Using PVS for Modeling and Verification of Probabilistic Connectors

M. Saqib Nawaz, Meng Sun

Using PVS for Modeling and Verification of Probabilistic Connectors

M. Saqib Nawaz and Meng Sun

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

Abstract. Reo is a channel-based coordination language that allows the construction of connectors to coordinate behavior among different components in distributed systems. Probabilistic connectors in Reo capture the random and probabilistic behavior to deal with the uncertainty of the real world. In this paper we use PVS to provide a mechanical formalization for probabilistic connectors. We first present the formalization of random/probabilistic channels and the composition operators in PVS. Random and probabilistic channels are modeled as relations on timed data distribution sequences that are observed at the source and sink ends of these channels. Composition operators are used to combine random/probabilistic channels together with primitive channels to construct complex component connectors. The approach can be used to naturally specify complex connectors and prove important properties for probabilistic connectors as well as the refinement/equivalence relations between them with the PVS proof assistant.

Keywords: Reo, PVS, Random/probabilistic connectors, Specification, Verification.

1 Introduction

Large-scale distributed systems, that are generally transparent and heterogeneous in nature, are built from components that interact with each other to perform some specific tasks. Coordination languages offer possible binding for components in a distributed environment to make the interactions possible. Reo [2,8] is a popular exogenous coordination language where exogenous coordination [1] means that the primitives that support the coordination of an entity with others reside outside of that entity. Reo allows the orchestration of complex connectors from simple ones (called channels) through composition operators.

Connectors in Reo provide the protocols for controlling and organizing the communication, synchronization and cooperation among the components that they interconnect. Formal analysis and verification of connectors have gained much interest in the past decade for component-based software engineering due to the recent evolution of software systems and advancements in cloud and grid technologies. It is also important to certify the correctness of connectors, which

makes large-scale distributed systems more reliable. Some works have been done in this regard in the past years. For example, a modeling approach based on first-order relational logic in *Alloy* modeling language was provided in [14] for Reo connectors. The symbolic model checker "*Vereofy*" has been developed in [6] to verify CTL-like properties for connectors. Moreover, a formal transformation from Reo to the specification language *mCRL2* that is based on process algebra was presented in [15]. The models were then verified conveniently with the *mCRL2* model checker.

Complex distributed systems need to incorporate many aspects of the communication and coordination between components, such as nondeterminism, probabilistic and stochastic interactions, real-time information and resource consumption, etc. The works reported in [3–5, 9, 12, 17, 22] extend classical Reo from different perspective to deal with such requirements. The Unifying Theories of Programming (UTP) semantic framework was used in [21, 23] to formalize connectors by providing design models for untimed and timed Reo connectors respectively, and recently extended in [24] to cover connectors that are composed from channels with random and probabilistic behavior. The theorem proving technique has been used in [18] to encode and reason about the design models for untimed/timed Reo connectors in PVS [19]. In this paper, we extend the approach to cover the formalization for Reo connectors with random and probabilistic behavior. The basic idea is to model the observable behavior of a probabilistic connector as a relation on the timed data distribution sequences being observed at the source (input) and sink (output) ends of the connector. The extended approach covers the scenarios for unpredictable, uncertain behavior. Furthermore, the refinement/equivalence relations between probabilistic connectors can be formalized and verified in PVS easily.

Our mechanized verification for probabilistic analysis of connectors is certainly not the first one. A variant of constraint automata called probabilistic constraint automata (PCA) has been developed in [5] to provide the operational semantics for probabilistic Reo connectors. Stochastic Reo automata was proposed in [17] to compositionally derive a QoS-aware semantics for Reo. The automata model was translated to Continuous-Time Markov Chains (CTMCs) so that third-party verification tools can be used for stochastic analysis. Similarly, priced probabilistic timed constraint automata (pPTCA) was used in [12] for the reasoning about nondeterministic, probabilistic and timed behavior with aspects of energy consumption. Reo was also used in [7] to coordinate modules in the PRISM model checker. Although such formalisms scale up quite well, they suffer from the state space explosion problem as Reo connectors generally describe the manifold interactions among components that they interconnect, rather than simple input-output behavior on one individual interface. Moreover, the modeling and verification of unbounded primitives or even bounded primitives with unbounded data domains always lead to the state space explosion problem, which cannot be solved with such finite automata models. However, such behavior can be specified and verified efficiently in theorem provers as shown in our previous works [13, 18, 25, 26].

The remainder of the paper is organized as follows: The coordination language Reo is briefly introduced in Section 2. In Section 3, we present the specifications in PVS for some basic definitions that are used later to model random/probabilistic channels. Section 4 presents the formal modeling of random/probabilistic channels and composition operators in PVS. Section 5 shows how to reason about properties of probabilistic connectors in PVS and refinement/equivalence relations between them. Finally, Section 6 concludes the paper with some future work. The PVS dump file for this work can be found at [20].

