A Relational Model for Probabilistic Connectors Based on Timed Data Distribution Streams

Meng Sun(✉) and Xiyue Zhang

Department of Informatics and LMAM, School of Mathematical Sciences, Peking University, Beijing, China
{sunm,zhangxiyue}@pku.edu.cn

Abstract. Connectors have shown their great potential for coordination of concurrent activities encapsulated as components and services in large-scale distributed applications. In this paper, we develop a formal model for a probabilistic extension of the channel-based coordination language Reo. The model formalizes connectors with probabilistic behavior as relations on *Timed Data Distribution Streams* (TDDSs), which specifies properties of primitive channels and complex connectors with probabilistic behavior properly. Furthermore, the implementation of this probabilistic model has been developed in Coq, which serves to demonstrate how the model can be used to prove probabilistic connectors' properties.

Keywords: Coordination · Probabilistic connector
Timed data distribution streams · Coq

1 Introduction

Coordination models that formalize the interaction among different components play a key role in the development of large-scale distributed applications, which are typically heterogeneous and geographically distributed over the internet. Such coordination models usually provide a notion of *connectors* that interconnect the components and organize the mutual interactions and communications among them in a distributed environment, where complex connectors can be compositionally constructed out of simpler ones. As an example, Reo [2,8] offers a powerful gluing mechanism for the implementation of such coordinating connectors. Primitive connectors called *channels* in Reo, such as synchronous channels, FIFO channels and timer channels, can be composed to build circuit-like connectors which serve as the glue code to exogenously coordinate the behavior of components in distributed applications.

Investigating probabilistic behavior of connectors precisely is a necessary task for developing trustworthy applications. In this paper we focus on the probabilistic aspects of Reo connectors, and provide a formal model for connectors built out of channels that might behave nondeterministically and probabilistically, such

D. N. Jansen and P. Prabhakar (Eds.): FORMATS 2018, LNCS 11022, pp. 125–141, 2018.
https://doi.org/10.1007/978-3-030-00151-3_8

as unreliable FIFO channels that may loose certain data items written to the buffer, or synchronous channels that may corrupt written data with some small probability. In this model, the behavior of channels (and connectors) are specified as relations of observations on the channel ends (and sink/source nodes of connectors) given by timed data distribution streams. And we also show how the model of probabilistic channels can be used in the construction of more complex connectors with probabilistic behavior. Furthermore, the model for probabilistic channels/connectors has been encoded in Coq [18] which forms an extension to our previous work on modeling and verifying connectors in Coq [13,20], and properties of such probabilistic connectors can be formally proved using Coq.

This is in fact not the first investigation on probabilistic connectors. An operational semantics for probabilistic Reo in terms of probabilistic constraint automata (PCA) has been developed by Baier in [7]. Later the Quantitative Intentional Automata (QIA) model was proposed in [5] to capture the operational semantics of connectors with stochastic behavior. The QIA model correctly captures context dependency, but it is not compositional and suffers from state explosion heavily even for simple connectors. Another model called Stochastic Timed Automata for Reo (STA_r) was developed in [15] to support both stochastic and real-time behavior of connectors in Reo. In [16], Interactive Markov Chains are adopted as a compositional semantic model for stochastic Reo connectors. The Priced Probabilistic Timed Constraint Automata model [12] enables users to reason about both probabilistic and timed behavior, as well as resource consumption. Although some of such state-based models scale up quite well, state explosion is an inherent problem in these formalisms and not avoidable by the probabilistic extension. Furthermore, modeling unbounded primitives or even bounded primitives with unbounded data domains is impossible with finite automata models, and infinite or finite but large data domains usually also cause an explosion of state space in such state-based models which becomes seriously problematic for verification.

As shown in [13,20], specifying connectors as relations on its sink and source nodes makes it possible to verify connector properties by using theorem proving techniques and we do not have to face the state space explosion problem. Properties of a complex connector can be decomposed into some subgoals which can be proved separately in theorem provers like Coq, where relations on the nodes are specified by predicates [20]. Furthermore, comparing with other works on (both deterministic and probabilistic) Reo semantics [14], our framework defines two ternary channels *replicator* and *merger*, which makes different types of connector composition operators reduced to one single flow-through composition, and thus the composition of connectors can be interpreted more explicitly than other approaches, such as TDS in the coalgebraic semantics [6]. And by separating input and output explicitly in this model, the behavior of a connector becomes easier to be described and further composed.

The paper is structured as follows. After this general introduction, we briefly summarize the coordination language Reo in Sect. 2. Section 3 presents the model of observations on the nodes of connectors with probabilistic behavior as timed data distribution streams. Section 4 specifies the model for basic (untimed and timed) Reo channels, as well as channels with probabilistic behavior, and summarizes the composing operations to build connectors from channels. In Sect. 5, we discuss the implementation of the model in Coq, and show how to prove properties of connectors. Finally, Sect. 6 concludes with some further research directions.

2 A Reo Primer

In this section, we briefly review some basic concepts in the coordination language Reo. Reo [2] is a channel-based exogenous coordination language wherein complex coordinators, called *connectors*, are compositionally constructed from simpler ones. We summarize only the main concepts in Reo here. Further details can be found in [2,8].

Fig. 1. Some basic channels in Reo

A Reo connector usually consists of a network of primitive connectors, called *channels*. A connector provides the protocol that controls and organizes the communication and cooperation among different components. Each channel has two *channel ends*. There are two types of channel ends: *source* and *sink*. A source channel end accepts data into its channel, and a sink channel end dispenses data out of its channel. It is possible for the ends of a channel to be both sinks or both sources. Figure 1 shows the graphical representation of some basic channel types in Reo whose composition allows for expressing a rich set of coordination patterns [2,3].

A *synchronous channel* has a source and a sink end. It accepts a data item through its source end iff it can simultaneously dispense the data item through its sink end. A *lossy synchronous channel* is similar to a synchronous channel except that it always accepts all data items through its source end. The data item is transferred if it is possible to be dispensed through the sink end immediately, otherwise the data item is lost. A *FIFO1 channel* represents an asynchronous channel with one buffer cell which is empty initially (this is the case in Fig. 1). If a data element d is written through the source end, it is kept in the buffer of the FIFO1 channel until being taken out through the sink end. *Synchronous drain* has two source ends and no sink end. A synchronous drain can accept a data item through one of its ends iff a data item is also available for it to simultaneously accept through the other end as well, and both data items accepted by the channel are lost. A *t-timer* channel accepts any data item at its source end and produces a *timeout* signal after a delay of t time units on its sink end.

