2008 Sixth IEEE International Conference on Software Engineering and Formal Methods

Modeling Component Connectors: Synchronisation and Context-Dependency

Mohammad Izadi
LIACS, Leiden University, NL.
Institute for Humanities and Cultural Studies and Sharif University of Technology, Tehran, Iran.
mizadi@licas.nl

Marcello M. Bonsangue
LIACS, Leiden University, NL.
marcello@liacs.nl

Dave Clarke
CWI, Amsterdam, NL
dave@cwi.nl

Abstract

Reo is an exogenous coordination language for component connectors extending data flow networks with synchronization and context-dependent behavior. We propose an operational model of Reo based on Büchi automata in which port synchronization is modeled by records labeling the transitions, whereas context dependencies are stored in the states. We provide a composition operator which models the joining of two connectors, and show that it can be obtained by using two standard operators: alphabet extension and automata product. Our semantics has the advantage over previous models in that it is based on standard automata theory, so that existing theories and tools can be easily reused. Moreover, it is the first formal model addressing all of Reo's features: synchronization, mutual exclusion, hiding, and context-dependency.

1. Introduction

Reo [1] is a coordination language based on connectors for the orchestration of components in a component based systems. Primitive connectors such as synchronous channels or FIFO queues are composed to build circuit-like component connectors which exhibit complex behavior and play the role of glue code in exogenously coordinating the components to produce a system.

Reo generalizes dataflow networks and Khan networks because it allows to express behavior including state-based, context dependent, multi-party synchronization and mutual exclusion. The original description of Reo was purely informal [1] and no natural semantics for it exists. Recently, a number of models have been developed to capture the desired behavior of Reo connectors and of their composition. These include models based on constraint automata [4], timed data streams (also known as abstract behavioral types) [2], connector colouring [9], structural operational semantics [16], linear logic [8] and intentional constraint automata [10]. None of these models, however, is entirely satisfactory. Timed data streams model the possible data flow of a network, but because of their declarative nature they have no support for model checking. All other models are more operational and more suitable for analysis techniques, but either they do not give the desired semantics for certain connectors, or they suffer from technical problems such as not being able to give semantics to all connectors, or both.

Constraint automata are acceptors of timed data streams, but are much more concrete and suitable for model checking analysis. A constraint automaton is labeled transition system in which each transition label contains two parts: a set N of port names that are *synchronized* if the transition is taken and a proposition g on the data. The latter acts as constraint on the data that could be communicated through the ports in N. The data flowing through the ports in N is *mutually exclusive* with respect to any communication by a port not in N.

Two specific shortcomings of the constraint automata model of Reo, for example, are that it cannot model desired fairness constraints and it cannot model operations that depend upon pending I/O operations on the communication ports of a connector. This latter feature is called *context dependency*, which occurs when the behaviour of a connector can change depending upon not only the presence of requests on a connector boundary, but also on their absence. In such cases, the behavior of a connector can change dramatically with changing context. Both connector coloring and intentional constraint automata address the context dependency issue, but connector coloring does not include a description of the temporal unfolding of a Reo connector, and intentional automata suffers from a larger state explosion. Both models are incomplete in that they cannot give semantics to all Reo connectors.

In this paper we introduce a variant of Büchi automata that can express both fairness and context dependency

978-0-7695-3437-4/08 $25.00 © 2008 IEEE
DOI 10.1109/SEFM.2008.24

as well as multi-party synchronization and mutual exclusion. We build on a recent result which shows that every constraint automaton can be translated into an essentially equivalent Büchi automaton [13]. The basic idea is the use of records as data structures for modeling *multi-party synchronization* as well as *mutual exclusion*: ports in the domain of a record are allowed to communicate simultaneously the data assigned to them, while ports not in the domain of the record are blocked so that no communication can happen. Because our model is based on Büchi automata, we can easily express *fairness condition* admitting only executions for which some actions occur infinitely many times [19].

In order to address context-dependent behavior, we extend our model with the possibility of testing if some ports of the environment are ready to communicate or not. That is, we consider a Büchi variant of Kozen's finite automata on guarded strings [15]. In our case, an infinite guarded string is an alternating sequence of sets of *ready* ports and records of *fired* ports (together with their respective data flow). The difficulty in correctly addressing a context dependent behavior is not in its modeling per se, but in its effect when composing different connectors. In fact, as for the combination of synchronous and mutual exclusion constraints, also context dependencies should propagate across a connector. This means that the models of two connectors when composed should agree on both the synchronized and mutually excluded common ports as well as on the tests of the common ports. With this aim, we present a novel definition of a composition operator that generalizes the automata product construction by allowing the alphabets of the two automata to be different.