2 Preliminaries

Reo offers a compositional framework where component *connectors* can be constructed from primitive *channels* of arbitrary types through composition operators. Connectors provide the protocol for controlling and organizing the communication, synchronization and cooperation between components. Each channel has two channel ends, with one of two types: *source* and *sink*. A source end provides input values to the channel via write actions and a sink end dispenses data out of the channel with read actions. A channel's ends can also be both sinks or both sources. Figure 1 shows few primitive channel types in Reo.

Fig. 1. Some primitive channels in Reo

A *synchronous (Sync) channel* has one source and one sink end. Input/Output (I/O) operations can succeed only if the writing and reading operation is synchronized at source and sink end respectively. A *lossy synchronous (LossySync) channel* is a variant of the *Sync* channel. Data items in *LossySync* are transferred successfully if the write operation on the source end and the read operation on the sink end occur simultaneously, otherwise the data items are lost. A *FIFO1 channel* has one buffer cell of capacity 1, one source end and one sink end. *FIFO1* accepts a data item whenever the buffer is empty. After accepting a data item from the source end, it is first stored in the buffer and dispensed out of the channel through the sink end later in the FIFO order. The *synchronous Drain (SyncDrain) channel* is used for synchronizing the writing operations at its two source ends. It has no sink end and all written data items are lost. A *t-timer channel* accepts any data item at its source end and produces a *timeout* signal on its sink end after a delay of t time units. Further details on Reo and primitive untimed/timed channels can be found in [2, 3, 21, 23]. Furthermore, users can specify new channel types with their own requirements and interaction policies in Reo. For example, several probabilistic and stochastic extensions of Reo have been proposed in [5, 9, 16].

A connector can be depicted visually as a graph with some additional information. The nodes represent sets of the channel ends and the edges represent the

established channels between the nodes. Nodes can be categorized into source, sink or mixed nodes, depending on whether the node contains only source channel ends, sink channel ends, or both. Therefore, source nodes are analogous to input ports and sink nodes to output ports. Data that flow through source and sink nodes depends on the pending write and read operations of the environment. For channel composition, three types of operators are used, which are (1) *flow-through*, (2) *replicate* and (3) *merge*, as shown in Figure 2.

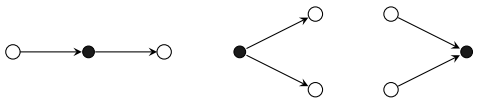

Fig. 2. Operators for channel composition

3 Basic Definitions in PVS

The behavior of untimed and timed connectors are formalized by modeling their observable behavior as relations on the timed data sequences at their sink and source nodes. For random and probabilistic behavior, sequences of data distributions where the data passes through connector nodes together with the time moments for data items observation emerges as the key building block to properly describe the connectors. Therefore, the observation on nodes can be specified naturally as *timed data distribution* (TDD) sequences for connectors that behaves probabilistically or randomly.

The PVS library for *probability* [10] and some pre-defined functions from PVS prelude are used in the modeling of random and probabilistic channels. The probability library is built on the firm foundations for probability theory [11]. Based on a σ-algebra, probability measure and probability space, the distribution function (*df*) for a real-valued random variable X is defined in PVS. We are interested in the cumulative distribution function (CDF) of a random variable. To deal with continuous random variable, we would partially instantiate the sample space T with *real*, σ-algebra with *borel_set* (borel sets) and specify the probability measure to describe the distributions for random variable.

A record structure in PVS is used to represent the *TDD* sequences on the sink and source nodes, where *time* is defined as a *positive real number* ($\mathbb{R}^+$), which is natural and expressive enough for the modeling of connector behavior. For untimed/timed channels, *data* is defined as a *positive type*. To capture the probabilistic behavior, *data* is defined as a function of type $[\ T \rightarrow real\ ]$ (where T is a positive (non-empty) type). With such a kind of functions, other abstract sets of data can be processed first by mapping them to a set of real numbers in an appropriate way. Moreover, such mapping for data can be expanded easily in accordance with different application domains. The data distribution *DD* is defined as a Cartesian product with square brackets [_, _] in PVS.

```
Time: Type = posreal
```

```
Data: TYPE [T -> real]

dfs?(F:[real->probability]):bool = EXISTS X: FORALL x: F(x) = P(X <= x)
df: TYPE+ = (dfs?) CONTAINING (LAMBDA x: IF x < 0 THEN 0 ELSE 1 ENDIF)

DD: TYPE = [Data, df]

TDD: TYPE = [# T: sequence[Time],
               D: sequence[DD] #]

Input, Output: VAR TDD
```

A *TDD* is a record structure type that has two components: T and D. The T component is a sequence of *time points* being used to represent the *time* when the data in the D component is observed. The D component is a sequence of *data distributions*. The *Input* and *Output* are declared as variables of type *TDD*. The components of a record type can be accessed through the corresponding field name. For example in our case, the T component of *Input* is accessed by $Input`T$.