More exotic channels permitted in Reo are omitted here and can be found in [2,3,17]. Moreover, the set of channel types is not fixed in Reo, and new ones can be defined freely by users according to their own interaction policies, like the probabilistic and stochastic extensions defined in [7,9,15].

Complex connectors are constructed by composing simpler ones via the *join* and *hiding* operations. Channels are joined together in nodes. A node consists of a set of channel ends. The set of channel ends coincident on a node A is disjointly partitioned into the sets of source and sink channel ends. Nodes are categorized into *source*, *sink* and *mixed nodes* as shown in Fig. 2, depending on whether all channel ends that coincide on a node are source ends, sink ends or a combination of the two. The hiding operation is used to hide the internal topology of a connector. The hidden nodes can no longer be accessed or observed from outside. The behavior of a complex connector is formalized by means of the data-flow at its sink and source nodes.

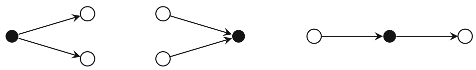

Fig. 2. Three types of nodes

A component can write data items to a source node that it is connected to. The write operation succeeds only if all (source) channel ends coincident on the node accept the data item, in which case the data item is transparently written to every source end coincident on the node. A source node, thus, acts as a replicator. A component can obtain data items, by an input operation, from a sink node that it is connected to. A take operation succeeds only if at least one of the (sink) channel ends coincident on the node offers a suitable data item. A sink node, thus, acts as a merger. A mixed node takes a suitable data item offered by one of its coincident sink channel ends and replicates it into all of its coincident source channel ends.

3 Observations as Timed Data Distribution Streams

Let D be an arbitrary finite set, the elements of which are called data elements. It will be concrete when a specific application domain is provided. We use the symbol $\perp \in D$ to denote a corrupted data item. A *data distribution* is a total function that maps D to the closed interval of reals $[0, 1]$. We define

$$\mathbf{PROB} =_{df} D \to [0, 1]$$

where for any member p of **PROB** the total sum of probabilities must not exceed 1: $\sum_{d \in D} p(d) \leq 1$. For any $X \subseteq D$, $p(X) = \sum_{d \in X} p(d)$. We use $\mathbf{0}$ to denote the zero distribution $\lambda d \bullet 0$ and define

$$p_1 \leq p_2 =_{df} \forall d \in D \bullet (p_1(d) \leq p_2(d))$$

For any $p \in \mathbf{PROB}$, we have $\mathbf{0} \leq p$. And for any $d \in D$, we have a corresponding point distribution:

$$\eta_d =_{df} \lambda x : D \bullet (1 \lhd x = d \rhd 0)$$

where the conditional expression $P \lhd b \rhd Q$ equals to P if the condition b is satisfied and Q otherwise.

The set DDS of data distribution streams is defined as $DDS = \mathbf{PROB}^\omega$, i.e., the set of all sequences $\alpha = (\alpha(0), \alpha(1), \alpha(2), \cdots)$ over $\mathbf{PROB}$ where each $\alpha(i)$ is a data distribution.

Let $\mathbb{R}_+$ be the set of non-negative real numbers, which in the present context can be used to represent time moments. Let $\mathbb{R}_+^\omega$ be the set of infinite sequences $a = (a(0), a(1), a(2), \cdots)$ over $\mathbb{R}_+$, and for all a, b in $\mathbb{R}_+^\omega$,

$$\begin{array}{lll} a < b & \text{iff} & \forall n \geq 0, a(n) < b(n) \\ a \leq b & \text{iff} & \forall n \geq 0, a(n) \leq b(n) \end{array}$$

For a sequence $a = (a(0), a(1), a(2), \cdots) \in \mathbb{R}_+^\omega$, and $t \in \mathbb{R}_+$, $a[+t]$ is a sequence defined as follows:

$$a[+t] = (a(0) + t, a(1) + t, a(2) + t, \cdots)$$

Furthermore, the element $a(n)$ in a sequence $a = (a(0), a(1), a(2), \cdots)$ can also be expressed in terms of derivatives $a(n) = a^{(n)}(0)$, where $a^{(n)}$ is defined by

$$a^{(0)} = a,\ a^{(1)} = (a(1), a(2), \cdots),\ a^{(k+1)} = (a^{(k)})^{(1)}$$

and sometimes we use a' instead of $a^{(1)}$ for simplicity.

The set TS of time streams is defined as

$$TS = \{a \in \mathbb{R}_+^\omega \mid (\forall n \geq 0. a(n) < a(n+1)) \wedge (\forall t \in \mathbb{R}_+. \exists k \in \mathbb{N}. a(k) > t)\}$$

Thus, a time stream $a \in TS$ consists of increasing and diverging time moments: $a(0) < a(1) < a(2) < \cdots$ and $\lim_{n \to +\infty} a(n) = +\infty$.

To specify inputs and outputs on connectors explicitly, for a connector $\mathbf{R}$, we use the mappings

$$\begin{array}{l} in_{\mathbf{R}} : \mathcal{N}_{in} \to TDDS \\ out_{\mathbf{R}} : \mathcal{N}_{out} \to TDDS \end{array}$$

to denote the observations on its source nodes and sink nodes, respectively. Here $\mathcal{N}_{in}$ and $\mathcal{N}_{out}$ are the sets of source and sink node names of $\mathbf{R}$, respectively. For every node N in a connector $\mathbf{R}$, the corresponding observation on N is specified by a *timed data distribution stream*, and $TDDS$ is the set of *timed data distribution streams* defined as $TDDS \subseteq DDS \times TS$, which is the set of pairs $\langle \alpha, a \rangle$ consisting of a data distribution stream α and a time stream a. Similar to the timed data sequence model used in [17], timed data distribution streams can be alternatively and equivalently defined as (a subset of) $(\mathbf{PROB} \times \mathbb{R}_+)^\omega$ because of the existence of the isomorphism

$$\langle \alpha, a \rangle \mapsto (\langle \alpha(0), a(0) \rangle, \langle \alpha(1), a(1) \rangle, \langle \alpha(2), a(2) \rangle, \cdots)$$

The occurrence of a data transfer at some node N of a connector is modeled by an element in the timed data distribution stream for that node, i.e., a pair of a data distribution $\alpha(i)$ and a time moment $a(i)$ when the data item is observed.