Our model has the advantage over previous models in that it covers the basic concepts of Reo as well as the context sensitive behavior within a standard automata theoretical framework. The benefits are a clear and easy notation for the representation of a component connector, as well as the efficient existing tool support for automatical analysis.

The remainder of this paper is organized as follows. In Section 2 we present an informal introduction to the Reo calculus of connectors. We continue in Section 3, by introducing Büchi automata on streams of records as a basic model of connectors exhibiting multi-party synchronization, mutual exclusion and fair behavior. In Section 4 we augment our model for component connectors so to take into account of context dependencies. We also introduce a composition operator and show its correctness with respect to the intended semantics by means of several examples. Finally, we show that it can be decomposed into two operators: record extension and ordinary automata product. We conclude in Section 6 with some final thoughts and future research directions.

2. Reo and component connectors

Reo is an exogenous coordination language which is based on a calculus of component connectors [1]. In Reo software components are independent processes which communicate solely through ports. Ports are related by a network of connectors that specifies the glue code. These connectors build together what is called a coordination system.

Reo relies on a very liberal and simple notion of connector. It consists of a set of source, sink and internal nodes, and a user-defined semantics. Data items enter the connector from a component port linked with a source node, while components receive data from the ports connected with the sink nodes of the connector. Reo connectors are composed by conjoining some of their source or sink nodes to form internal nodes. Internal nodes are not accessible from the environment.

A connector may accept the data offered at source nodes by components, or produce data for sink nodes. Component coordination is achieved by delaying or synchronizing those operations. A port that is willing to communicate with a connector end, but that is delayed is said to be *pending*. Figure 1 shows the graphical representation of some basic Reo connectors whose composition allows for expressing a rich set of coordination strategies [1].

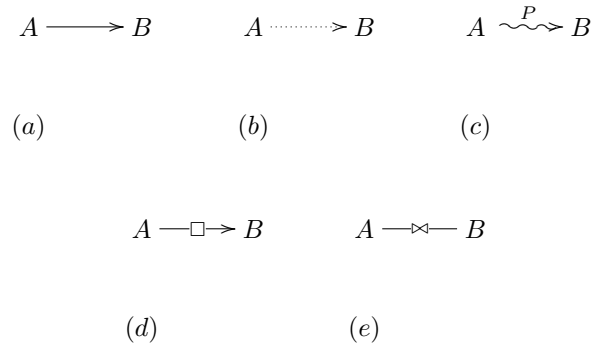


Figure 1. Basic Reo connectors

The *synchronous channel* (Figure 1.a) is a connector with one source and one sink end. It accepts a data item through its source end if and only if it can simultaneously dispense it through its sink.

A *lossy synchronous channel* (Figure 1.b) is similar to synchronous channel, but it never delays the port at the source end. If the port at its sink is pending the channel transfers the data item, otherwise the data item is lost. The behavior of this channel is context-sensitive: it has to be able to sense the absence of a request for data from the port connected at the sink in order for the data received at from the source to get lost.

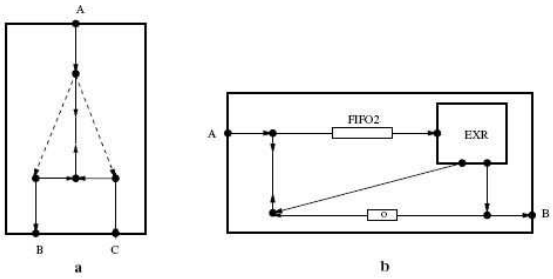


Figure 2. Exclusive router (a) and shift-lossy (b) connectors

A *synchronous filter channel* (Figure 1.c) is a synchronous channels transmitting only data items satisfying a proposition P (the filter). All data items received from the port at the source that do not satisfy P are accepted but lost.

A *FIFO1 channel* (Figure 1.d) is an asynchronous connector. Data from the source is accepted as long as the buffer is empty. The data item received is stored in the channel and communicated to the port at the sink node, when requested. FIFO channels with two or more buffer cells can be produced by composing several FIFO1 channels [4].

Another very useful channel for the design of complex coordination in Reo is the *synchronous drain* (Figure 1.e). It has only two source ends, so that no data value can be produced by the channel. A synchronous drain accepts a data item through one of its ends if and only it is also available simultaneously at the other end as well. The accepted data values are lost.

Figure 2 shows the composition of some of the above channels to implement an exclusive router and a shift-lossy FIFO1 connectors. The intuitive behavior of the exclusive router is that if a data item is communicated at the source node A, the connector delivers it to only one of its sink nodes, either B or C. If both B and C are ready to accept d, the exclusive router nondeterministically chooses one of them . A shift-lossy channel behaves the same as a FIFO1 channel, except that writing to its source end never blocked. If the buffer is full when a new data item arrives, the value stored in the buffer is lost and is replaced by the new data item replaces. For more examples and details of Reo circuits see [1, 5, 4].