Since the type of component T in *TDD* is defined as $sequence[Time]$, we have to define the operators "<" and ">" for sequences of times. A strict order (that is both transitive and irreflexive) is assumed for "<" and ">". The type system of PVS is not algorithmically decidable and may lead to proof obligations called type correctness conditions (TCCs). Defining "<,>" for sequence of time generated two TCCs. Proof steps for these two TCCs can be found at [20].

```
<: (strict_order?[sequence[Time]])
>: (strict_order?[sequence[Time]]) =
   LAMBDA (s1, s2: sequence[Time]): s2 < s1
```

Some more functions and predicates are used in PVS for concise modeling of probabilistic channels and composition operators. For example, *Teq* returns *true* if the time of two sequences are exactly equal to each other. *Tle* (*Tgt*) represents that time of the first sequence is strictly less (greater) than the second sequence. *Deq* (*Dneq*) shows the equality of data: data sequence at one end is equal (not equal) to data sequence at the other end. For primitive (untimed) channels, the time of input sequence is a simulation of real time which means that time sequence is increasing as time passes by. For probabilistic connectors including timed channels, some more predicate formulas are defined in a similar way. For example, *Tltt* (*Tgtt*) represents that the time of the first sequence with a delay t is less (greater) than the second sequence. And the *next* function takes a *TDD* and returns the derivative of the sequence. Suffix function that is used in the *next* function is used to return a suffix sequence and its definition can be found in the prelude library of PVS.

```
next(T1): TDD = T1 WITH [T:=(suffix(T1`T, 1)),
                         D:=(suffix(T1`D, 1))]
```

4 Probabilistic Channels and Operators

The modeling of some basic probabilistic/random channels and composition operators that are used to construct probabilistic connectors is presented in this section.

4.1 Random and Probabilistic Channels

The behavior of probabilistic/random channels are specified in PVS with the disjunction or conjunction of different predicates and constraints on the *TDD* sequences at source and sink nodes. We consider one random channel, *randomized synchronous channel*, and four probabilistic channels: *message-corrupting synchronous channel*, *probabilistic lossy synchronous channel*, *faulty FIFO1 channel* and *lossy FIFO1 channel*.

RSync: Randomized synchronous channel ($A \xrightarrow{rand(0,1)} B$) is the randomized variant of synchronous channel. When the channel is activated through an arbitrary write operation at source node A, it generates a random number $b \in \{0, 1\}$ at sink node B. Sink node synchronously takes the random number with equal probability for 0 (*zero*) and 1 (*one*). In *RSync*, *zero* and *one* are declared as random variables. Their specifications generated two TCCs for expected type *random variable*, which is proved interactively with the PVS theorem prover. To prove both TCCs, we used already defined judgment *constant_is_measurable* in the *measure space definition* theory that can be found in the library *measure_integration*. The proofs for both TCCs are omitted here because of the page limitation and can be found at [20]. *RSync* is specified as follows:

```
zero: random_variable = (LAMBDA t: 0)
one:  random_variable = (LAMBDA t: 1)

RSync(Input, Output): bool = FORALL(n:nat):
        Output`D(n) = (zero, oah) OR
         Output`D(n) = (one, oah) &
             Teq(Input, Output)
```

The universal quantification and the first disjunction capture the random behavior being observed at the sink node. Each data element in the *TDD* sequence at the sink node can be either *zero* or *one* with probability *oah*, which is defined as:

```
oah(x): probability = 1/2
```

The synchronous behavior for this channel is satisfied with the predicate *Teq*, which shows that the time for the occurrence of data elements being observed at both channel ends are equal.

CSync: Message corrupting synchronous channel ($A \xrightarrow{p} B$) is the probabilistic variant of synchronous channel, where with probability p, the delivered message can be corrupted. In such a channel, if a data element is written to the source end, then the probability that the exact correct data value will be obtained at the sink end is $1-p$. A corrupted data value, represented with c in the specification, will be obtained with probability p.

```
CSync(Input, Output)(p:probability):
 INDUCTIVE bool = (Output‘T(0) = Input‘T(0) &
  Output‘D(0) = (Input‘D(0)‘1,(1-p)*Input‘D(0)‘2))
   OR (EXISTS(c:Data): Output‘D(0) = (c,(p)*Input‘D(0)‘2)
    & CSync(next(Input), next(Output))(p))
```

The *CSync* channel is defined inductively. Inductive definitions in PVS, which are predicates with eventual range type *boolean*, are similar to recursive definitions as both involve induction and must satisfy some constraints to guarantee that they are total. The first formula is for the time equality constraint for the synchronous behavior. We can also model the first constraint with Teq predicate. The second and third formula with the disjunction reflects the probabilistic behavior. The sink node receives the same data that was written at the source node with the updated probability, where $1-p$ is multiplied with the probability for data at source node. On the other hand, sink node receives the corrupted value (c) with probability p multiplied with the probability for the written data. The last formula is for the recursive step that channel takes.

