Stochastic models for quality of service of component connectors

Moon, Y.J.

Chapter 2

Models for component coordination

In this section, we recall the basics of the Reo coordination language and its semantic models. We also present Stochastic Reo, an extension of Reo, which enables the modeling of QoS properties. In addition, we introduce the basic definitions of some stochastic models, in particular Markov Chains and Interactive Markov Chains which we will use later as target models for the translation from Stochastic Reo for performance analysis. We conclude this chapter with a brief discussion on related work.

2.1 Reo language

Reo is a channel-based coordination model wherein so-called *connectors* are used to coordinate (i.e., control the interaction among) components or services exogenously (from outside of those components and services). In Reo, complex connectors are compositionally built out of primitive channels. Channels are atomic connectors with exactly two ends. An end can be either a *source* or a *sink* end. 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 two source or two sink ends.

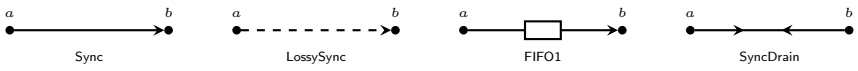


Figure 2.1: Some basic Reo channels

Figure 2.1 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. The 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. We assume that at most one request can wait for the acceptance at a boundary node. 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 of its sink ends and replicating it to all of its source ends.

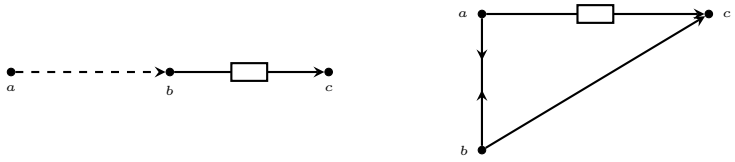


Figure 2.2: LossyFIFO1 and Ordering circuit

For example, the connectors shown in Figure 2.2 are a (overflow) LossyFIFO1 and an alternator. The LossyFIFO1 reads a data item from a, buffers it in a FIFO1 and writes to c. This connector loses data items at a if and only if the FIFO1 buffer is already full. The alternator imposes an ordering on the data from its input nodes a and b to its output node c. The SyncDrain channel enforces that data flow through a and b only synchronously. The empty buffer together with the propagation of synchrony through the three nodes guarantee that the data item obtained from b is delivered to c while the data item obtained from a is stored in the FIFO1 buffer. After this, the buffer of the FIFO1 is full and propagation of exclusion from a through the SyncDrain channel to b guarantees that data cannot flow in through either a or b, but c can dispense the data stored in the FIFO1 buffer, which makes it empty again. Assume three independent processes (that follow no communication protocol and each of which knows nothing about the others) place I/O requests on nodes a, b, and c, each according to its own internal timing. By delaying the reply to their requests, when necessary, this circuit guarantees that successive read operations at c obtain the values produced by the successive write operations at b and a alternately.

2.2 Stochastic Reo

Stochastic Reo is an extension of Reo where channels are annotated with stochastic values denoting distributions of their relevant data-flow events and arrival of I/O request at the channel ends. We refer to these distributions as processing delay rates and arrival rates of I/O requests, respectively. Such stochastic values are non-negative real values and describe the probability of a certain value (or interval) of a discrete (or

continuous) random variable. Figure 2.3 shows some primitive channels of Stochastic Reo that correspond to the primitives of Reo in Figure 2.1. In this figure and throughout, for simplicity, we do not show node names, but these names can be inferred from the names of their respective arrival rates: for instance, 'γa' refers to the node 'a'.

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 operational semantics between Reo and Stochastic Reo is quite straightforward, i.e., one-to-one mapping.

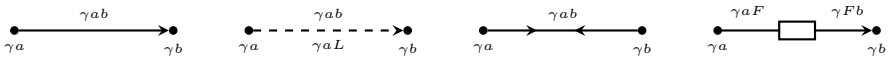


Figure 2.3: Some basic Stochastic Reo channels

A processing delay rate represents the duration that a channel takes to perform a certain activity such as transporting a data item. For instance, a LossySync has two associated variables γab and γaL for the stochastic delay rates of, respectively, successful data-flow from node a to node b, and losing the data item at node a when a read request is absent at node b. In a FIFO1, γaF means the delay for data-flow from its source node a into the buffer, and γFb means the delay for sending the data from the buffer to the sink b. Similarly, γab of a Sync (and a SyncDrain, respectively) indicates the delay for data-flow from its source node a to its sink node b (and losing data at both ends, respectively).

Arrival rates describe the time between consecutive arrivals of I/O requests at source and sink nodes of Reo channels. For instance, γa and γb in Figure 2.3 represent the associated arrival rates of write/take requests at nodes a and b. As mentioned earlier, at most one request can wait at a boundary node for acceptance. That is, if a boundary node is occupied by a pending request, then the node is blocked and consequently all further arrivals at that node are lost.

Stochastic Reo supports the same compositional framework of joins of connectors as in Reo. Most of the technical details of this *join* operation are identical to that of Reo. The nodes in Stochastic Reo have certain QoS information on them, hence joining nodes must accommodate QoS composition.

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). Thus, when joining a source with a sink node into a mixed node, their arrival rates are discarded[1].

[1]For simplicity, we assume that the activity of ideal nodes incur no delay. Any real implementation of a node, of course, induces some processing delay rate. However, such 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 even associate delay distributions with Stochastic Reo nodes and automatically translate such nodes into "Sync plus ideal node" constructs. We ignore this issue in the rest of this thesis.

