Stochastic Reo Automata of basic channels and then translating the composed Stochastic Reo Automaton into an IMC.

We show now that the first case is not adequate since it provides a wrong semantics for connectors that involve propagation of synchrony. For example, consider the following connector, denoted as 2sync, that consists of two Sync channels joined at nodes b and c.



The behavior of basic channels consists of data-arrivals and data-flows which occur sequentially, i.e., data-flows follow data-arrivals. Both data-arrivals and data-flows are divided into two phases: an action and the random processing delay for each action. For instance, a data-arrival at node a consists of the arrival action at node a and waiting for the acceptance from node a. To reason about the end-to-end QoS, the IMCs for each Sync channel must have Markovian transitions for the random processing delays of both data-arrivals and data-flows. The two phases of channels must be considered sequentially, that is, the phase of random processing delays follows that of the action. Table 2 shows the possible IMCs for the Sync channels ab and cd.

Table 2
IMCs for each Sync channel.



Here, we use '^' and '~' over node names in order to represent data-arrivals and data-flows, respectively. Rates for each data-arrival and each data-flow are represented with the prefix γ.

However, the composition of the IMCs for the two Sync channels does not capture the correct behavior of 2sync as specified by Reo. Fig. 17 shows a fragment of the IMC product result. Note that, for simplicity, here and throughout, the rest of the product result is omitted and represented as a cloud shape.

If we apply the assumption that the synchronization by joining nodes is an immediate action, then transitions with $(\hat{b}, \hat{c})$, γb, and γc labels are considered internal interactive transitions or discarded by certain refinements before or after the product. The result of the product and certain refinements is depicted in Fig. 18.

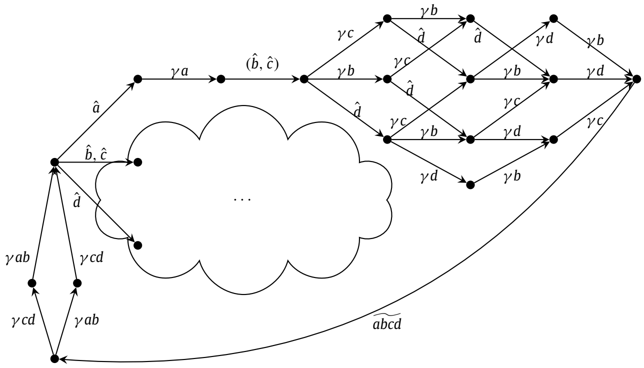


Fig. 17. Composed IMC for 2sync.

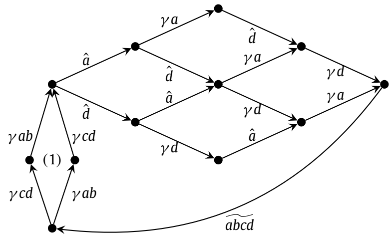


Fig. 18. IMC after refinements on 2sync.

Consider the diamond shape (1) in Fig. 18, formed by the two data-flows from a to b (γab) and from c to d (γcd), which occur interleaved. In the 2sync connector, these two data-flows occur sequentially, which means that data-flows do not occur concurrently. This example illustrates that using the concurrent composition of IMCs is not appropriate for specifying the behavior of connectors because propagation of synchrony is not properly modeled. It is natural and interesting to consider whether it is possible to adapt the composition operator of IMCs in order to delete unintended transitions and still remain a compositional model. However, we did not investigate this possibility since it is out of the scope of this paper.

We now show how IMCs can be used as a target stochastic model, instead of CTMCs. In this approach, the synchronization is considered in Stochastic Reo Automata, and we do not need to consider the IMC level refinements for synchronization such as the transitions with $(\hat{b}, \hat{c})$, γb, and γc labels in Fig. 17.