PLSync: In the probabilistic lossy synchronous channel ($A \xrightarrow{q} B$), the transmission of the message from the source to sink fails with probability q. And with probability $1-q$, *PLSync* acts like a standard *Sync* channel where the message is successfully transmitted from the source end to the sink end. In PVS, the *PLSync* channel is modeled as follows:

```
PLSync(Input, Output)(q:probability):
 INDUCTIVE bool = (Output‘T(0) = Input‘T(0) &
  Output‘D(0) = (Input‘D(0)‘1,(1-q)*Input‘D(0)‘2)
   & PLSync(next(Input), next(Output))(q)) &
    (Output‘D(0)‘2 = (q)*Input‘D(0)‘2 => PLSync(next(Input),Output)(q))
```

The *PLSync* channel is defined inductively but unlike *CSync*, it may take two different routes in each step. Three conjuncted formulas are for the case when the data is successfully received by the sink end. First formula satisfies the time constraint that ensures the synchronous behavior. Second formula reflects that the data item is received by the sink end with probability $1-q$ multiplied to the probability for that data at the source end. The third formula is the recursive step that channel takes when a data is transmitted successfully. For the case when the transmitted data is lost, the recursive behavior of the channel is reflected by the last two formulas with implication between them. In such case,

no data is obtained at the sink end.

FFIFO1: Faulty FIFO1 channel ($A \cdot \overset{r}{\dashrightarrow\!\square\!\rightarrow} B$) is a probabilistic variant of *FIFO1* channel, that might loose (with probability r) the message when it is inserted into the buffer and the buffer remains empty. It can also behave as a normal *FIFO1* channel where the insertion of the data into the buffer is successful with probability $1 - r$.

```
FFifo(Input, Output)(r:probability):
 INDUCTIVE bool = (Output‘T(0) > Input‘T(0) &
  Output‘T(0) < Input‘T(1) & Output‘D(0) =
  (Input‘D(0)‘1, (1-r)*Input‘D(0)‘2) &
   FFifo(next(Input), next(Output))(r)) &
    (Output‘D(0)‘2 = (r)*Input‘D(0)‘2 => FFifo(next(Input), Output)(r))
```

The *FFIFO1* channel is also defined inductively and like *PLSync*, it may take two different routes in each step. For the case when data written at source end is inserted successfully into the buffer, the channel should satisfy four constraints which are specified with the conjunction of four predicates. The first two formulas are for the time constraints, where the first formula is for the time delay between data at source and sink ends. As *FIFO1* has a buffering capacity of 1, next data item waits till the current data item in the buffer is taken out at the sink end. This is specified with the second formula that the time of the next data item is greater than the time for the current data item in the buffer. The last two formulas are for the recursive behavior that channel takes when the written data is lost before its insertion in the buffer. Like *PLSync*, no data is received at the sink end in such case.

LFIFO1: Another probabilistic variant called lossy FIFO channel ($A \overset{r}{\dashrightarrow\!\square\!\dashrightarrow} B$) might loose each stored data item with some fixed probability (r) in any step. Compared to *FFIFO1*, this channel may loose the data in the process of taking the data from the buffer. As channels are modeled by the relations between observations on source and sink ends, therefore, the specifications for *LFIFO1* and *FFIFO1* are same.

With this modeling approach, we can easily adjust the specifications for untimed and timed channels in a proper way, where the observations on source and sink ends of all channels are specified by *TDD* sequences. Then the probabilistic/random channels as well as untimed/timed channels can be combined together to build connectors. A connector is probabilistic if it constitutes at least one probabilistic or random channel.

4.2 Operators

Compositional operators can be applied on channels in various topological order for the construction of complex connectors. As already discussed in Section 2, there are three kinds of composition operators: (1) *flow-through*, (2) *replicate* and (3) *merge*.

The *flow-through* operator simply allows the data items to pass through mixed node(s) without any change. A component connected to a connector can write data items at source node and can obtain data items from the sink node. The *replicate* operator puts the source ends of different channels together into one source node. A write operation by a component on source node succeeds only if all coinciding channels ends accept the data item. The behavior of *flow-through* and *replicate* operators does not depend on the context of the data-flow. The approach for the modeling of these two operators in [18] can be adopted here without any change. The structure of connectors allow us to specify both operators implicitly by means of nodes renaming and conjunction instead of writing a new function.

This is explained with simple examples. For two channels *PLSync(A, B)* and *FIFO1(B, C)*, the *flow-through* operator acting on node B is implemented already. For two channels *PLSync(A, B)* and *FIFO1(C, D)*, the *replicate* operator can be implemented explicitly by renaming the C with A for the *FIFO1* channel. Using conjunction and node renaming for these two operators make it possible to specify connectors directly as lemmas and theorems.