The activities of a Reo connector consist of I/O request arrivals at boundary nodes, synchronization in mixed nodes, and data-flows through primitive channels. Adding time information to a connector gives rise to the causality of such activities. That is, for a given Reo connector, first I/O requests must arrive at the boundary nodes of a connector, second synchronization occurs, and finally data-flows happen. For instance, in Figure 2.4, first I/O requests arrive at a and d; second the synchronization on the mixed node b or c, selected by merger d, occurs; finally a data item is delivered from the source node a to the sink node d via the mixed node b or c.

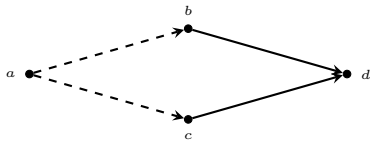


Figure 2.4: Example for the causality of a Reo connector

In order to describe the processing delay rates of a primitive channel explicitly, we name the rate by the combination of a pair of (source, sink) nodes and the buffer of the channel. For example, γab for the Sync channel and γaF for the FIFO1 channel in Figure 2.3. As mentioned in Section 2.1, a source node and a sink node act as a replicator and a non-deterministic merger, respectively, and each activity, such as replicating data to its source nodes or selecting a sink node, has its own stochastic value, the reference of which can be represented using their source and sink nodes. However, for simplicity, we do not describe the names of source and sink nodes of a replicator and a merger explicitly when the nodes are not boundary nodes. In these cases, the processing delay rates for the selection or the replication by, respectively, a merger or a replicator are not distinguishably described. Thus, we name the internal nodes of a replicator or a merger by naming after the initial name of the replicator or the merger with index. For example, merger d in Figure 2.4 has three different nodes: two source nodes and one sink node. Let the source node transmitting data from node b, the other source node, and the sink node be, respectively, d_1, d_2, and d, whereas

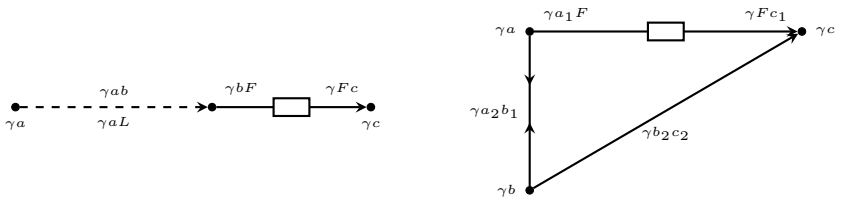


Figure 2.5: LossyFIFO1 and ordering circuit in Stochastic Reo

the first two of those distinctive names are omitted here. Then, the processing delay rates of merger d are described as $\gamma d_1 d$ and $\gamma d_2 d$ which refer to the rates for the selection of data from node b and c, respectively.

Figure 2.5 shows the LossyFIFO1 and the ordering circuit in Stochastic Reo with their stochastic values. (Compare Figure 2.2)

2.3 Semantic models for Reo

2.3.1 Constraint Automata

Constraint Automata (CA) were introduced in [12] as a formalism to capture the operational semantics of Reo, based on timed data streams, which constitute the foundation of the coalgebraic semantics of Reo [9].

We assume a finite set Σ of nodes, and denote by $Data$ a fixed, non-empty set of data that can be sent and received through these nodes via channels. CA use a symbolic representation of data assignments by data constraints, which are propositional formulas built from the atoms "$d_a \in P$", "$d_a = d_b$" and "$d_a = d$" using standard Boolean operators. Here, $a, b \in \Sigma$, d_a is a symbol for the observed data item at node a, $d \in Data$, and $P \subseteq Data$. $DC(N)$ denotes the set of data constraints can refer to the observed data items d_a at node a for $a \in N$ where $N \subseteq \Sigma$. Logical implication induces a partial order $\leq$ on DC: $g \leq g'$ iff $g \Rightarrow g'$.

A CA over the data domain $Data$ is a tuple $\mathcal{A} = (S, S_0, \Sigma, \rightarrow)$ where S is a set of states, also called configurations, $\emptyset \neq S_0 \subseteq S$ is the set of its initial states, Σ is a finite set of nodes, $\rightarrow$ is a finite subset of $\bigcup_{\emptyset \subset N \in 2^\Sigma} S \times \{N\} \times DC(N) \times S$, called the transition relation. A transition fires if it observes data items in its respective ports/nodes of the component that satisfy the data constraint of the transition, and this firing may consequently change the state of the automaton.

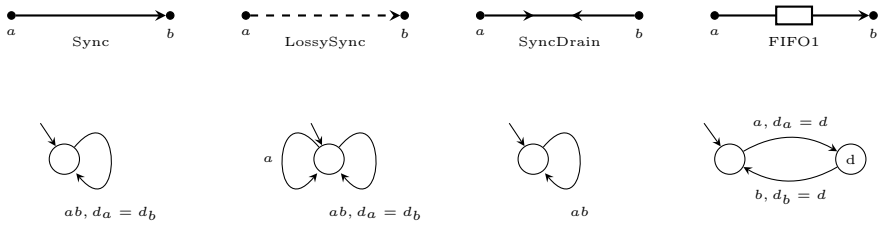


Figure 2.6: Constraint Automata for basic Reo channels of Figure 2.1

Figure 2.6 shows the CA for the primitive Reo channels in Figure 2.1. In this figure and the remainder of this thesis, the initial states are indicated with an extra incoming arrows. For simplicity, we assume the data constraints of all transitions are implicitly *true* (which simply imposes no constraints on the contents of the data-flows) and omit them to avoid clutter. In addition, we use a simplified notation for the set of nodes in

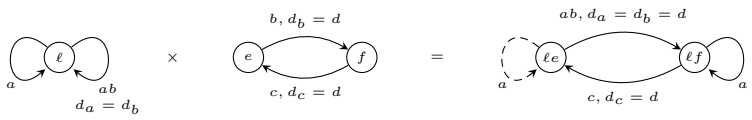
the labels of transitions by deleting the curly brackets { and } and commas between the set elements. For a full treatment of data constraints in CA, see [12].