Since a Stochastic Reo Automaton does not have an initial state, the derived result is precisely an IMC transition system (IMCTS) [6], i.e., an IMC without an initial state. However, an initial state can be decided by the interpretation of the behavior of a connector. Thus, in this paper, we consider the IMCTS derived from a Stochastic Reo Automaton as an IMC. An IMC derived from a Stochastic Reo Automaton $(\mathcal{A}, \mathbf{r}, \mathbf{t})$ with $\mathcal{A} = (\Sigma, Q, \delta_{\mathcal{A}})$ is a tuple $(S, Act, \rightarrow, \Rightarrow)$ where $S = S_A \cup S_M$ is the set of states. S_A represents the configurations of the system derived from its Stochastic Reo Automaton and the pending status of I/O requests; S_M is the set of states that represent the occurrences of synchronized data-flows and result from the micro-step divisions of the synchronized data-flows. In general, Act is a set of actions of the arrival of a data item at a boundary node and synchronized data-flows through a connector. Thus, $Act = \Sigma' \cup Frs$ where Σ' is a set of boundary nodes, and Frs is a set of firings, e.g., for f in a label on every transition $s \xrightarrow{g|f} s' \in \delta_{\mathcal{A}}, f \in Frs$. The relation $\rightarrow = \delta_{Arr} \cup \delta_{Proc} \subseteq S \times \mathbb{R}^+ \times S$ is a set of Markovian transitions, and $\Rightarrow = \zeta_{Arr} \cup \zeta_{Proc} \subseteq (S \times 2^{\Sigma'} \times S) \cup (S \times 2^{Frs} \times S)$ is a set of interactive transitions. The sets indexed with Arr and $Proc$ represent transitions for data-arrivals and data-flows, respectively.

A state in $S \subseteq Q \times 2^{\Sigma'} \times 2^{\Sigma'}$ represents three kinds of configurations: configurations of a connector (Q), the occurrence of actions (first $2^{\Sigma'}$), and the presence of the I/O requests pending on its boundary nodes (second $2^{\Sigma'}$), if any. The set of S_A and the preliminary sets of data-arrival transitions are defined as:

$$\begin{aligned} S_A \; &= \{(q, A, P) \mid q \in Q,\ P \subseteq A \subseteq \Sigma'\} \\ \zeta'_{Arr} &= \{(q, A, P) \xrightarrow{\hat{c}} (q, A \cup \{c\}, P) \mid (q, A, P),\ (q, A \cup \{c\}, P) \in \Sigma',\ c \notin A\} \\ \delta'_{Arr} &= \{(q, A, P) \xRightarrow{\mathbf{r}(c)} (q, A, P \cup \{c\}) \mid (q, A, P),\ (q, A, P \cup \{c\}) \in \Sigma',\ c \notin P\} \end{aligned}$$

ζ'_{Arr} and δ'_{Arr} are used to define ζ_{Arr} and δ_{Arr}, respectively, below.

As mentioned in Section 4, synchronized data-flows are described by a single transition in a Stochastic Reo Automaton. From the interactive transition perspective, the synchronized data-flows are also described by a single interactive transition.

However, from the Markovian transition perspective in a continuous time domain, a transition corresponding to multiple synchronized data-flows needs to be divided into micro-step transitions. For this purpose, we reuse a delay-sequence which is extracted by Algorithm 4.2.1. We now derive transitions for synchronized data-flows in two steps:

1. For each transition $p \xrightarrow{g|f} q \in \delta_{\mathcal{A}}$, we derive interactive and macro Markovian transitions $(p, A, P) \xrightarrow{\widetilde{f}} (p, A \setminus f, P)$ and $(p, A \setminus f, P) \xrightarrow{\lambda} (q, A \setminus f, P \setminus f)$, respectively, for every set of pending requests P that suffices to activate the guard g ($\widehat{P} \leq g \setminus \widehat{\widehat{\Sigma}}$), where λ is the delay-sequence extracted by Algorithm 4.2.1, $\mathbf{Ext}(\mathbf{t}(p \xrightarrow{g|f} q))$. The sets of derived transitions are defined below as ζ_{Macro} and δ_{Macro} for interactive and macro Markovian transitions, respectively.
2. We divide a transition $s \xrightarrow{\lambda} s' \in \delta_{Macro}$ into a combination of micro-step transitions, each of which corresponds to a single event.