4 Relations on Timed Data Distribution Streams for Connectors

In this section we provide an overview on how channels and connectors can be formally modeled by relations of timed data distribution streams observed on the channel ends and sink/source nodes. We first see how primitive channels in Reo are specified by such relations, and then study the model of probabilistic channels. Finally we show how composite connectors can be constructed from simpler ones structurally.

We use $\mathcal{WD}$ as a predicate for well-defined TDDS types. In other words, we define the behavior only for valid streams expressed via the predicate $\mathcal{WD}$. Then, every connector $\mathbf{R}$ can be represented as follows:

$$\begin{aligned} \mathbf{con}: &\quad \mathbf{R}(in : in_{\mathbf{R}}; out : out_{\mathbf{R}}) \\ \mathbf{in}: &\quad P(in_{\mathbf{R}}) \\ \mathbf{out}: &\quad Q(in_{\mathbf{R}}, out_{\mathbf{R}}) \end{aligned}$$

where $\mathbf{R}$ is the name of the connector, $P(in_{\mathbf{R}})$ is the condition that should be satisfied by inputs $in_{\mathbf{R}}$ on the source nodes of $\mathbf{R}$, and $Q(in_{\mathbf{R}}, out_{\mathbf{R}})$ is the condition that should be satisfied by outputs $out_{\mathbf{R}}$ on the sink nodes of $\mathbf{R}$.

Furthermore, to capture the probabilistic behavior of connectors, we use $P_\tau \oplus Q$ to indicate that the probability for $P_\tau \oplus Q$ to be equal to P is τ, and the probability for $P_\tau \oplus Q$ to be equal to Q is $1 - \tau$. And we use $P_1 \,@\tau_1 \,|\, P_2 \,@\tau_2 \,|\, \cdots \,|\, P_n \,@\tau_n$ or

$$\begin{cases} P_1 & @\tau_1 \\ P_2 & @\tau_2 \\ \cdots & \\ P_n & @\tau_n \end{cases}$$

to represent the probabilistic choice over multiple alternatives, in which the probabilities are enumerated and sum to no more than 1: $\sum_{1\leq i\leq n} \tau_i \leq 1$.

4.1 Primitive Reo Channels

We now start by presenting a few examples of basic channels in Reo and their corresponding models in the probabilistic setting.

The simplest form of an asynchronous channel is a FIFO channel with one buffer cell, which is denoted as **FIFO1**. A **FIFO1** channel with source end A

and sink end B is graphically represented by $A\!-\!\!\boxed{}\!\!\rightarrow B$. The corresponding model is given as follows:

$$\begin{aligned}
\mathbf{con}:&\quad \mathbf{FIFO1}(\mathit{in}:(A\mapsto\langle\alpha,a\rangle);\mathit{out}:(B\mapsto\langle\beta,b\rangle))\\
\mathbf{in}:&\quad \mathcal{WD}\langle\alpha,a\rangle\\
\mathbf{out}:&\quad \mathcal{WD}\langle\beta,b\rangle\wedge\beta=\alpha\wedge a<b<a'
\end{aligned}$$

For a **FIFO1** channel, when the buffer is not filled, the input is accepted without immediately outputting it. The accepted data item is kept in the internal FIFO buffer of the channel. The next input can happen only after an output occurs. Note that the probabilistic distribution of every output data value over D is exactly the same as the distribution on the corresponding input, i.e., $\beta = \alpha$. Furthermore, we use $a < b < a'$ to represent the relation between the time moments for outputs and their corresponding (and next) inputs.

For the **FIFO1** channel $A\!-\!\!\boxed{e}\!\!\rightarrow B$ where the buffer contains a data element e initially, the communication can be initiated only if the data element e can be taken through the sink end. So the first data distribution that happens on the sink end is exactly η_e, and the following ones are the same as those observed on the source end. In this case, we denote the channel by $\mathbf{FIFO1}[e]$ as follows[1]:

$$\begin{aligned}
\mathbf{con}:&\quad \mathbf{FIFO1}[e](\mathit{in}:(A\mapsto\langle\alpha,a\rangle);\mathit{out}:(B\mapsto\langle\beta,b\rangle))\\
\mathbf{in}:&\quad \mathcal{WD}\langle\alpha,a\rangle\\
\mathbf{out}:&\quad \mathcal{WD}\langle\beta,b\rangle\wedge\beta=(\eta_e)^\frown\alpha\wedge b<a<b'
\end{aligned}$$

A synchronous channel transfers the data without any delay in time. So it behaves just like the identity function. The pair of I/O operations on its two ends can succeed only simultaneously. A synchronous channel with source end A and sink end B is graphically represented as $A \longrightarrow B$ and formally specified as follows:

$$\begin{aligned}
\mathbf{con}:&\quad \mathbf{Sync}(\mathit{in}:(A\mapsto\langle\alpha,a\rangle);\mathit{out}:(B\mapsto\langle\beta,b\rangle))\\
\mathbf{in}:&\quad \mathcal{WD}\langle\alpha,a\rangle\\
\mathbf{out}:&\quad \mathcal{WD}\langle\beta,b\rangle\wedge\beta=\alpha\wedge b=a
\end{aligned}$$

A lossy synchronous channel (graphically depicted as $A\!-\;\rightarrow B$) is similar to a normal synchronous channel, except that it always accepts all data items through its source end. If it is possible for it to simultaneously dispense the data item through its sink end, the channel transfers the data item; otherwise the data item is lost.

$$\begin{aligned}
\mathbf{con}:&\quad \mathbf{LossySync}(\mathit{in}:(A\mapsto\langle\alpha,a\rangle);\mathit{out}:(B\mapsto\langle\beta,b\rangle))\\
\mathbf{in}:&\quad \mathcal{WD}\langle\alpha,a\rangle\\
\mathbf{out}:&\quad \mathcal{WD}\langle\beta,b\rangle\wedge L(\langle\alpha,a\rangle,\langle\beta,b\rangle)
\end{aligned}$$