As the counterpart for the join operation in Reo, the product of two CA $\mathcal{A}_1 = (S_1, S_{1,0}, \Sigma_1, \rightarrow_1)$ and $\mathcal{A}_2 = (S_2, S_{2,0}, \Sigma_2, \rightarrow_2)$ is defined as a constraint automaton $\mathcal{A}_1 \bowtie \mathcal{A}_2 \equiv (S_1 \times S_2, S_{1,0} \times S_{2,0}, \Sigma_1 \cup \Sigma_2, \rightarrow)$ where $\rightarrow$ is given by the following rules:

- If $s_1 \xrightarrow{N_1, g_1}_1 s_1'$, $s_2 \xrightarrow{N_2, g_2}_2 s_2'$ and $N_1 \cap \Sigma_2 = N_2 \cap \Sigma_1$, then $\langle s_1, s_2 \rangle \xrightarrow{N_1 \cup N_2, g_1 \wedge g_2} \langle s_1', s_2' \rangle$.
- If $s_1 \xrightarrow{N_1, g_1}_1 s_1'$ and $N_1 \cap \Sigma_2 = \emptyset$ then $\langle s_1, s_2 \rangle \xrightarrow{N_1, g_1} \langle s_1', s_2 \rangle$.
- If $s_2 \xrightarrow{N_2, g_2}_2 s_2'$ and $N_2 \cap \Sigma_1 = \emptyset$ then $\langle s_1, s_2 \rangle \xrightarrow{N_2, g_2} \langle s_1, s_2' \rangle$.

Context-dependency

The context-dependency of a Reo connector is not captured by CA. For example, recall the LossyFIFO1 example in Figure 2.2. The corresponding CA for the LossyFIFO1 is built by the product of a Sync channel ab and a FIFO1 channel bc as shown below. For simplicity, here and in the remainder of this chapter, the representations of the configurations are simplified by omitting commas between composed configurations and round brackets '(' and ')' surrounding the composed configurations.



The dashed transition from the source state ℓe is unintended because it implies that a data item is lost at node a even though the buffer is empty and able to take a data item from node a.

2.3.2 Intentional Automata

Intentional Automata (IA) [31, 32] are another semantic model for Reo, where the arrivals of I/O requests and the actual communication are described separately. Based on such characteristics, IA are useful to represent certain behavior that depends on the presence or absence of pending I/O requests in its environment/context. Thus, it can be used to specify context-dependent connectors [2] which CA cannot capture.

In general, a connector has a range of possible outputs for the same inputs from its environment. To model such a connector, throughout this thesis IA are considered to be non-deterministic even if the non-determinism is not explicitly mentioned.

Definition 2.3.1 (Intentional Automaton [31]). *An* Intentional Automaton *is a tuple* (Q, Σ, δ) *with a set of states (internal configurations)* Q*, a set of nodes* Σ*, and a transition relation* $\delta : Q \rightarrow \mathcal{P}(\mathcal{F} \times Q)^{\mathcal{R}}$ *where*

- $\mathcal{R} = \mathcal{P}(\Sigma)$ *is a set for the arrival of I/O requests, a so-called request-set, and*
- $\mathcal{F} = \mathcal{P}(\Sigma)$ *is a set for the actual communication, a so-called firing-set.*

This transition relation associates a function $\delta_q : \mathcal{R} \to \mathcal{P}(\mathcal{F} \times Q)$ *with every state* $q \in Q$*, defined by* $\delta_q(R) = \delta(q)(R)$*.* ■

Note that $\mathcal{P}(S)$ is the collection of all subsets of any set S, i.e., $\mathcal{P}(S) = 2^S$. A transition in an IA model (Q, Σ, δ) is represented as $q \xrightarrow{R|F} q'$ where $R,\ F \in \mathcal{P}(\Sigma)$ which is interpreted as $(F, q') \in \delta_q(R)$. Based on this definition, Figure 2.7 shows the IA for a Sync channel. For readability, here and in the remainder of this chapter, we simplify the representation of labels on transitions by omitting curly brackets for the sets of R and F and the commas between the elements in R and F.

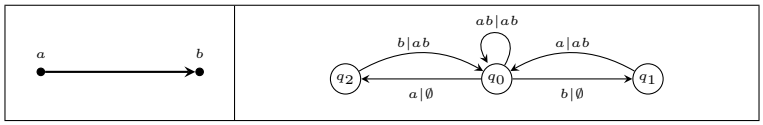


Figure 2.7: IA for a Sync channel

However, the IA only considers internal configurations of connectors. This is not enough to fully specify the behavior of Reo connectors since the behavior of a connector does not only involve its internal configuration, but also the external configuration of the system interacting with its environment. For this purpose, IA have been extended by states in $S \subseteq Q \times \mathcal{P}(\Sigma)$ where Q is the set of internal configurations of a connector and Σ is the set of nodes. Such an extension allows us to infer important invariants for the evaluation steps (transitions) of the extended IA model of a Reo connector [31, Chapter 5]:

1. a node can fire only if it either has already a pending request, or receives a request in this step;
2. when it receives a request, a node either fires the request in this step or the request becomes pending;
3. a node with a pending request, either fires it in this step or it remains pending;
4. a node has a pending request after an evaluation step only if the node receives a request and does not fire it in this step, or a request was already pending and does not fire in this step;
5. a node with a pending request is unavailable to receive requests;
6. a node that fires cannot become/remain pending.