Unlike *flow-through* and *replicate*, the *merge* operator depends on the content of the data-flow. The time and data dimension is same for *TD* and *TDD* sequences, which means that these two dimensions do not need any change. As we are dealing with data distribution, so the equality relation for data is changed to the equality relation on data distribution. Thus, both data items and their associated probabilities should be equal. Merge is modeled as follows in PVS:

```
Merge(s1,s2,s3:TDD): INDUCTIVE bool =
  (NOT s1‘T(0) = s2‘T(0)) & (s1‘T(0) < s2‘T(0) => s3‘T(0) = s1‘T(0) &
    s3‘D(0) = s1‘D(0) & Merge(next(s1),s2,next(s3))) & (s1‘T(0) > s2‘T(0)
    => s3‘T(0) = s2‘T(0) & s3‘D(0)= s2‘D(0) & Merge(s1,next(s2),next(s3)))
```

The modeling approach provided in this section for probabilistic/random channels and composition operators can be used to construct different probabilistic connectors according to their topological orders.

5 Reasoning

After connectors modeling, we can analyze and prove their properties. In this section, some examples are provided for the reasoning about probabilistic/random connectors as well as the refinement and equivalence relations between them.

Example 1. Figure 3 shows a probabilistic Reo connector that distributed components can use for message communication. Component 1 can deliver its messages to the connector via connecting to node *in*, while component 2 is connected to the node *out* to obtain the message from the connector. Messages are transmitted from component 1 to component 2 with FFIFO1 channel (AB). Other primitive channels (Sync, SyncD, FIFO1, LossySync) are organized in the connector to repeat component 1 message as often as necessary. The property that component 2 almost surely obtain the message via *out* from *in* is established with the following theorem in PVS.

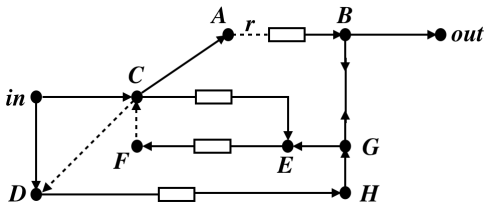

Fig. 3. A probabilistic Reo connector

Theorem 1. `Sync(in,D)` & `Sync(in,C)` & `Sync(C,A)` & `Fifo1(C,E)` & `Fifo1(E,F)` & `LSync(C,D)(n)` & `LSync(F,C)(n)` & `FFifo(A,B)(r)` & `Fifo1(D,H)` & `Sync(H,G)` & `SyncD(G,B)` & `Sync(G,E)` & `Sync(B,out)` => `out‘D(0)‘1 = in‘D(0)‘1` & `Tle(in,out)`

Proof. Mathematical induction is used to prove theorem 1. After applying induction on n, main goal is split into two sub-goals. The first sub-goal is for the base case and the other one is for the inductive case.

For the base case, the antecedent formula is simplified by creating a free *skolem* variable and removing *implies*. The definitions of channels and predicates are expanded. Some irrelevant formulas in the antecedent are suppressed with the *hide* command. The sub-goal is divided into two more sub-goals: one for the data dimension and other for the time dimension. Both sub-sub-goals are proved with PVS proof commands and decision procedures.

For the inductive case, the sequent formula is first simplified with repeated *skolemization* and *flattening*. In the first antecedent formula, the universal quantifiers are instantiated automatically with *inst?* commands. Sometimes a single *inst?* can only find a partial instantiation where successive invocations of *inst?* can succeed in fully instantiating all of the quantified variables. The rest of the proof is similar to the base case, where the sub-goal is split to two more sub-goals for data and time dimension. The detailed PVS proof for theorem 1 can be found at [20].

The notion of refinement has been adopted widely in development of complex systems. Refinement relation provides guarantee for the correctness of implementation with respect to the abstract specification of the same system, and thus helps in bridging the gap between requirements and the final implementations.

Here, we use the refinement relation for connectors defined in [23], where the refinement order over connectors is established on the basis of the implication relation of predicates. As discussed already, connectors are represented by conjunction of a set of predicates, where the variables are bound by the universal and existential quantification. Let C_1 and C_2 represent two connectors that are modeled by set of predicates. C_2 is a refinement of C_1 only if $C_2 \rightarrow C_1$, meaning the behavioral properties of C_1 can be derived from the properties of C_2. C_2 properties are regarded as hypothesis and the properties of connector C_1 as conclusion. The refinement relation between C_1 and C_2 is denoted as $C_1 \sqsubseteq C_2$. Next, we provide an example for refinement relation between probabilistic connectors.

Example 2 (Refinement). For the two connectors shown in Figure 4, connector P is a refinement of connector Q ($Q \sqsubseteq P$).