[1] Here $^\frown$ is the concatenation operator on sequences. The concatenation of two sequences produces a new sequence that starts with the first sequence followed by the second sequence.

where

$$
\begin{aligned}
& L(\langle \alpha, a\rangle, \langle \beta, b\rangle) \\
\equiv\ & (\beta = (\ ) \wedge b = (\ )) \vee (a(0) \leq b(0) \wedge \\
& (L(\langle \alpha', a'\rangle, \langle \beta', b'\rangle) \wedge \alpha(0) = \beta(0)) \lhd a(0) = b(0) \rhd L(\langle \alpha', a'\rangle, \langle \beta, b\rangle))
\end{aligned}
$$

The synchronous drain $A \rightarrowtail\!\!\leftarrowtail B$ is an exotic Reo channel that has two source ends A and B. Because a drain has no sink end, no data value can ever be obtained from this channel. Thus, all data accepted by this channel are lost. A synchronous drain can only accept two data items through both of its ends simultaneously.

$$
\begin{aligned}
\textbf{con}: &\quad \textbf{SyncDrain}(in : (A \mapsto \langle \alpha, a\rangle, B \mapsto \langle \beta, b\rangle); out : (\ )) \\
\textbf{in}: &\quad \mathcal{WD}\langle \alpha, a\rangle \wedge \mathcal{WD}\langle \beta, b\rangle \wedge a = b \\
\textbf{out}: &\quad \textbf{true}
\end{aligned}
$$

A filter channel $A \dashv\!\{p\}\!\!\rightarrowtail B$ specifies a filter pattern p which is a set of data values. It transfers only those data items that are matched with the pattern p and loses the rest. A write operation on the source end succeeds only if either the data item to be written does not match the pattern p or the data item matches the pattern p and it can be taken synchronously via the sink end of the channel.

$$
\begin{aligned}
\textbf{con}: &\quad \textbf{Filter}[p](in : (A \mapsto \langle \alpha, a\rangle); out : (B \mapsto \langle \beta, b\rangle)) \\
\textbf{in}: &\quad \mathcal{WD}\langle \alpha, a\rangle \\
\textbf{out}: &\quad \mathcal{WD}\langle \beta, b\rangle \wedge F(\langle \alpha, a\rangle, \langle \beta, b\rangle)
\end{aligned}
$$

where

$$
\begin{aligned}
& F(\langle \alpha, a\rangle, \langle \beta, b\rangle) \\
\equiv & \begin{cases}
\beta = (\ ) \wedge b = (\ ) & \text{if } \alpha = (\ ) \wedge a = (\ ) \\
\beta(0) = \alpha(0) \wedge b(0) = a(0) \wedge F(\langle \alpha', a'\rangle, \langle \beta', b'\rangle) & \text{if } \alpha(0) \in p \\
F(\langle \alpha', a'\rangle, \langle \beta, b\rangle) & \text{if } \alpha(0) \notin p
\end{cases}
\end{aligned}
$$

The source end of a t-timer $A \overset{t}{\longrightarrow\!\circ\!\longrightarrow} B$ channel accepts any input value d and returns on its sink end B a *timeout* signal after a delay of t time units, where t is provided as a parameter of the channel.

$$
\begin{aligned}
\textbf{con}: &\quad \textbf{Timer}[t](in : (A \mapsto \langle \alpha, a\rangle); out : (B \mapsto \langle \beta, b\rangle)) \\
\textbf{in}: &\quad \mathcal{WD}\langle \alpha, a\rangle \wedge a[+t] \leq a' \\
\textbf{out}: &\quad \mathcal{WD}\langle \beta, b\rangle \wedge \beta \in \{\eta_{timeout}\}^{\omega} \wedge b = a[+t]
\end{aligned}
$$

4.2 Probabilistic Channels

A family of channels with probabilistic behavior are specified in the following.

A *faulty FIFO1 channel* $A \cdot\overset{\tau}{\cdot}\!\square\!\!\rightarrow B$ might loose messages while inserting them into the buffer. Any write operation on the source end A might fail with probability τ in which case the buffer remains empty, or might be successful with probability $1-\tau$.

$$\begin{aligned}
\textbf{con}: &\quad \textbf{FtyFIFO1}[\tau](in : (A \mapsto \langle \alpha, a \rangle); out : (B \mapsto \langle \beta, b \rangle)) \\
\textbf{in}: &\quad \mathcal{WD}\langle \alpha, a \rangle \\
\textbf{out}: &\quad \mathcal{WD}\langle \beta, b \rangle \wedge FF(\langle \alpha, a \rangle, \langle \beta, b \rangle)
\end{aligned}$$

where

$$FF(\langle \alpha, a \rangle, \langle \beta, b \rangle) \equiv \begin{cases}
a(0) < b(0) < a(1) \wedge \beta(0) = \alpha(0) \wedge & \\
\quad FF(\langle \alpha', a' \rangle, \langle \beta', b' \rangle) & @1-\tau \\
a(1) < b(0) < a(2) \wedge \beta(0) = \alpha(1) \wedge & \\
\quad FF(\langle \alpha^{(2)}, a^{(2)} \rangle, \langle \beta', b' \rangle) & @\tau(1-\tau) \\
\cdots & \\
a(k-1) < b(0) < a(k) \wedge \beta(0) = \alpha(k-1) \wedge & \\
\quad FF(\langle \alpha^{(k)}, a^{(k)} \rangle, \langle \beta', b' \rangle) & @\tau^{k-1}(1-\tau) \\
\cdots & \\
\beta = () \wedge b = () & @\lim_{n\to\infty} \tau^n
\end{cases}$$

In other words, there are infinite alternatives when we consider infinite streams on the input and the probability for $\beta = () \wedge b = ()$ is $\lim_{n\to\infty} \tau^n = 0$.