Given a Stochastic Reo Automaton $(\mathcal{A}, \mathbf{r}, \mathbf{t})$ with $=(Q, \Sigma, \delta_{\mathcal{A}})$ and a set of boundary nodes Σ', a macro-step transition for synchronized data-flows is defined as:

$$\begin{aligned}\zeta_{Macro} &= \{(p, A, P) \xrightarrow{\widetilde{f}} (p, A \setminus f, P) \mid p \xrightarrow{g|f} q \in \delta_{\mathcal{A}},\ A \subseteq P \subseteq \Sigma',\ \widehat{P} \leq g \setminus \widehat{\widehat{\Sigma}}\} \\ \delta_{Macro} &= \{(p, A, P) \xrightarrow{\lambda} (q, A, P \setminus f) \ \mid p \xrightarrow{g|f} q \in \delta_{\mathcal{A}},\ A \cap f = \emptyset,\ A \subset P \subseteq \Sigma', \\ &\qquad \widehat{P} \leq g \setminus \widehat{\widehat{\Sigma}},\ \lambda = \mathbf{Ext}(\mathbf{t}(p \xrightarrow{g|f} q))\}\end{aligned}$$

To derive an IMC from a Stochastic Reo Automaton, we reuse the function *nodes* and modify the definitions of functions *states* and *div* in Section 4.3. Then, $S_M = state((p, A, P) \xrightarrow{\lambda} (q, A, P'))$ where

$$states((p, A, P) \xrightarrow{\lambda} (q, A, P')) = \begin{cases} \{(p, A, P), (q, A, P')\} & \text{if } \lambda = \theta \\ \bigcup states(m)\ \forall m \in div((p, A, P) \xrightarrow{\lambda} (q, A, P')) & \text{otherwise} \end{cases}$$

The function $div : \delta_{Macro} \rightarrow 2^{\delta_{Macro}}$ is defined as:

$$div((p, A, P) \xrightarrow{\lambda} (q, A, P')) = \begin{cases} \{(p, A, P) \xrightarrow{\theta} (q, A, P')\} \quad \text{if } \lambda = \theta \wedge \nexists (p, A, P) \xrightarrow{\theta} (p', A, P') \in \delta_{Macro} \\ div((p, A, P) \xrightarrow{\lambda_1} (p_{\lambda_1}, A, P'')) \cup div((p_{\lambda_1}, A, P'') \xrightarrow{\lambda_2} (q, A, P')) \\ \qquad \text{if } \lambda = \lambda_1; \lambda_2 \text{ where } P'' = P \setminus nodes(\lambda_1) \\ \{m_1 \boxtimes m_2 \mid m_i \in div((p, A, P) \xrightarrow{\lambda_i} (p_{\lambda_i}, A, P'')),\ i \in \{1, 2\}\} \\ \qquad \text{if } \lambda = \lambda_1 | \lambda_2 \text{ where } P'' = P \setminus nodes(\lambda_i) \\ \emptyset \qquad \text{otherwise} \end{cases}$$

where the function $\boxtimes : \delta_{Macro} \times \delta_{Macro} \rightarrow 2^{\delta_{Macro}}$ computes all interleaving compositions of the two transitions as follows. For a transition $(p, A, P) \xrightarrow{\lambda_1|\lambda_2} (q, A, P') \in \delta_{Macro}$, $(p, A, P) \xrightarrow{\lambda_1} (p_{\lambda_1}, A, P \setminus nodes(\lambda_1))$ and $(p, A, P) \xrightarrow{\lambda_2} (p_{\lambda_2}, A, P \setminus nodes(\lambda_2))$ correspond to, respectively, m_1 and m_2 of the third condition in the definition of the div function. While m_1 and m_2 are handled by the div function recursively, some auxiliary states, i.e., $states(m_1)$ and $states(m_2)$, are generated. In the interleaving composition, m_1 can occur at any states that are generated by $states(m_2)$, and vice-versa. This interleaving composition of m_1 and m_2 is represented as:

$$\begin{aligned} m_1 \boxtimes m_2 = \Big\{ & div\big((p_1, A, P_1) \xrightarrow{\lambda_2} (p_{(1,\lambda_2)}, A, P_1 \setminus nodes(\lambda_2))\big), \\ & div\big((p_2, A, P_2) \xrightarrow{\lambda_1} (p_{(2,\lambda_1)}, A, P_2 \setminus nodes(\lambda_1))\big) \ \mid \\ & \quad (p_1, A, P_1) \in states(m_1) \text{ and } (p_2, A, P_2) \in states(m_2) \Big\} \end{aligned}$$

This composition is similar way to the function $\bowtie$ explained in Section 4.3. The only difference between these two functions is the structure of their states: CTMC states are elements of $Q \times 2^{\Sigma'}$, whereas IMC states are in $Q \times 2^{\Sigma'} \times 2^{\Sigma'}$ where Σ' is a set of boundary nodes in a Stochastic Reo connector.