3. Modeling port synchronization

In this section we introduce the Büchi automata of records as our basic semantic model for the synchronization among the ports in a network of component connectors. A Büchi automaton of records is an ordinary Büchi automaton with as alphabet a set of *records* assigning port names to data. The latter reflects our assumption that only the data flow among the ports of the connectors is observable. The language accepted by the automaton describes the behavior of a connector in terms of all its possible executions, i.e., infinite sequence of records.

To begin with, let us first recall few definition about records and languages over records. Let $\mathcal{N}$ be a finite nonempty set of port names and $\mathcal{D}$ be a finite nonempty set of data. We write $Rec_{\mathcal{N}}(\mathcal{D}) = \mathcal{N} \rightharpoonup \mathcal{D}$ for the set of *records* with entries from a set of data $\mathcal{D}$ and labels from a set of port names $\mathcal{N}$, consisting of all *partial* functions from $\mathcal{N}$ to $\mathcal{D}$. For a record $r \in Rec_{\mathcal{N}}(\mathcal{D})$ we write $dom(r)$ for the domain of r. Sometimes we use the more explicit notation $r \doteq [n_1 = d_1, \ldots, n_k = d_k]$ for a record $r \in Rec_{\mathcal{N}}(\mathcal{D})$, with $dom(r) = \{n_1, \ldots, n_k\}$ and $r(n_i) = d_i$ for $1 \leq i \leq k$. In case of a singleton data set, we just enumerate the set of port names in the domain of the record. We denote by τ the record with empty domain, that is $dom(\tau) = \emptyset$, and call it the *empty record*.

We use records as data structures for modeling constrained synchronization of ports in $\mathcal{N}$. Following [17], we see a record $r \in Rec_{\mathcal{N}}(\mathcal{D})$ as carrying both positive and negative information: only the ports in the domain of r have the possibility to exchange the data assigned to them by r, while the other ports in $\mathcal{N} \setminus dom(r)$ are definitely constrained to *not* perform any communication. This intuition is formalized by the fact that only for ports $n \in dom(r)$ data can be retrieved, using *record selection* $r.n$. Formally, $r.n$ is just (partial) function application $r(n)$.

Further, positive information may increase by means of the *update* (and extension) operation $r[n: = d]$, defined as the record with domain $dom(r) \cup \{n\}$ mapping the port n to d and remaining invariant with respect to all other ports. The *hiding* operator '$\setminus$' is used to increase the negative information. For $n \in \mathcal{N}$, the record $r \setminus n$ hides the port n to the environment by setting $dom(r \setminus n) = dom(r) \setminus \{n\}$, and $(r \setminus n).m = r.m$.

In this paper we do not distinguish between input and output communication. In fact, a record merely reports the data value exchanged at a port, but not whether it has been received or sent. Clearly, such a distinction can be incorporated by using a data domain that distinguishes between these two types of values.

Definition 1 *Let $r_1 \in Rec_{\mathcal{N}_1}(\mathcal{D})$ and $r_2 \in Rec_{\mathcal{N}_2}(\mathcal{D})$. We say that records r_1 and r_2 are* compatible*, if $dom(r_1) \cap \mathcal{N}_2 = dom(r_2) \cap \mathcal{N}_1$ and $\forall n \in dom(r_1) \cap dom(r_2), r_1.n = r_2.n$. The* union *of compatible records r_1 and r_2, denoted by $r_1 \cup r_2$, is a record over port names $\mathcal{N}_1 \cup \mathcal{N}_2$, such that, $\forall n \in dom(r_1)$, $(r_1 \cup r_2).n = r_1.n$ and $\forall n \in dom(r_2)$, $(r_1 \cup r_2).n = r_2.n$.*

A stream (i.e., an infinite string) of records describes a possible data flow among the ports of a system. Sets of

streams of records are just languages, and as such they can be recognized by ordinary Büchi automata. Next we recall some basic definitions and facts on Büchi automata [20].