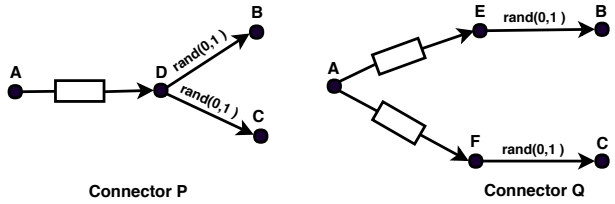

Fig. 4. Connectors refinement example

Given arbitrary input *TDD* sequence at node A and output *TDD* sequences at nodes B and C, connector P is a refinement of another connector Q only if the behavior property of Q can be derived from the connector P property. In connector Q, the outputs are not synchronized and data is received asynchronously by the sink ends B and C respectively. There is no constraints on the relationship between the time sequence of the two output events. On the other hand, P refines the behavior of Q by synchronizing the two sink nodes, which ensures that the two output events must happen simultaneously. We use a, b, c to denote the time sequence at nodes A, B and C respectively. Let d denotes the random number $d \in \{0, 1\}$, that ranges over all data items. Let β, γ represent the data sequence being observed at sink nodes B and C respectively. Probabilistic connector P satisfies the condition $a < b \wedge a < c \wedge b = c \wedge \beta = d^* \wedge \gamma = d^*$. Whereas, Q satisfies $a < b \wedge a < c \wedge \beta = d^* \wedge \gamma = d^*$. The refinement relation between Q and P is verified with following theorem.

Theorem 2. $\forall$ `(A,B,C:TDD)`*:*

`(∃ (D:TDD): Fifo1(A,D) & RSync(D, B) & RSync(D,C)) ⇒ (∃ (E,F:TDD): (Fifo1(A,E) & RSync(E,B)) & (Fifo1(A,F) & RSync(F,C)))`

Proof. The first suitable formula in the sequent ($\forall/\exists\ A, B, C : TDD$) is replaced to $TDD[A!1/A1, B!1/B1, C!1/C1]$ by creating three skolem constants. Implies is removed from consequent with *flattening*. Now we have one existentially quantified formula in both antecedent and consequent. Quantified formula in antecedent is reduced by automatic introduction of skolem constant with *skolem*! command. In consequent, we need to find the *TDD* sequences that specify the data flow through mixed nodes E and F for connector Q. In other words, we need to find an appropriate E and F that satisfies $(Fifo1(A, E) \wedge RSync(E, B)) \wedge (Fifo1(A, F) \wedge RSync(F, C))$. With (*inst 1 "D!1" "D!1"*), the first formula in consequent is instantiated where both E and F are substituted with $D!1$. Antecedent is divided into three formulas by removing logical &'s. Finally, the formula in the consequent is split into four sub-goals. All four sub-goals are trivially true and PVS proved those sub-goals automatically with *propax* propositional axioms. PVS proof tree for theorem 2 is shown in Figure 5.

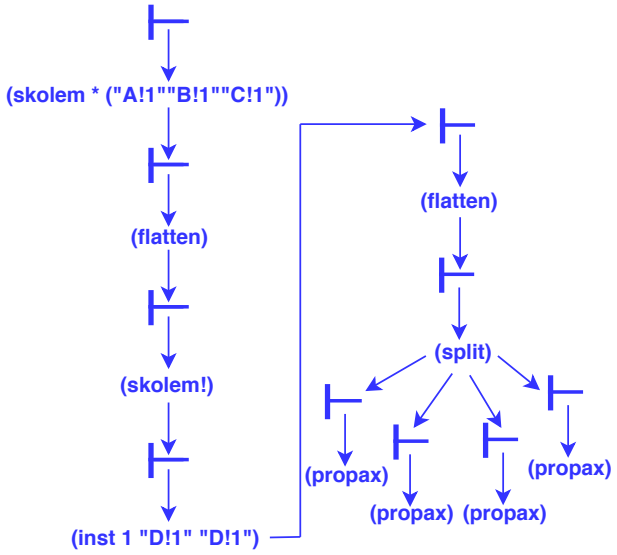

Fig. 5. Proof tree for connectors refinement proof

Generally, an equivalence relation is defined as a binary relation that holds the reflexivity, symmetric as well as transitivity properties. The equivalence relation between two connectors C_1 and C_2 is defined with mutual refinement:

$$C_1 \equiv C_2 \quad \textbf{iff} \quad C_1 \sqsubseteq C_2 \wedge C_2 \sqsubseteq C_1$$

Here, the equivalence relation is represented with implications that goes both ways, such as $C_2 \leftrightarrow C_1$.

Example 3 (Equivalence). Figure 6 shows two probabilistic connectors that are constructed by composing five channels RSync, FIFO1, t-Timer, SyncD and Sync in different topological orders. Both probabilistic connectors are equivalent ($R_1 \leftrightarrow R_2$), which is proved in PVS.