The division into micro-step transitions ensures that each transition has a single 3-tuple in its label. Thus, the micro-step transitions can be extracted as:

$$\begin{aligned} \delta_{Proc} = \{ & (p, A, P) \xrightarrow{v(\theta)} (p', A, P') \mid \\ & (p, A, P) \xrightarrow{\theta} (p', A, P') \in div(t) \text{ for all } t \in \delta_{Macro}\} \end{aligned}$$

As mentioned above, interactive transitions in ζ_{Macro} do not need to be divided, thus, $\zeta_{Proc} = \zeta_{Macro}$.

Splitting synchronized data-flows allows non-interfering events to interleave with their micro-steps, disregarding the strict sense of their atomicity. In order to allow such interleaving, we must explicitly add such data-arrivals. For a Stochastic

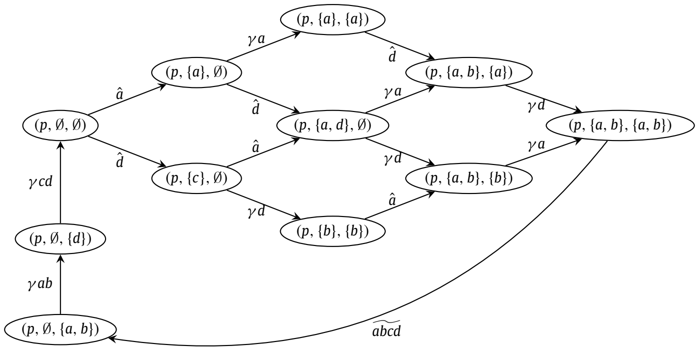


Fig. 19. Derived IMC for 2sync.

Reo Automaton $(\mathcal{A}, \mathbf{r}, \mathbf{t})$ with $\mathcal{A} = (\Sigma, Q, \delta_{\mathcal{A}})$ and a set of micro-step states S_M, its full sets of data-arrival transitions, including its data-arrivals, are defined as:

$$\begin{aligned}\zeta_{Arr} &= \zeta'_{Arr} \cup \{(p, A, P) \xrightarrow{\hat{d}} (p, A \cup \{d\}, P) \mid \\ &\qquad (p, A, P), (p, A \cup \{d\}, P) \in S_M,\ d \in \Sigma,\ d \notin A\} \\ \delta_{Arr} &= \delta'_{Arr} \cup \{(p, A, P) \xRightarrow{\mathbf{r}(d)} (p, A, P \cup \{d\}) \mid \\ &\qquad (p, A, P), (p, A, P \cup \{d\}) \in S_M,\ d \in \Sigma,\ d \notin P\}\end{aligned}$$

Applying this method, Fig. 19 shows the IMC corresponding to our 2sync example. The derived result is similar to the IMC for a Sync in Table 2 and captures the correct behavior of the 2sync connector.

The foregoing illustrates that IMCs can serve as another alternative target model for the translation from Stochastic Reo Automata, instead of CTMCs. Although doing so does not use the compositionality of IMCs, translation into IMCs is still meaningful. The derived IMCs, for instance, can represent not only exponential distributions, but also non-exponential distributions, especially phase-type distributions. The analysis of IMCs is supported by tools such as the Construction and Analysis of Distributed Processes (CADP) [11]. CADP verifies the functional correctness of the specification of system behavior and also minimizes IMCs effectively [12]. Moreover, IMCs can be used in various other applications, such as Dynamic Fault Trees (DFTs) [13–15], Architectural Analysis and Design Language (AADL) [16–18], and so on [19].