Another kind of faulty FIFO1 channel $A \!-\!\!\square\!\overset{\tau}{\dashrightarrow} B$ might loose messages from its buffer, but works perfectly for the write operation on the source end A. The difference between this channel and **FtyFIFO1** is the possibilities for the data items to be successfully stored in the buffer and to be successfully taken from the buffer to the sink end, but the models which specify the relations between observations on input and output channel ends for these two channels are exactly the same.

A *message-corrupting synchronous channel* $A \!-\!\tau\!\!\rightarrow B$ is a synchronous channel with source node A and sink node B where the delivered message is corrupted with probability τ. The value τ serves as a parameter for this channel type. If A accepts a data item, then with probability $1-\tau$ the correct data value is obtained at B, but with probability τ, B takes a corrupted message $\perp$.

$$\begin{aligned}
\textbf{con}: &\quad \textbf{CptSync}[\tau](in : (A \mapsto \langle \alpha, a \rangle); out : (B \mapsto \langle \beta, b \rangle)) \\
\textbf{in}: &\quad \mathcal{WD}\langle \alpha, a \rangle \\
\textbf{out}: &\quad \mathcal{WD}\langle \beta, b \rangle \wedge b = a \wedge C(\alpha, \beta)
\end{aligned}$$

where

$$C(\alpha, \beta) \equiv ((\beta(0) = \eta_\perp)_\tau \oplus (\beta(0) = \alpha(0))) \wedge C(\alpha', \beta')$$

A *randomized synchronous channel* $A \xrightarrow{rand(0,1)} B$ generates a random number $b \in \{0, 1\}$ when it is activated through an arbitrary writing action at its source end A, and the random number is synchronously taken through the sink end B.

$$\begin{aligned}
\textbf{con}: &\quad \textbf{RdmSync}[rand(0,1)](in : (A \mapsto \langle \alpha, a \rangle); out : (B \mapsto \langle \beta, b \rangle))\\
\textbf{in}: &\quad \mathcal{WD}\langle \alpha, a \rangle\\
\textbf{out}: &\quad \mathcal{WD}\langle \beta, b \rangle \wedge b = a \wedge R(\alpha, \beta)
\end{aligned}$$

where

$$R(\alpha, \beta) \equiv \; ((\beta(0) = \eta_0)\,{}_{\frac{1}{2}} \oplus (\beta(0) = \eta_1)) \; \wedge R(\alpha', \beta')$$

A *probabilistic lossy synchronous channel* $A \overset{\tau}{- - \rightarrow} B$ requires both channel ends A and B to be available to synchronize. However, the transmission of the message fails with a certain probability τ, while the correct message passing occurs with probability $1 - \tau$.

$$\begin{aligned}
\textbf{con}: &\quad \textbf{ProbLossy}[\tau](in : (A \mapsto \langle \alpha, a \rangle); out : (B \mapsto \langle \beta, b \rangle))\\
\textbf{in}: &\quad \mathcal{WD}\langle \alpha, a \rangle\\
\textbf{out}: &\quad \mathcal{WD}\langle \beta, b \rangle \wedge PL(\langle \alpha, a \rangle, \langle \beta, b \rangle)
\end{aligned}$$

where

$$\begin{aligned}
&PL(\langle \alpha, a \rangle, \langle \beta, b \rangle)\\
\equiv\; &PL(\langle \alpha', a' \rangle, \langle \beta, b \rangle)_\tau \oplus ((b(0) = a(0)) \wedge (\beta(0) = \alpha(0)) \wedge PL(\langle \alpha', a' \rangle, \langle \beta', b' \rangle))
\end{aligned}$$

This channel type has to be not confused with the non-probabilistic lossy synchronous channel (depicted by a dashed line without any parameter).

4.3 Composition Operators

Different channels can be composed by linking their channel ends together into nodes to build more complex connectors. The formalization of nodes sometimes becomes rather complicated, especially when an arbitrary number of incoming and outgoing edges are involved. Therefore, we introduce two ternary channels *Replicator* and *Merger*, as shown in Fig. 3, and use their combinations to capture the behavior of arbitrary source, sink or mixed nodes.

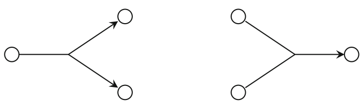

Fig. 3. Replicator and merger

Replicator is a synchronous broadcasting channel with one source end A and two sink ends B, C. The channel accepts input data values from A, and broadcasts them to B, C iff both B and C are ready to accept the data.

$$\begin{aligned}
\textbf{con}: &\quad \textbf{Replicator}(in : (A \mapsto \langle\alpha, a\rangle); out : (B \mapsto \langle\beta, b\rangle, C \mapsto \langle\gamma, c\rangle)) \\
\textbf{in}: &\quad \mathcal{WD}\langle\alpha, a\rangle \\
\textbf{out}: &\quad \mathcal{WD}\langle\beta, b\rangle \wedge \mathcal{WD}\langle\gamma, c\rangle \wedge \beta = \gamma = \alpha \wedge b = c = a
\end{aligned}$$

Merger is a channel that has two source ends A, B and one sink end C, which collects inputs from either A or B and sends them to C simultaneously if C is ready to accept the data.

$$\begin{aligned}
\textbf{con}: &\quad \textbf{Merger}(in : (A \mapsto \langle\alpha, a\rangle, B \mapsto \langle\beta, b\rangle); out : (C \mapsto \langle\gamma, c\rangle)) \\
\textbf{in}: &\quad \mathcal{WD}\langle\alpha, a\rangle \wedge \mathcal{WD}\langle\beta, b\rangle \wedge \mathcal{DF}(a, b) \\
\textbf{out}: &\quad \mathcal{WD}\langle\gamma, c\rangle \wedge M(\langle\alpha, a\rangle, \langle\beta, b\rangle, \langle\gamma, c\rangle)
\end{aligned}$$

where

$$\mathcal{DF}(a, b) =_{df} a(0) \neq b(0) \wedge \begin{cases} \mathcal{DF}(a', b) & \text{if } a(0) < b(0) \\ \mathcal{DF}(a, b') & \text{if } a(0) > b(0) \end{cases}$$

and the ternary relation M is defined as

$$\begin{aligned}
&M(\langle\alpha, a\rangle, \langle\beta, b\rangle, \langle\gamma, c\rangle) \\
&= \begin{cases} \gamma(0) = \alpha(0) \wedge c(0) = a(0) \wedge M(\langle\alpha', a'\rangle, \langle\beta, b\rangle, \langle\gamma', c'\rangle) & \text{if } a(0) < b(0) \\ \gamma(0) = \beta(0) \wedge c(0) = b(0) \wedge M(\langle\alpha, a\rangle, \langle\beta', b'\rangle, \langle\gamma', c'\rangle) & \text{if } a(0) > b(0) \end{cases}
\end{aligned}$$