The following formulas show these invariants formally; each formula corresponds to the invariant with the same number. For the evaluation step of the extended IA of a connector $(q, P) \xrightarrow{R|F} (q', P')$, it holds that

1. $F \subseteq R \cup P$
2. $R \subseteq F \cup P'$
3. $P \subseteq F \cup P'$
4. $P' \subseteq R \cup P$
5. $P \cap R = \emptyset$
6. $F \cap P' = \emptyset$

Here and in the remainder of this thesis, we consider the extended IA that satisfy the above invariants.

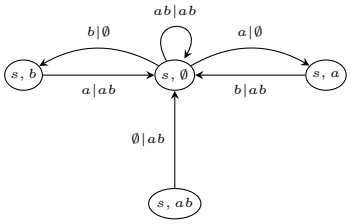
Compared to CA, the extended IA models have more states since IA consider both internal and external configurations, whereas CA only consider internal configurations. For a concise specification of the configurations of the extended IA, a listing, called an *abstract configuration table*, is used.

Definition 2.3.2 (Abstract configuration table [31]). *Given a set of internal configurations S and a set of nodes Σ, an* abstract configuration table *over S and Σ, denoted by $\theta(S, \Sigma)$, is a table such that:*

- *for each $s \in S$, there is one column labeled by s;*
- *for each $R \subseteq \Sigma$, there is one row labeled by R;*
- *at each cell of the table at the intersection of row R with column s we have a set, denoted $\theta\langle s, R\rangle$, such that $\theta\langle s, R\rangle \subseteq \mathcal{P}(\Sigma) \times (S \times \mathcal{P}(\Sigma))$, and for all $\langle F, (s', P')\rangle \in \theta\langle s, R\rangle$, we have $R = F \cup P'$ and $F \cap P' = \emptyset$.*

■

For example, Figure 2.8 shows the extended IA for a Sync channel ab and its configuration table. For readability, here and in the remainder of this chapter, we simplify the representation of the configurations by omitting brackets '()' and '{}' for, respectively, the overall configurations and the external configuration. Moreover, we delete commas between the elements in the external configuration.



	a	b
	s	
$\emptyset$	$\langle\emptyset, (s, \emptyset)\rangle$	
$\{a\}$	$\langle\emptyset, (s, \{a\})\rangle$	
$\{b\}$	$\langle\emptyset, (s, \{b\})\rangle$	
$\{a, b\}$	$\langle\{a, b\}, (s, \emptyset)\rangle$	

Figure 2.8: Extended IA for Sync ab and its configuration table θ_{Sync}

Such an abstract configuration table defines the extended IA model for a Reo connector and is, generally, more compact than its automaton model. Thus, an abstract

configuration table is used to apply other operations to its corresponding automaton model, for example, the product of the extended IA corresponding to a Reo connector is defined with abstract configuration tables (see below). The extended IA model of an abstract configuration table for a connector C is denoted by $[\![\theta_C(S,\Sigma)]\!]_R$ where S is a set of configuration and Σ is a set of nodes.

Operations

For the compositional semantics of a *join* operation in a Reo connector, the configuration tables of automata models are used. The advantage of this method, instead of using the operation of automata composition, is that it has lower computational cost, since in general, abstract configuration tables are smaller than automata models.

Definition 2.3.3 (Product of abstract configuration tables [31]). *Given two abstract configuration tables $\theta\langle S_1,\Sigma_1\rangle$ and $\theta\langle S_2,\Sigma_2\rangle$, their product abstract configuration table is*

$$\theta\langle S_1,\Sigma_1\rangle \times_T \theta\langle S_2,\Sigma_2\rangle = \theta\langle S_1\times S_2, \Sigma_1\cup\Sigma_2\rangle$$

where each cell of the table is given by: for every $R\in\mathcal{P}(\Sigma_1\cup\Sigma_2)$ and $R_i\in\mathcal{P}(\Sigma_i)$ with $i\in\{1,2\}$

$$\begin{aligned}
&\theta\langle(s_1,s_2),R\rangle = \\
&\{\ \langle F,((s_1',s_2'),P')\rangle \ \mid\ R=R_1\cup R_2,\ F=F_1\cup F_2,\ P'=P_1'\cup P_2',\\
&\qquad\qquad F_1\cap\Sigma_2=F_2\cap\Sigma_1,\ \langle F_i,(s_i',P_i')\rangle\in\theta\langle s_i,R_i\rangle,\ i=1,2\ \}\\
&\cup\{\ \langle F_1,((s_1',s_2),P')\rangle \ \mid\\
&\qquad F_1\cap\Sigma_2=\emptyset,\ R=R_1\cup R_2,\ P'=P_1'\cup R_2,\ \langle F_1,(s_1',P_1')\rangle\in\theta\langle s_1,R_1\rangle\ \}\\
&\cup\{\ \langle F_2,((s_1,s_2'),P')\rangle \ \mid\\
&\qquad F_2\cap\Sigma_1=\emptyset,\ R=R_1\cup R_2,\ P'=R_1\cup P_2',\ \langle F_2,(s_2',P_2)\rangle\in\theta\langle s_2,R_2\rangle\ \}
\end{aligned}$$

■

Note that, here and the rest of this section, $\times_T$ is used to represent the product of two abstract configuration tables, as defined in [31, Chapter 5].

The notion of equivalence $\simeq^2$ is used as a bisimilarity, defined below.