7. Related work

The research in formal specification of system behavior with quantitative aspects encompasses a variety of developments such as Stochastic Process Algebras (SPAs) [20], Stochastic Automata Networks (SANs) [21,22], and Stochastic Petri nets (SPNs) [23,24]. SPA is a model for both qualitative and quantitative specification and analysis with a compositional and hierarchical framework. It has algebraic laws (the so-called static laws) and expansion laws which express parallel compositions in terms of SPA operators. In SPA the interpretation of the parallel composition is a vexed one because it allows various interpretations such as Performance Evaluation Process Algebra (PEPA) [25], and Extended Markovian Process Algebra (EMPA) [26,27]. SPA describes '*how*' each process behaves, while (Stochastic) Reo directly describes '*what*' communication protocols connect and coordinate the processes in a system, in terms of primitive channels and their composition. Therefore, (Stochastic) Reo explicitly models the pure coordination and communication protocols including the impact of real communication networks on software systems and their interactions. Compared to SPA, our approach more naturally leads to a formulation using queueing models.

SPN is widely used for modeling concurrency, synchronization, and precedence, and is conducive to both top-down and bottom-up modeling. Stochastic Reo shares the same properties with SPN and natively supports composition of synchrony and exclusion together with asynchrony. The topology of connectors in (Stochastic) Reo is inherently dynamic, and it accommodates mobility as described in [28]. Moreover, (Stochastic) Reo supports a liberal notion of channels, which allows to express synchrony and asynchrony, which can be viewed as specialized channel-based models that incorporate certain built-in primitive coordination constructs.

SAN consists of a number of stochastic automata each of which acts independently. Thus, the state of a SAN at time t is expressed by the states of each automaton at time t. The concept of a collection of individual automata helps modeling distributed and parallel systems more easily. The interactions in SANs are rather limited to patterns like synchronizing events or operating at different rates. Compared with the SAN approach, the expressiveness of (Stochastic) Reo makes it possible to model different interaction patterns involving both asynchronous and synchronous communications. We remark however that in SAN, as in other synchronous models, asynchronous communication can be partially achieved by modeling bounded FIFO channels and replacing direct synchronous communication with indirect asynchronous communication via the FIFO channels.

Continuous-Time Constraint Automata (CCA) [29] are another stochastic extension of CA which support reasoning about QoS aspects such as expected response times. CCA are close to IMCs in that they distinguish between interactive transitions and Markovian transitions. In CCA, data-arrivals and data-flows in connectors are represented by interactive transitions, and processing data in components is represented by Markovian transitions. Processing data in each component is independent of processing in the others. Thus, interleaving composition of Markovian transitions is appropriate. The stochastic extension in CCA focuses on internal behavior of a connector, but it does not take into account the interaction with the environment, i.e., the arrivals of I/O requests at the ends of a connector as stochastic processes. Reasoning about the end-to-end QoS of system behavior requires incorporation of such stochastic processes. In addition, CCA do not capture the context-dependency of a Reo connector, i.e., it is possible for CCA models to have unintended transitions. Compared to such CCA, Stochastic Reo Automata not only specify the end-to-end QoS of a Reo connector, but also capture context-dependent behavior.

8. Conclusions and future work

We introduced Stochastic Reo Automata by extending Reo Automata with functions that assign stochastic values of arrival rates and processing delay rates to boundary nodes and channels in Stochastic Reo. This model is very compact compared to the existing models, e.g., in [4]. Various formal properties of our model are obtained, reusing the formal justifications of the various properties of Reo Automata [3], such as compositionality.

The technical core in this paper shows the complexity of the original problem whence it stems from: derivation of stochastic models for formal analysis of end-to-end QoS properties of systems composed of services/components supplied by disparate providers, in their user environments. This complexity highlights the gross inadequacy of informal, or one-off techniques and emphasizes the importance of formal approaches and sound models that can serve as the basis for automated tools.