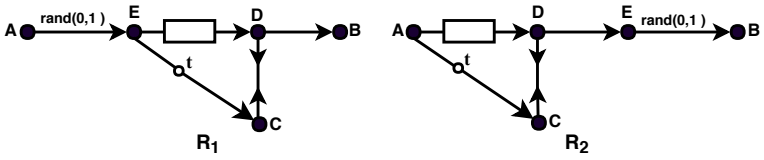

Fig. 6. Connectors equivalence example

The three untimed (*Sync, SyncD* and *FIFO1*) channels and one timed channel (*t-Timer*) can be combined together to make a timed connector known as *tFIFO1*, that was also studied in [13]. In a primitive *FIFO1* channel, the data distributions for the sequences at sink and source nodes are same with arbitrary

time delay. On the other hand, the time delay is fixed by the parameter t in *tFIFO1*. Here, we call the *tFIFO1* a sub-connector and is modeled as:

```
Tfifo(A, B)(t:Time)(d:Data): bool =
 EXISTS (R,S:TDD): Fifo1(A, R) & SyncD(R, S)
       & Timert(A, S)(t)(d) & Sync(R, B)
```

In general, the connectors build from same set of sub-connectors in commutative orders are not equivalent as the configuration of connectors do not satisfy the commutative law. However, connectors R_1 and R_2 are equal for the above example.

Unlike the general approach that we adopted previously to construct a connector from basic channels, connectors R_1 and R_2 are composed by connecting the *tFIFO1* sub-connector with the *RSync* channel in different topological order. The main reason to use the reduced method (where a sub-connector is combined with a channel) for connectors construction is to make the proof process simpler and easier to understand. The equivalence relations between a channel linked with a sub-connector in different positions are first proved as lemmas.

Lemma 1. ∀ (A,B:TDD)(t:Time)(d:Data)*:*

```
∃ (E:TDD): RSync(A,E) & Tfifo(E,B)(t)(d) <=> RSync(A,E) & ∃
  (C,D:TDD): Fifo1(E,D) & SyncD(D,C) & Timert(E,C)(t)(d) &
                          Sync(D,B)
```

Lemma 1 shows the equivalence relation between the reduced construction of a connector and a connector constructed from basic channels for R_1. Similarly for R_2, another lemma is provided.

Lemma 2. ∀ (A,B:TDD)(t:Time)(d:Data)*:*

```
∃ (E:TDD): Tfifo(A,E)(t)(d) & RSync(E,B) <=> ∃ (C,D:TDD):
Fifo1(A,D) & SyncD(D,C) & Timert(A,C)(t)(d) & Sync(D,E) &
                          RSync(E,B)
```

The main goal of equivalence relation between R_1 and R_2 is proved with following theorem:

Theorem 3. ∀(A,B:TDD)(t:Time)(d:Data)*:*

```
∃ (E:TDD): RSync(A, E) & Tfifo(E,B)(t)(d) <=>
   ∃ (R:TDD): Tfifo(A,R)(t)(d) & RSync(R,B)
```

Both lemmas are used to prove theorem 3 and the complete proof can be found at [20]. It is important to point out that one of the main limitations of using proof assistants such as PVS is that heavy user intervention is required in the proof development. For a non-trivial theorem, the user does lots of repetitive work to prove the theorem. For example, theorem 3 proof required repeated proof commands to prove the main proof-goal, which is divided later into sub-goals. To avoid this, PVS offers a powerful decision procedures such as *grind* that can be used to complete the proof that does not require induction and only requires the expansion of definitions in the model and reasoning for equality, arithmetic and quantification.

6 Conclusion

The formalization approach for untimed/timed Reo connectors in PVS is extended in this paper to model and reason about probabilistic/random connectors that are constructed from channels with random and probabilistic behavior. Probabilistic/random channels are modeled as relations on *TDD* sequences being observed at the source and sink nodes. Untimed/timed channels specifications are adjusted accordingly from *TD* sequences to *TDD* sequences. The specifications for probabilistic/random channels generated seven TCCs in total. Two TCCs are proved automatically by the prover and five are proved interactively. With formalised compositional operators and channels, complex connectors are modeled and their properties as well as the refinement and equivalence relation between them are proved with the help of PVS proof-commands, inference rules and decision procedures.

For future work, we would like to add more complex probabilistic and stochastic constraints in the connectors and reason about them. We also plan to extend the formalization approach further to deal with hybrid connectors, QoS (Quality of Service) and resource consumption aspects on connectors.

Acknowledgement. The work has been supported by the National Natural Science Foundation of China under grant no. 61772038, 61532019 and 61272160, and the Guandong Science and Technology Department (Grant no. 2018B010107004).