Definition 2.3.4 (Bisimulation of IA [31]). *Given two IA $\mathcal{A}_1=(Q_1,\Sigma_1,\delta_1)$ and $\mathcal{A}_2=(Q_2,\Sigma_2,\delta_2)$, a relation $\mathcal{Z}\subseteq Q_1\times Q_2$ is called a bisimulation if for $q_1\in Q_1$ and $q_2\in Q_2$, $(q_1,q_2)\in\mathcal{Z}$, then*

- $q_1 \xrightarrow{R|F}_{\delta_1} q_1'$ *implies there is a* $q_2'\in Q_2$ *such that* $q_2 \xrightarrow{R|F}_{\delta_2} q_2'$ *with* $(q_1',q_2')\in\mathcal{Z}$
- $q_2 \xrightarrow{R|F}_{\delta_2} q_2'$ *implies there is a* $q_1'\in Q_1$ *such that* $q_1 \xrightarrow{R|F}_{\delta_1} q_1'$ *with* $(q_1',q_2')\in\mathcal{Z}$

[2] In this thesis, we mention IA and Reo Automata as preliminaries. For a bisimilarity relation, the same notation $\sim$ is used for both automata models in their original literatures (IA in [31] and Reo Automata in [19]). To distinguish these two relations, in this thesis, $\simeq$ is used for the bisimilarity of IA, and $\sim$ is used for Reo Automata.

■

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

Theorem 2.3.5. *[31] Given two abstract configuration tables $\theta\langle S_1, \Sigma_1\rangle$ and $\theta\langle S_2, \Sigma_2\rangle$,*

$$[\![\theta\langle S_1, \Sigma_1\rangle]\!]_R \times_I [\![\theta\langle S_2, \Sigma_2\rangle]\!]_R \simeq [\![\theta\langle S_1, \Sigma_1\rangle \times_T \theta\langle S_2, \Sigma_2\rangle]\!]_R$$

Note that $\times_I$ is used to represent the product of the extended IA models, as defined in [31, Chapter 5]. The proof of Theorem 2.3.5 is shown in [31, Chapter 5].

A hiding operation is also defined for IA on abstract configuration tables.

Definition 2.3.6 (Hiding on abstract configuration tables [31]). *Consider an abstract configuration table $\theta\langle S, \Sigma\rangle$ and a node $h \in \Sigma$. We define*

$$\exists_T[h]\theta\langle S, \Sigma\rangle = \theta[h]\langle S, \Sigma \setminus \{h\}\rangle$$

where

$$\theta[h]\langle s, R\rangle = \begin{cases} \{\langle F \setminus \{h\}, q\rangle \mid \langle F, q\rangle \in \theta\langle s, R \cup \{h\}\rangle,\ h \in F\} & \text{if non-empty} \\ \theta\langle s, R\rangle & \text{otherwise} \end{cases}$$

■

In addition, the extended IA model context-dependent connectors. For instance, the LossyFIFO1 example mentioned above is given below with the correct semantics, where a data item is lost only if the buffer is full, i.e., a loop with $a|a$ occurs in configuration ℓf.

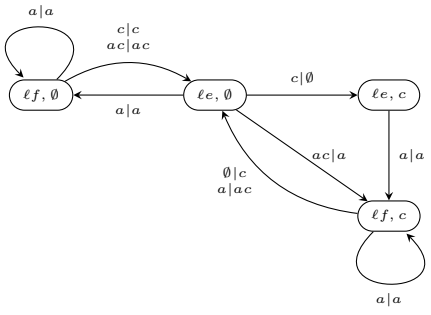


Figure 2.9: Extended IA for a LossyFIFO1 connector in Figure 2.2

2.3.3 Reo Automata

In this section, we recall Reo Automata [19], another semantic model for Reo. This model also provides a compositional operational semantics and the correct semantics for the context-dependent 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.

Compared to IA, Reo Automata provide the formal proof of their compositionality [19]. Moreover, Reo Automata are simpler and more compact, retaining the power of correctly encoding context-dependency of Reo connectors.

We recall some facts about Boolean algebras. Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ be a set of symbols that denote the names of connector nodes, $\overline{\sigma}$ be the negation of σ, and $\mathcal{B}_\Sigma$ be the free Boolean algebra generated by the 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 their representation we frequently omit $\wedge$ and write $g_1 g_2$ 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 $\alpha, \beta, \ldots$ 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 $\widehat{S} \equiv abc$.

Definition 2.3.7 (Reo automaton [19]). *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* $q \xrightarrow{g|f} q' \in \delta$*:*

(1) $g \leq \widehat{f}$ *(reactivity)*

(2) $\forall g \leq g' \leq \widehat{f} \cdot \forall \alpha \leq g' \cdot \exists q \xrightarrow{g''|f} q' \in \delta \cdot \ \alpha \leq g''$ *(uniformity)*

■

In Reo Automata, for simplicity we abstract data constraints [12] 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 an atom 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

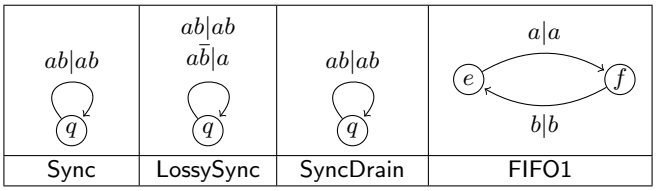


Figure 2.10: Automata for basic Reo channels of Figure 2.1

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 [19]. In compliance with these criteria, for a firing f, its guard g considers the presence of the least sufficient requests.

In Figure 2.10 we depict the Reo Automata for the basic channel types listed in Figure 2.1. Note that here and in the remainder of this thesis, 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) $g_1 \vee \ldots \vee g_k$ 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.

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 2.3.8. *[19] 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 2.3.9 (Product of Reo Automata [19]). *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:*

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

■

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 according to [19], the product result preserves the properties of Reo automata, i.e., reactivity and uniformity in Definition 2.3.7. Figure 2.11 shows an example of the product of two automata.