Once we have the replicator and merger defined as channels as well, the only composition operator for connectors is *flow-through*. For two connectors $\mathbf{R}_1$ and $\mathbf{R}_2$, suppose one sink node of $\mathbf{R}_1$ and one source node of $\mathbf{R}_2$ are joined together into a new node. In this case, the new node becomes a *mixed node* which behaves as a self-contained pumping station. When we compose connectors, the events on the mixed nodes happen silently and automatically whenever they can, without the participation or even the knowledge of the environment. Such mixed nodes are hidden (encapsulated) by using the existential quantifier.

For $i = 1, 2$, let

$$\begin{aligned}
\textbf{con}: &\quad \mathbf{R}_i(in : in_{\mathbf{R}_i}; out : out_{\mathbf{R}_i}) \\
\textbf{in}: &\quad P_i(in_{\mathbf{R}_i}) \\
\textbf{out}: &\quad Q_i(in_{\mathbf{R}_i}, out_{\mathbf{R}_i})
\end{aligned}$$

denote the two connectors being composed by the flow-through composition. Suppose one sink node B_1 of $\mathbf{R}_1$ and one source node B_2 of $\mathbf{R}_2$ are joined together into a mixed node B. Let $B_1 \mapsto \langle\beta_1, b_1\rangle \in out_{\mathbf{R}_1}$ and $B_2 \mapsto \langle\beta_2, b_2\rangle \in in_{\mathbf{R}_2}$ be the output on the node B_1 in $\mathbf{R}_1$ and input on the node B_2 in $\mathbf{R}_2$, respectively. Then the new connector is denoted by $\mathbf{R} = \mathbf{R}_1;_{(B_1,B_2)\mapsto B}\mathbf{R}_2$, and defined as follows:

$$
\begin{aligned}
\textbf{con}: &\quad \mathbf{R}(in : (\bigcup_{i=1,2} in_{\mathbf{R}_i}) \setminus \{B_2 \mapsto \langle \beta_2, b_2 \rangle\}; out : (\bigcup_{i=1,2} out_{\mathbf{R}_i}) \setminus \{B_1 \mapsto \langle \beta_1, b_1 \rangle\}) \\
\textbf{in}: &\quad P_1(in_{\mathbf{R}_1}) \wedge \neg(\exists \langle \beta, b \rangle.(Q_1(in_{\mathbf{R}_1}, out_{\mathbf{R}_1})[\langle \beta, b \rangle / \langle \beta_1, b_1 \rangle] \wedge \\
&\quad \neg P_2(in_{\mathbf{R}_2})[\langle \beta, b \rangle / \langle \beta_2, b_2 \rangle])) \\
\textbf{out}: &\quad \exists \langle \beta, b \rangle . Q_1(in_{\mathbf{R}_1}, out_{\mathbf{R}_1})[\langle \beta, b \rangle / \langle \beta_1, b_1 \rangle] \wedge Q_2(in_{\mathbf{R}_2}, out_{\mathbf{R}_2})[\langle \beta, b \rangle / \langle \beta_2, b_2 \rangle]
\end{aligned}
$$

where for a predicate P, if v is a variable in P, $P[u/v]$ is the predicate obtained by replacing all occurrences of v in P by u.

Example 1. We consider the randomized router given in Fig. 4 as a simple example. This connector has one source node A and two sink nodes B and C, which randomly chooses B or C (both with probability $\frac{1}{2}$) to obtain the data written at A. It is constructed by composing two synchronous channels, two filter channels, two synchronous drains, two lossy synchronous channels and one randomized synchronous channel. This connector can be easily obtained from the composition of the basic channels (with replicators at A, D, E, G) after some equivalent transformations and quantifier eliminations:

$$
\begin{aligned}
\textbf{con}: &\quad \textbf{RandRouter}(in : (A \mapsto \langle \alpha, a \rangle); out : (B \mapsto \langle \beta, b \rangle, C \mapsto \langle \gamma, c \rangle)) \\
\textbf{in}: &\quad \mathcal{WD}\langle \alpha, a \rangle \\
\textbf{out}: &\quad \mathcal{WD}\langle \beta, b \rangle \wedge \mathcal{WD}\langle \gamma, c \rangle \wedge RR(\langle \alpha, a \rangle, \langle \beta, b \rangle, \langle \gamma, c \rangle)
\end{aligned}
$$

where

$$
\begin{aligned}
&RR(\langle \alpha, a \rangle, \langle \beta, b \rangle, \langle \gamma, c \rangle) \\
\equiv &(\beta(0) = \alpha(0) \wedge b(0) = a(0) \wedge RR(\langle \overrightarrow{\alpha}, \overrightarrow{a} \rangle, \langle \overrightarrow{\beta}, \overrightarrow{b} \rangle, \langle \gamma, c \rangle)) \,{}_{\frac{1}{2}}\!\oplus \\
&(\gamma(0) = \alpha(0) \wedge c(0) = a(0) \wedge RR(\langle \overrightarrow{\alpha}, \overrightarrow{a} \rangle, \langle \beta, b \rangle, \langle \overrightarrow{\gamma}, \overrightarrow{c} \rangle))
\end{aligned}
$$

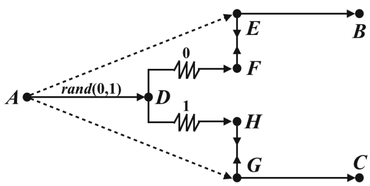

Fig. 4. Random router

5 Implementation

The implementation of this relational model for probabilistic connectors has been developed in Coq. Coq is a widely-used formal proof management system which provides a formal language called *Gallina* to write definitions, mathematical propositions and theorems, together with an environment for interactive construction of formal proofs. One of the main advantages of using Coq is that it is equipped with a set of well-developed standard libraries. For example, *Stream* provides a co-inductive definition of infinite sequences, *Reals* defines various operations and axioms on real numbers, and *Utheory* axiomatizes the properties required on the abstract type U representing the real interval $[0, 1]$. In general, quite a few axioms and theorems are predefined in such libraries. This makes it easy to support continuous time behavior and describe probabilistic channels. Moreover, any valid Coq expression can be used to depict properties, which is more powerful than just using formulas in one logic, like LTL or CTL.