Stochastic Reo does not impose any restriction on the distribution of its annotated rates such as the rates for data-arrivals at channel ends or data-flows through channels. However, for translation of Stochastic Reo to a homogeneous CTMC model, we considered only the exponential distributions for the rates. For more general usage of Stochastic Reo Automata, we also want to consider non-exponential distributions by considering phase-type distributions or using Semi-Markov Processes [30] as target models of our translation. A simulation engine [31], already integrated into our toolset, Extensible Coordination Tools (ECT) [32] environment, supports a wide variety of more general distributions for Stochastic Reo. We discussed why IMCs are not an appropriate semantic model for Stochastic Reo, and showed the translation from Stochastic Reo into IMCs via Stochastic Reo Automata. A natural and interesting future work is to consider whether it is possible to adapt the composition operator of IMCs in order to delete unintended transitions that it currently produces in synchrony propagation scenarios, and still remain within a compositional framework. In addition, we plan to consider rewards of a system along with its stochastic behavior as well. Our translation result will thus become a CTMC model with reward information on its transitions and states, which can be fed into an appropriate stochastic analysis tool, such as PRISM. As an example, the translation of the Stochastic Reo connector of the task queue with a maximum capacity 2 into a CTMC model, using Stochastic Reo Automata, reported in this paper was carried out manually. We have already incorporated tools for this translation using QIA (instead of Stochastic Reo Automata) within our ECT environment. We are currently extending and improving these tools to use our Stochastic Reo Automata. The more compact sizes of the automata models will then allow us to analyze larger system.

Acknowledgements

We would like to thank the referees for the many constructive comments, which greatly helped us improving the paper. The second author was partially supported by Fundação para a Ciência e a Tecnologia, Portugal, under grant number SFRH/BPD/71956/2010.

References

[1] F. Arbab, Reo: a channel-based coordination model for component composition, Mathematical Structures in Computer Science 14 (3) (2004) 329–366.
[2] C. Baier, M. Sirjani, F. Arbab, J.J.M.M. Rutten, Modeling component connectors in Reo by constraint automata, Science Computer Programming 61 (2) (2006) 75–113.
[3] M. Bonsangue, D. Clarke, A. Silva, A model of context-dependent component connectors, Science of Computer Programming (in press) Corrected Proof. doi:10.1016/j.scico.2011.01.006.
[4] F. Arbab, T. Chothia, R. van der Mei, S. Meng, Y.-J. Moon, C. Verhoef, From coordination to stochastic models of QoS, in: COORDINATION, in: Lecture Notes in Computer Science, vol. 5521, Springer, 2009, pp. 268–287.
[5] Y.-J. Moon, A. Silva, C. Krause, F. Arbab, A compositional semantics for stochastic Reo connectors, in: FOCLASA, in: EPTCS, vol. 30, 2010, pp. 93–107.
[6] H. Hermanns, Interactive Markov Chains: The Quest for Quantified Quality, in: Lecture Notes in Computer Science, vol. 2428, Springer, 2002.
[7] W.J. Stewart, Introduction to the Numerical Solution of Markov Chains, Princeton University Press, 1994.
[8] A. Hinton, M.Z. Kwiatkowska, G. Norman, D. Parker, PRISM: a tool for automatic verification of probabilistic systems, in: TACAS, in: Lecture Notes in Computer Science, vol. 3920, Springer, 2006, pp. 441–444.
[9] C.A. O'Cinneide, Characterization of phase-type distributions, in: Stochastic Models, vol. 6, Taylor & Francis, 1990, pp. 1–57.
[10] M.F. Neuts, Matrix-geometric Solutions in Stochastic Models: An Algorithmic Approach, The Johns Hopkins University Press, 1981.
[11] H. Garavel, R. Mateescu, F. Lang, W. Serwe, CADP 2006: a toolbox for the construction and analysis of distributed processes, in: CAV, in: Lecture Notes in Computer Science, vol. 4590, Springer, 2007, pp. 158–163.
[12] H. Garavel, H. Hermanns, On combining functional verification and performance evaluation using CADP, in: FME, in: Lecture Notes in Computer Science, vol. 2391, Springer, 2002, pp. 410–429.