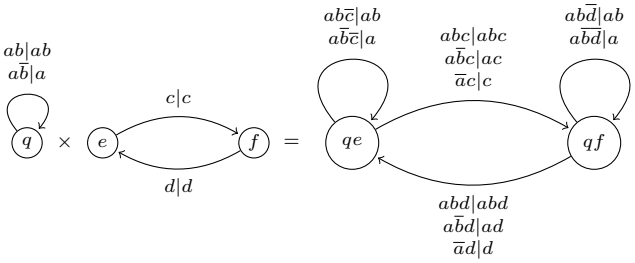


Figure 2.11: Product of LossySync and FIFO1

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), only the transitions where either both a and b or neither a nor b fire are kept in the resulting automaton, i.e., $a \in f \Leftrightarrow b \in f$ — this is what it means for a and b to synchronize. Moreover, we keep only those 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* [2], which implies that an internal node cannot store data nor actively block behavior.

Definition 2.3.10 (Synchronization [19]). *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*

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

■

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.

Synchronizing nodes b and c of the product automaton in Figure 2.11 yields the automaton depicted in Figure 2.12[3], which provides the semantics for the LossyFIFO1 example.

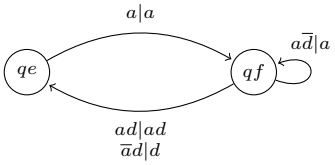


Figure 2.12: Reo Automaton for LossyFIFO1

Compositionality

Given two Reo Automata $\mathcal{A}_1$ and $\mathcal{A}_2$ over the disjoint alphabet sets 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 2.3.11. *[19] 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. $(\partial_{a,b}\mathcal{A}_1) \times \mathcal{A}_2 \sim \partial_{a,b}(\mathcal{A}_1 \times \mathcal{A}_2)$, *if* $a, b \notin \Sigma_2$ $\Sigma_1 \cap \Sigma_2 = \emptyset$.

♦

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

Definition 2.3.12 (Bisimulation [19]). *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 $\alpha \in \mathcal{B}_\Sigma$, $\alpha \leq g$, 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. ■

[3]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 circuit, becomes indistinguishable from the automaton for a shift LossyFIFO1 [12] circuit. However, by reviving data-constraints we can distinguish the automata for these two circuits.

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.

2.4 Markov Chains

Stochastic processes are used for modeling random phenomena as transition systems with probability distributions for the outgoing transitions of a state. Markov Chains (MCs) are a special case of such stochastic processes, which satisfy

1. *discrete state space* which implies that their state space is countable and
2. *Markov property* which implies that the state change from a current state depends on only the current state, not on the history, i.e., the sequence of visited states.

Such state change in MCs can be considered with or without taking into account the time instance when the change occurs. In case that the state change is independent of the time instance, MCs are said to be *homogeneous*. The time homogeneity in stochastic processes gives us the freedom for a certain event to occur at any time instance. In the other case, it is called *inhomogeneous*, which gives much flexibility for specifying system behavior.

In addition, the Markov property requires that the waiting time (i.e., sojourn time) satisfies *memoryless property*: at time instance t, the remaining time before leaving a state is independent of the time already spent in that state.

According to the time domains, MCs are categorized into two classes: Discrete-Time Markov Chains (DTMCs) and Continuous-Time Markov Chains (CTMCs). To satisfy the memoryless property in respective time domains, the geometric distribution and the exponential distributions are necessary for DTMC and CTMC, respectively.

With these conditions, MCs can be seen as relatively simple stochastic processes. Nonetheless, MCs are frequently used to model various probabilistic systems. Moreover, its simplicity yields efficient algorithms [85] for numerical analysis.

Here and in the remainder of this thesis, we deal only with homogeneous MCs, especially homogeneous CTMCs, even though we do not mention the homogeneity of MCs explicitly.

Continuous-Time Markov Chains

A Continuous-Time Markov Chain (CTMC) is a discrete-state Markov process with continuous time domain, $\{X(t)|t \geq 0\}$, which can be used to model and analyze random system behavior. $X(t) \in S$ denotes the state in a given state space S at time t. Let $\mathbf{P}\{X(t) = i\}$ be the probability that the process is in state i at time t. The stochastic process $X(t)$ is a homogeneous CTMC if, for ordered times $t_0 < \cdots <$

$t_n < (t_n + \Delta t)$, the conditional probability of staying in any state j satisfies:

$$\mathbf{P}\{X(t_n + \Delta t) = j \mid X(t_n) = i_n, X(t_{n-1}) = i_{n-1}, \cdots, X(t_0) = i_0\} = \mathbf{P}\{X(t_n + \Delta t) = j \mid X(t_n) = i_n\}.$$

Briefly, the probability that the process is in *future* state j depends on only the *current* state i_n, not the past states.

The sojourn time in any state of a CTMC model must be *exponentially distributed* since the exponential distributions are the only class that satisfies the memoryless property in continuous time domain. Below we list the properties of the exponential distributions that are relevant to our work.

- An exponential distribution $P\{delay \leq t\} = 1 - e^{-\lambda t}$ is characterized by a positive real value λ, the so-called *rate* of the distribution. Its mean duration is $1/\lambda$ time units.
- While satisfying the memoryless property, the remaining delay after some time t_0 has elapsed is also exponentially distributed:

$$P\{delay \leq t + t_0 \mid delay > t_0\} \;=\; P\{delay \leq t\}$$

- Exponential distributions are closed under minimum which is the sum of the rates:

$$P\{min(delay_1, delay_2)\} = 1 - e^{-(\lambda_1 + \lambda_2)t}$$

  where λ_1 and λ_2 are the rates of the distributions $delay_1$ and $delay_2$, respectively.
- The probability that $delay_1$ with the rate λ_1 is smaller than $delay_2$ with the rate λ_2 is

$$P\{delay_1 < delay_2\} = \frac{\lambda_1}{\lambda_1 + \lambda_2}$$

- In the continuous-time domain, the probability that two delays elapse at the same time is *zero*.