The source code of the formalization in Coq is available at [19]. Compared with the initial formalization for (non-probabilistic) Reo connectors, the probabilistic behavior is captured properly in this extension. As described in Sect. 3, the observed sequences on nodes are adjusted to timed data distribution streams instead of timed data streams. But this new formalization can still be consistent with the initial one through assigning the value 1 to the companied probability of the data (i.e., the point distribution η_d instead of data item d). Based on this foundation and the specific library $Utheory$, the behavior of probabilistic channels can be characterized by the input and output timed data distribution streams properly. The probability accompanied the data will be updated accordingly when the timed data distribution pair flows through different probabilistic channels. With the definitions of channels serving as the basis, connector properties, as well as equivalence and refinement relations between different connectors can be naturally formalized as theorems in Coq and proved using tactics predefined in Coq[2].

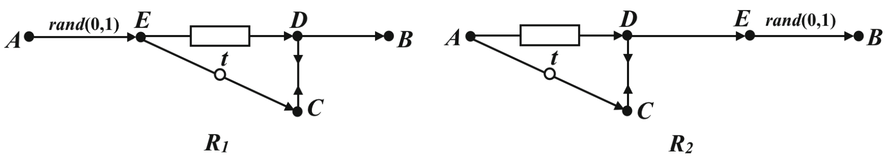

Fig. 5. Equivalence between connectors

Example 2. An interesting example of the equivalence relation between connectors is shown in Fig. 5. The two connectors are composed with the same set of basic channels but with different topologies of combination. Connector $\mathbf{R}_1$ is constructed by a randomized synchronous channel followed by a subconnector **tFIFO1** (which will be introduced in the following), while $\mathbf{R}_2$ is constructed by the subconnector **tFIFO1** and a following randomized synchronous channel. The subconnector **tFIFO1** contains a FIFO1 channel, a SyncDrain channel, a timer channel with parameter t and a Sync channel. It has been studied in [13]

[2] For two connectors $\mathbf{R}_1$ and $\mathbf{R}_2$, we say that $\mathbf{R}_2$ is a refinement of $\mathbf{R}_1$ (denoted by $\mathbf{R}_1 \sqsubseteq \mathbf{R}_2$) if $(P_1 \Rightarrow P_2) \wedge (P_1 \wedge Q_2 \Rightarrow Q_1)$, and they are equivalent if $\mathbf{R}_1 \sqsubseteq \mathbf{R}_2$ and $\mathbf{R}_2 \sqsubseteq \mathbf{R}_1$.

and properties related to its behavior have been proved in Coq. For the basic FIFO1 channel, the input and output timed data distribution streams will have the same data distribution but with an arbitrary time delay. Compared with the basic FIFO1 channel, the time delay is fixed by the parameter t in **tFIFO1**, apart from the same data distribution between the input and output streams.

The goal (formalized as a theorem) in this example is the equivalence relation between connectors $\mathbf{R}_1$ and $\mathbf{R}_2$ in Fig. 5. Before proving the equivalence relation, the configurations of the two connectors are first reduced to the constitution of a **RdmSync** channel and a **tFIFO1** connector with different topological orders for proof simplicity. This reduction leads to two more lemmas that need to be proved, which are the equivalence relations between the construction from basic channels and the reduced method of construction from a **RdmSync** channel and a **tFIFO1** connector. The two equivalence relations are formalized in Coq as follows:

```
Lemma RSync_tFIFO_eq: forall (A B: Stream TDD) (t:Time),
      exists E: Stream TDD,
     (RdmSync A E) /\ (t_FIFO1 E B t)
     <->
     (RdmSync A E) /\
      (exists (D C:Stream TDD), (FIFO1 E D) /\ (SyncDrain D C)
      /\ (Timert E C t) /\ (Sync D B)).

Lemma tFIFO_RSync_eq: forall (A B: Stream TDD) (t:Time),
     exists E: Stream TDD,
     (t_FIFO1 A E t) /\ (RdmSync E B)
     <->
     (exists (D C:Stream TDD), (FIFO1 A D) /\ (SyncDrain D C)
     /\ (Timert A C t) /\ (Sync D E)) /\ (RdmSync E B).
```

Once these two equivalence relations are proved, we can establish the goal of equivalence between $\mathbf{R}_1$ and $\mathbf{R}_2$ as the following theorem:

```
Theorem equivalence: forall (A B:Stream TDD) (t:Time),
  (exists E, (RdmSync A E) /\ (t_FIFO1 E B t))
<->
  (exists R, (t_FIFO1 A R t) /\ (RdmSync R B)).
```

The core of the proof for this theorem is that we need to find the corresponding intermediate timed data distribution streams to complete the construction, with the construction method of the other connector provided. The equivalence proof of this example is different from the one in [20]. Unlike the proof of equivalence in [20], we cannot find one single timed data distribution stream directly serving as a match. Thus, two timed data distribution streams are constructed first and then proved as precise matches for the refinement relations in two directions, respectively. The complete proof of the theorem is available at [19].

It is straightforward to find out the reason why the commutative property is satisfied in the construction of $\mathbf{R}_1$ and $\mathbf{R}_2$ in Fig. 5. The RdmSync channel only modifies the data distribution streams while the tFIFO1 connector only transforms the time stream. As a result, the change of topological positions of these two connectors does not affect the final relation between the timed data distribution streams on the source node A and sink node B.

Actually, as this model focuses on the relations between input and output timed data distribution streams, different orders of data distribution and time stream transformations lead to the same resultant relations. Therefore, for any two connectors (or channels), as long as these two connectors transform time streams and data distribution streams exclusively, the composition order will satisfy the commutative property.