[13] H. Boudali, P. Crouzen, M. Stoelinga, A compositional semantics for dynamic fault trees in terms of interactive Markov chains, in: ATVA, in: Lecture Notes in Computer Science, vol. 4762, Springer, 2007, pp. 441–456.
[14] H. Boudali, P. Crouzen, M. Stoelinga, Dynamic fault tree analysis using input/output interactive Markov chains, in: International Conference on Dependable Systems and Networks, IEEE Computer Society, 2007, pp. 708–717.
[15] H. Boudali, P. Crouzen, M. Stoelinga, A rigorous, compositional, and extensible framework for dynamic fault tree analysis, IEEE Transactions on Dependable and Secure Computing 7 (2) (2010) 128–143.
[16] H. Boudali, P. Crouzen, B.R. Haverkort, M. Kuntz, M. Stoelinga, Architectural dependability evaluation with Arcade, in: DSN, IEEE Computer Society, 2008, pp. 512–521.
[17] M. Bozzano, A. Cimatti, M. Roveri, J.-P. Katoen, V.Y. Nguyen, T. Noll, Codesign of dependable systems: a component-based modeling language, in: MEMOCODE'09: Proceedings of the 7th IEEE/ACM international conference on Formal Methods and Models for Codesign, IEEE Press, Piscataway, NJ, USA, 2009, pp. 121–130.
[18] M. Bozzano, A. Cimatti, J.-P. Katoen, V.Y. Nguyen, T. Noll, M. Roveri, The COMPASS approach: correctness, modelling and performability of aerospace systems, in: SAFECOMP, in: Lecture Notes in Computer Science, vol. 5775, Springer, 2009, pp. 173–186.
[19] H. Hermanns, J.-P. Katoen, The how and why of interactive Markov chains, in: Formal Methods for Components and Objects (FMCO), in: Lecture Notes in Computer Science, vol. 6286, Springer-Verlag, 2010, pp. 311–337.
[20] M. Calzarossa, S. Tucci (Eds.), Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002, Tutorial Lectures, in: Lecture Notes in Computer Science, vol. 2459, Springer, 2002.
[21] P. Fernandes, B. Plateau, W.J. Stewart, Efficient descriptor-vector multiplications in stochastic automata networks, Journal of the ACM 45 (3) (1998) 381–414.
[22] W.J. Stewart, K. Atif, B. Plateau, The numerical solution of stochastic automata networks, European Journal of Operational Research 86 (3) (1995) 503–525.
[23] B.R. Haverkort, R. Marie, G. Rubino, K.S. Trivedi (Eds.), Performability Modelling: Techniques and Tools, Wiley, 2001.
[24] R.A. Sahner, K.S. Trivedi, A. Puliafito, Performance and Reliability Analysis of Computer Systems: An Example-based Approach Using the SHARPE Software Package, Kluwer Academic Publishers, Norwell, MA, USA, 1996.
[25] J. Hillston, A Compositional Approach to Performance Modelling, Cambridge University Press, 1996.
[26] M. Bernardo, R. Gorrieri, Extended Markovian process algebra, in: CONCUR, in: Lecture Notes in Computer Science, vol. 1119, Springer, 1996, pp. 315–330.
[27] M. Bernardo, R. Gorrieri, A tutorial on EMPA: a theory of concurrent processes with nondeterminism, priorities, probabilities and time, Theoretical Computer Science 202 (1–2) (1998) 1–54.
[28] C. Krause, Z. Maraikar, A. Lazovik, F. Arbab, Modeling dynamic reconfigurations in Reo using high-level replacement systems, Science of Computer Programming 76 (1) (2011) 23–36 (selected papers from the 6th International Workshop on the Foundations of Coordination Languages and Software Architectures — FOCLASA'07).
[29] C. Baier, V. Wolf, Stochastic reasoning about channel-based component connectors, in: COORDINATION, in: Lecture Notes in Computer Science, vol. 4038, Springer, 2006, pp. 1–15.
[30] H.L.S. Younes, R.G. Simmons, Solving generalized semi-Markov decision processes using continuous phase-type distributions, in: Proceedings of the 19th National Conference on Artificial Intelligence, California AAAI Press, 2004, pp. 742–748.
[31] O. Kanters, QoS analysis by simulation in Reo, Master's thesis, Vrije Universiteit, Amsterdam, The Netherlands, 2010.
[32] Extensible coordination tools, http://reo.project.cwi.nl/.