Such properties of exponential distributions state that the probability to stay in a state decreases as time elapses, i.e., a transition emanating from a certain state will be triggered eventually. When a certain state has more than one possible leaving transitions, the transition will be triggered proportional to its rate.

2.5 Interactive Markov Chains

Interactive Markov Chains (IMCs) [43] are a stochastic model to specify reactive systems. In IMCs, timing information and actions are represented separately. Timing information is described by *Markovian transitions*, and actions are described by *interactive transitions*. Roughly speaking, IMCs are a combination of Labeled Transition Systems (LTSs) and CTMCs.

An IMC 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 an 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 2.5.1. ***(Product of IMCs [43])*** *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* $\mathcal{I}_2$ *with respect to a set* A *of actions 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:*

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

■

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

Compared to CTMCs, IMCs 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) [40]. CADP verifies the functional correctness of the specification of system behavior and also minimizes IMCs efficiently [39]. Moreover, IMCs can be used in various other applications, such as Dynamic Fault Trees (DFTs), Architectural Analysis and Design Language (AADL), and so on [44].

2.6 Related work

2.6.1 Other coordination languages

Orc [64] is a theory of *orchestration* of *sites* which are considered as basic services distributed over a network. In Orc, each connection between sites takes place highly asynchronously and performs only once. While performing the connection, the orchestrator (Orc expression) initiates its connections dynamically. Such dynamics enables

to deal with failures in sites well. Compared to Orc, the connection in Reo is static, as it is based on the assumption that components communicate continuously. Recently, the research on the dynamic reconfiguration of Reo connectors has been initiated in [53]. In addition, Reo is highly synchronous, thus, it can specify the propagation of synchrony and mutual exclusion through Reo connectors. More detailed comparison is provided in [78].

Linda [41] is the first coordination language that describes the communication between different processes by exchanging data. In Linda, data objects are referred to as *tuples*, and communicating data takes place in a *shared tuple-space*. Communication actions in the shared tuple-space can occur atomically, and interactions occur in an interleaved way. That is, Linda does not handle the propagation of synchrony which is supported by Reo.

BIP [15] (an acronym of *Behavior*, *Interaction*, and *Priority*) is a methodology for modeling heterogeneous real-time components and their composition. The composition in BIP happens in three different layers, viz. that of behavior, interaction, and priority. The lower layer, an atomic component, describes its behavior; the intermediate layer specifies possible interactions between atomic components; the upper layer presents the priority relation to select amongst possible interactions. Compared to Reo, the priority relation in BIP is the main difference. This priority is used to explicitly consider the scheduling the connection between components. Whereas, in Reo, the scheduling/selection aspects is decided non-deterministically randomly by each merger.

These coordination languages have been proposed to model the composition of distributed system over a network. Each of them has its own features to specify some situations in the composition. However, Reo is the only coordination language that supports global synchronization (the propagation of synchrony), mutual exclusion through connectors, and the combination of synchrony and asynchrony.

2.6.2 Continuous-Time Constraint Automata

Continuous-Time Constraint Automata (CCA) [13] are a stochastic extension of CA that 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:

- *interactive transitions* $p \xrightarrow{N,g} q$ as an ordinary transition in CA and
  - hidden transitions if $N = \emptyset$
  - visible transitions otherwise
- *Markovian transitions* $p \xrightarrow{\lambda} q$ where $\lambda \in \mathbb{R}^+$, called the rates of distributions.

In CCA, data-flows in connectors are represented by interactive transitions since the synchrony and the asynchrony of data-flows can be captured by the ordinary CA transitions. Processing data in components is represented by Markovian transitions

since processing data in each component is independent of the processing in the others, and each processing occurs concurrently.

CCA can be used to specify the interaction of components and connectors that connect the components, as well as to reason about some QoS aspects of the connectors such as the average processing time of I/O requests in a certain component. Moreover, CCA support both *non-deterministic choice* and *probabilistic choice*. When a current state has one or more outgoing hidden (invisible interactive) transitions, one of the outgoing interactive transitions from the state is chosen non-deterministically. When there is no outgoing hidden transitions from a current state, then one outgoing Markovian transition from the current state is chosen probabilistically and fires.

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 boundary nodes of a connector are not considered as stochastic processes. Reasoning about the end-to-end QoS of systems requires incorporation of this external behavior. In addition, CCA do not capture the context-dependency of a Reo connector since interactive transitions in CCA merely follow CA transitions that do not formalize the context-dependency of Reo connectors. Compared to such CCA, the specification models in this thesis, Quantitative Intentional Automata and Stochastic Reo Automata (See Chapter 3 and Chapter 4, respectively), not only specify the end-to-end QoS of a Reo connector, but also capture context-dependent behavior.

2.6.3 Stochastic Process Algebra

Process Algebra (PA) [63, 49, 11] is a compositional specification formalism of algebraic nature for concurrent systems. It describes interactions, communications, and synchronizations between processes in a system. PA provides a compositional approach, where a system is modeled by a collection of subsystems called *agents* that execute atomic *actions*. These actions describe communications between agents and sequential behavior that may run concurrently.