References

1. F. Arbab. The IWIM model for coordination of concurrent activities. In *Proceedings of COORDINATION 1996*, volume 1061 of *LNCS*, pages 34–56. Springer, 1996.
2. F. Arbab. Reo: A channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.
3. F. Arbab, C. Baier, F. S. de Boer, and J. J. M. M. Rutten. Models and temporal logical specifications for timed component connectors. *Software and System Modeling*, 6(1):59–82, 2007.
4. F. Arbab, T. Chothia, R. van der Mei, S. Meng, Y. Moon, and C. Verhoef. From coordination to stochastic models of QoS. In *Proceedings of COORDINATION 2009*, volume 5521 of *LNCS*, pages 268–287. Springer, 2009.
5. C. Baier. Probabilistic models for Reo connector circuits. *Journal of Universal Computer Science*, 11(10):1718–1748, 2005.
6. C. Baier, T. Blechmann, J. Klein, S. Klüppelholz, and W. Leister. Design and verification of systems with exogenous coordination using Vereofy. In *Proceedings of ISoLA 2010*, volume 6416 of *LNCS*, pages 97–111. Springer, 2010.
7. C. Baier, P. Chrszon, C. Dubslaff, J. Klein, and S. Klüppelholz. Energy-utility analysis of probabilistic systems with exogenous coordination. In *It's All About Coordination - Essays to Celebrate the Lifelong Scientific Achievements of Farhad Arbab*, volume 10865 of *LNCS*, pages 38–56. Springer, 2018.
8. C. Baier, M. Sirjani, F. Arbab, and J. Rutten. Modeling component connectors in Reo by constraint automata. *Science of Computer Programming*, 61:75–113, 2006.

9. C. Baier and V. Wolf. Stochastic reasoning about channel-based component connectors. In *Proceedings of COORDINATION 2006*, volume 4038 of *LNCS*, pages 1–15. Springer, 2006.
10. M. Daumas and D. R. Lester. Stochastic formal methods: An application to accuracy of numeric software. In *Proceedings of HICSS, 2007*, page 262. IEEE, 2007.
11. P. R. Halmos. The foundations of probability. *American Mathematical Monthly*, 51(9):493–510, 1944.
12. K. He, H. Hermanns, and Y. Chen. Models of connected things: On priced probabilistic timed Reo. In *Proceedings of COMPSAC 2017*, pages 234–243. IEEE, 2017.
13. W. Hong, M. S. Nawaz, X. Zhang, Y. Li, and M. Sun. Using Coq for formal modeling and verification of timed connectors. In *Proceedings of Software Engineering and Formal Methods: SEFM 2017 Collocated Workshops, Revised Selected Papers*, volume 10729 of *LNCS*, pages 558–573. Springer, 2018.
14. R. Khosravi, M. Sirjani, N. Asoudeh, S. Sahebi, and H. Iravanchi. Modeling and analysis of Reo connectors using Alloy. In *Proceedings of COORDINATION 2008*, volume 5052 of *LNCS*, pages 169–183. Springer, 2008.
15. N. Kokash, C. Krause, and E. de Vink. Reo+mCRL2: A framework for model-checking dataflow in service compositions. *Formal Aspects of Computing*, 24(2):187–216, 2012.
16. Y. Li, X. Zhang, Y. Ji, and M. Sun. A formal framework capturing real-time and stochastic behavior in connectors. *Science of Computer Programming*, 177:19–40, 2019.
17. Y. Moon, A. Silva, C. Krause, and F. Arbab. A compositional semantics for stochastic Reo connectors. In *Proceedings of FOCLASA 2010*, volume 30 of *EPTCS*, pages 93–107, 2010.
18. M. S. Nawaz and M. Sun. Reo2PVS: Formal specification and verification of component connectors. In *Proceedings of SEKE 2018*, pages 391–396. KSI Research Inc., 2018.
19. S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. PVS system Guide, PVS prover Guide, PVS language reference. Technical report, SRI International, November 2001.
20. PVS dump file. Available at: github.com/saqibdola/PReo-PVS/blob/master/preo.
21. M. Sun. Connectors as designs: The time dimension. In *Proceedings of TASE 2012*, pages 201–208. IEEE Computer Society, 2012.
22. M. Sun and F. Arbab. On resource-sensitive timed component connectors. In *Proceedings of FMOODS 2007*, volume 4468 of *LNCS*, pages 301–316. Springer, 2007.
23. M. Sun, F. Arbab, B. K. Aichernig, L. Astefanoaei, F. S. de Boer, and J. Rutten. Connectors as designs: Modeling, refinement and test case generation. *Science of Computer Programming*, 77(7-8):799–822, 2012.
24. M. Sun and X. Zhang. A relational model for probabilistic connectors based on timed data distribution streams. In *Proceedings of FORMATS 2018*, volume 11022 of *LNCS*, pages 125–141. Springer, 2018.
25. X. Zhang, W. Hong, Y. Li, and M. Sun. Reasoning about connectors using Coq and Z3. *Science of Computer Programming*, 170:27–44, 2019.
26. X. Zhang and M. Sun. Towards formal modeling and verification of probabilistic connectors in Coq. In *Proceedings of SEKE, 2018*, pages 385–390. KSI Research Inc., 2018.