Although this example is a bit trivial, it is presented as a demonstration of the possibility to express all well-defined properties or equivalence relations between connectors and develop machine checked proof in Coq. The original formalization of classic Reo can model a certain range of scenarios, but it is not good at dealing with the uncertainty of the real world. With this probabilistic Reo extension provided, formal modeling and reasoning about uncertainty is supported. As a result, more scenarios in real world can be modeled, and the crucial issues or properties need to be considered can be further verified in Coq.

6 Conclusion and Future Work

This paper extends our previous work on the design model for (unprobabilistic) Reo connectors and introduces the relational model for probabilistic Reo connectors based on observations as timed data distribution streams. This approach provides a unified semantic model for different kinds of channels and connectors, covers different communication mechanisms encoded in Reo, and allows the combination of both deterministic and probabilistic channels in Reo. In this work, we model (both deterministic and probabilistic) channels in Reo as relations of timed data distribution streams, where the observation on each node of a connector is specified as a stream of timed data distribution. The composition of connectors is captured by flow-through composition with the help of two ternary channels *merger* and *replicator*. Our semantic model offers potential benefits in developing tool support for Reo. For example, the syntax and semantics for probabilistic Reo connectors are implemented in Coq, which makes it possible to prove connector properties, as well as equivalence and refinement relations between different connectors.

Incorporating more complex probabilistic and stochastic constraints on connectors [9,15] into our model is an interesting topic that we are now investigating. In future work, we also plan to incorporate the hybrid connectors [10], and other QoS aspects on connectors [4,5] into this model. The development of refinement and testing theories for probabilistic connectors like refinement and testing for deterministic connectors in [1,17] and integration of such theories into Coq or other existing tools for Reo [11] are of special interest and in our scope as well. On the other hand, we will investigate the inherent dynamic topology and mobility in "full" Reo based on the design model, especially context-sensitive connector behavior and reconfiguration of connectors.

Acknowledgement. The work was partially supported by the National Natural Science Foundation of China under grant no. 61772038, 61532019, 61202069 and 61272160.

References

1. Aichernig, B.K., Arbab, F., Astefanoaei, L., de Boer, F.S., Sun, M., Rutten, J.: Fault-based test case generation for component connectors. In: Proceedings of TASE 2009, pp. 147–154. IEEE Computer Society (2009)
2. Arbab, F.: Reo: a channel-based coordination model for component composition. Math. Struct. Comput. Sci. **14**(3), 329–366 (2004)
3. Arbab, F., Baier, C., de Boer, C., Rutten, J.: Models and temporal logics for timed component connectors. In: Cuellar, J.R., Liu, Z. (eds.) Proceedings of SEFM 2004, pp. 198–207. IEEE Computer Society (2004)
4. Arbab, F., Chothia, T., Meng, S., Moon, Y.-J.: Component connectors with QoS guarantees. In: Murphy, A.L., Vitek, J. (eds.) COORDINATION 2007. LNCS, vol. 4467, pp. 286–304. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72794-1_16
5. Arbab, F., Chothia, T., van der Mei, R., Meng, S., Moon, Y.J., Verhoef, C.: From coordination to stochastic models of QoS. In: Field, J., Vasconcelos, V.T. (eds.) COORDINATION 2009. LNCS, vol. 5521, pp. 268–287. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02053-7_14
6. Arbab, F., Rutten, J.J.M.M.: A coinductive calculus of component connectors. In: Wirsing, M., Pattinson, D., Hennicker, R. (eds.) WADT 2002. LNCS, vol. 2755, pp. 34–55. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-40020-2_2
7. Baier, C.: Probabilistic models for Reo connector circuits. J. Univers. Comput. Sci. **11**(10), 1718–1748 (2005)
8. Baier, C., Sirjani, M., Arbab, F., Rutten, J.: Modeling component connectors in Reo by constraint automata. Sci. Comput. Program. **61**, 75–113 (2006)
9. Baier, C., Wolf, V.: Stochastic reasoning about channel-based component connectors. In: Ciancarini, P., Wiklicky, H. (eds.) COORDINATION 2006. LNCS, vol. 4038, pp. 1–15. Springer, Heidelberg (2006). https://doi.org/10.1007/11767954_1
10. Chen, X., Sun, J., Sun, M.: A hybrid model of connectors in cyber-physical systems. In: Merz, S., Pang, J. (eds.) ICFEM 2014. LNCS, vol. 8829, pp. 59–74. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11737-9_5
11. Eclipse Coordination Tools. http://reo.project.cwi.nl/
12. He, K., Hermanns, H., Chen, Y.: Models of connected things: on priced probabilistic timed Reo. In: 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), vol. 1, pp. 234–243 (2017)
13. Hong, W., Nawaz, M.S., Zhang, X., Li, Y., Sun, M.: Using Coq for formal modeling and verification of timed connectors. In: Cerone, A., Roveri, M. (eds.) SEFM 2017. LNCS, vol. 10729, pp. 558–573. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74781-1_37
14. Jongmans, S.T.Q., Arbab, F.: Overview of thirty semantic formalisms for Reo. Sci. Ann. Comput. Sci. **22**(1), 201–251 (2012)
15. Li, Y., Zhang, X., Ji, Y., Sun, M.: Capturing stochastic and real-time behavior in Reo connectors. In: Cavalheiro, S., Fiadeiro, J. (eds.) SBMF 2017. LNCS, vol. 10623, pp. 287–304. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70848-5_18

16. Oliveira, N., Silva, A., Barbosa, L.S.: IMC_{Reo}: interactive Markov chains for Stochastic Reo. J. Internet Serv. Inf. Secur. **5**(1), 3–28 (2015)
17. Sun, M., Arbab, F., Aichernig, B.K., Astefanoaei, L., de Boer, F.S., Rutten, J.: Connectors as designs: modeling, refinement and test case generation. Sci. Comput. Program. **77**(7–8), 799–822 (2012)
18. The Coq Proof Assistant. https://coq.inria.fr/
19. The source code of Probabilistic Reo. https://github.com/Xiyue-Selina/Prob-Reo
20. Zhang, X., Hong, W., Li, Y., Sun, M.: Reasoning about connectors in Coq. In: Kouchnarenko, O., Khosravi, R. (eds.) FACS 2016. LNCS, vol. 10231, pp. 172–190. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57666-4_11