Stochastic Process Algebra (SPA) [45] is a stochastic extension of PA, which integrates Process Algebra theory and stochastic processes. SPA is described by three parts: *actions* that model the system activities, *algebraic operators* that compose the subsystem specifications, and *synchronization discipline*. An action in SPA consists of an action type a and its exponential rate λ, i.e., $\langle a.\lambda \rangle$. Several algebraic operators are shown below:

name	expression	denotation
prefix	$\langle a.\lambda \rangle.E$	After action a with a rate λ, the agent becomes E.
abstraction	E/L	The actions in L are hidden.
relabeling	$E[a_1/a_0, \dots]$	The label a_1 is renamed a_0.
choice	$E_1 + E_2$	The agent behaves either E_1 or E_2.
parallel composition	$E_1 \| \| E_2$	The agents E_1 and E_2 proceed in parallel.

There are several synchronized solution disciplines for the rate of the synchronized (shared) actions, and different solutions yield various SPA formalisms such as Performance Evaluation Process Algebra (PEPA) [46, 47], Extended Markovian Process Algebra (EMPA) [17, 16]. In PEPA, it is assumed that each agent has a *bounded capacity* to carry out activities of any particular type, determined by the rate that is the sum of the rates of each action enabled in that agent. That is, an agent cannot exceed its boundary capacity, thus the rate of a synchronized action is the minimum of the rates of the agents involved. In EMPA, it is assumed that in a synchronization, at most one participant in the synchronization has an explicit representation for the rate of the resulting (synchronized) action.

SPA has the following benefits:

- to support a compositional specification, i.e., given a complicated system, modeling its sub-systems and the interaction between the sub-systems
- clear structure and semantics
- model reuse and maintaining a library of models

The limitation of SPA is the lack of expressiveness with respect to its timing distribution: only negative exponential distributions can be used. To make the SPA more general, some work has been carried out by associating general distributions with the actions of a model [52].

The operational semantic model of SPA is defined by means of a labeled transition model. Because of interleaving, the semantic model of SPA suffers from the state explosion problem. Research has been carried out to mitigate this problem in [26, 62, 42].

SPA describes '*how*' each process behaves, whereas, (Stochastic) Reo directly describes '*what*' communication protocols connect and '*how*' they 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.

2.6.4 Stochastic Petri Nets

Petri Nets (PNs) [74, 79] are graphical and mathematical models that describe system behavior with concurrency, asynchrony, and synchrony. As a graphical model, PN is similar to flow charts, block diagrams, and networks. As a mathematical model, it is used to set up state equations, algebraic equations, and so on.

Stochastic Petri Nets (SPNs) [65, 87, 60] are a stochastic extension of PN, by associating an exponentially distributed firing time with each transition in a PN. The reachability set of an SPN model is identical to the one of its underlying PN model, thus, the structural properties obtained for PN, such as liveness, boundness, conservativeness, repetitiveness, consistency, and controllability, are still valid for SPNs.

The countability of the markings and the memoryless property of exponential distributions allow an isomorphism between SPN models and CTMC models. Thus, the CTMC model corresponding to an SPN is obtained by constructing the reachability graph of the SPN model and by labeling its arcs with the firing rates of each transition that changes markings.

Such an SPN is a useful tool for the analysis of computer systems since it allows the system operations to be described precisely by means of a graph that translates into a Markovian model useful for obtaining performance estimates. Due to its graphical representation, an SPN can be easily understood. In addition, the derivation of the MC model and its solution can be made automatic, and transparent to the users.

However, as for state-based models, they in general suffer from the state-explosion problem, the graphical representation of an SPN causes fast increasing complexity in their numerical solution as the system size increases. Thus, a large SPN model is often used for simulation. In addition, a PN essentially deals with asynchronous events and does not propagate the synchrony of events, thus, its compositionality is not clear in general [4].

The topology of connectors in (Stochastic) Reo is inherently dynamic, and it accommodates mobility as described in [56]. Moreover, (Stochastic) Reo supports a liberal notion of channels, which allows to express synchrony and asynchrony. Reo is more general than data-flow models and PNs [4], which can be viewed as specialized channel-based models that incorporate certain built-in primitive coordination constructs.

2.6.5 Stochastic Automata Networks

A Stochastic Automata Network (SAN) [86, 37] specifies a system consisting of a number of individual Stochastic Automata. Each Stochastic Automaton runs independently or synchronously with the others. The rates on the transitions of a SAN are either constants or functions:

- constants, i.e., non-negative real numbers
- functions from the global state space to non-negative real numbers

Normally an automaton makes use of both kinds of transitions for modeling.

In general, events in each automaton are categorized into two different types of independent and synchronized events. In the case of independent events in a SAN, the effect of the constants or functional transitions is local, thus, all the information relevant to the transitions in a Stochastic Automaton is handled in that automaton with the assumption that the automaton has a knowledge of the global state space.

In the case of synchronized events, the effect of the transitions is global by altering the state of a number of Stochastic Automata.

SAN is used for performance modeling related to parallel distributed systems. Parallel and distributed systems can be seen as collections of components that interact with each other. Thus, each component corresponds to an individual Stochastic Automaton and the overall system corresponds to a collection of such automata.

However, SAN is a state-base model, where potentially the state explosion problem arises. To mitigate this problem, techniques to minimize the number of states have been suggested. For this purpose, in SAN, it is possible to make use of symmetries as well as lumping and various superpositioning of automata [27, 82]. In addition, SAN does not store nor generate the (global) state transition matrix. Instead of that, it is represented by a number of small matrices relevant for each Stochastic Automaton. Thus, a SAN approach has minimal memory requirements.

Compared to (Stochastic) Reo, the interactions in SAN are rather limited for patterns like synchronizing events. The representation of synchronized events requires an appropriate transition label that consists of a transition probability and an alternative probability. A transition probability must be unique for the synchronized events; an alternative probability is different for each individual automaton involved in the synchronized event [75].