Definition 2 *A* Büchi automaton *is a tuple* $B = \langle Q, \Sigma, \longrightarrow, Q_0, F\rangle$ *where,* Q *is a finite set of* states*,* Σ *is a finite nonempty set of symbols called* alphabet*,* $\longrightarrow \subseteq (Q \times \Sigma \times Q)$ *is a* transition relation*,* $Q_0 \subseteq Q$ *is a nonempty set of* initial *states and* $F \subseteq Q$ *is a set of* accepting (final) *states.*

We often write $q \xrightarrow{a} p$ instead of $(q, a, p) \in \longrightarrow$. An *infinite computation* for a stream $\omega = a_0, a_1, \cdots \in \Sigma^\omega$ in B is an infinite sequence $q_0, a_0, q_1, a_1, ...$, of alternating states and alphabet symbols in which $q_0 \in Q_0$ and $q_i \xrightarrow{a_i} q_{i+1}$ for all $i \in \mathbb{N}$. The *language accepted* by a Büchi automaton B consists of all streams $\omega \in \Sigma^\omega$ such that there is an infinite computation for ω in B with at least one of the final states occurring infinitely often. The language of a Büchi automaton B, denoted by $L(B)$, is the set of all streams accepted by it. We say that two Büchi automata B_1 and B_2 are *language equivalent* if $L(B_1) = L(B_2)$.

In this paper we are interested in Büchi automaton whose alphabet is $\Sigma = Rec_{\mathcal{N}}(\mathcal{D})$, for some finite set of port names $\mathcal{N}$ and finite set of data $\mathcal{D}$. We refer to such an automaton as a *Büchi automaton (on streams) of records* abbreviated by BAR.

The intuitive meaning of a BAR as an operational model for Reo connectors is similar to the interpretation of labeled transition systems as formal models for reactive systems. The states represent the configurations of the connector, the transitions the possible one-step behavior where the meaning of a transition

$$q \xrightarrow{r} p$$

is that in configuration q each port $A \in dom(r)$ has the possibility to receive or send the data $r.A$ leading from configuration q to p, while the other ports in $\mathcal{N} \setminus dom(r)$ do not perform any data communication.

For example, the BAR depicted in Figure 6.b shows a model of a synchronous channel between the ports B and C over the data set $D = \{d\}$. It accepts a data item through its port B if and only if it can simultaneously dispense it through its other port C. Note that we use a transition label BC as an abbreviation for the record $[B = d; C = d]$.

In general, Büchi automata of records may contain transitions labeled by τ. These can be considered as internal actions, because no port of the system can be involved in an I/O communication. Since they are externally invisible we may ignore them. However, if we remove all τ symbols from a stream of records ω, the resulting sequence need not to be infinite anymore. For example, removing all τ's from the stream consisting of only τ symbols will result in the empty (and hence finite) string.

Definition 3 *Let* B *be a Büchi automaton of records. The* visible language *of* B *is defined as:*

$$L_{vis}(B) = \{\rho \in Rec_{\mathcal{N}}(\mathcal{D})^\omega \mid \exists \omega \in L(B) : \rho = vis(\omega)\},$$

where $vis(\omega)$ *denote the sequence obtained by removing all* τ *symbols from* ω*. We say that automata* B_1 *and* B_2 *are* visibly equivalent *if* $L_{vis}(B_1) = L_{vis}(B_2)$.

Note that $L_{vis}(B)$ contains only infinite sequences and therefore is a subset of the set of sequences obtained from removing the τ's from the streams in $L(B)$. Clearly, $L_{vis}(B) = L(B)$ if B does not have τ-transitions. By a simple generalization of the standard algorithm for eliminating ϵ-transitions of an ordinary finite automaton over finite words [12], we can construct a Büchi automaton recognizing $L_{vis}(B)$ for a given BAR B.

Lemma 1 *For every BAR* B *there is a BAR* B' *without* τ*-transitions such that,* $L_{vis}(B) = L(B')$

For example, ignoring for the moment the label in the states, the BAR depicted in Figure 6.d shows another model of a synchronous channel between the ports B and C over the singleton data set $D = \{d\}$. As before, it accepts a data item through its port B if and only if it can simultaneously dispense it through its other port C. However, in this model a synchronous channel can have an internal step, representing for example the detection that both ports B and C are ready to communicate from a configuration where this do not holds. Note that the two BAR's are *visibly equivalent*.

3.1 Fair Reo connectors

Because BAR's are ordinary Büchi automata, several kind of fairness requirements on Reo connectors can be easily modeled. We just give few examples here. The automaton in Figure 3.a models a *lossy synchronous* connector between the ports A and B. For the sake of simplicity, we assumed here a singleton data set, so that a record can be safely represented by the ports in its domain only. Intuitively, the lossy synchronous connector accepts a data through its source port A and sends it synchronously to the sink port B, but the data value accepted by A can also get lost. One single state is sufficient to model the behavior, but *all* values can get lost because nothing forbid to take the transition labeled by A for ever.

In Figure 3.b, we show another BAR for the same connector which guarantee not to lose all input data values. Two states are necessary now, but B will receive infinite many values sent by A. In the accepting runs of this model, the transitions with label A cannot be taken consecutively forever. Note, however, that infinitely many values sent by A can still get lost, even if not consecutively.

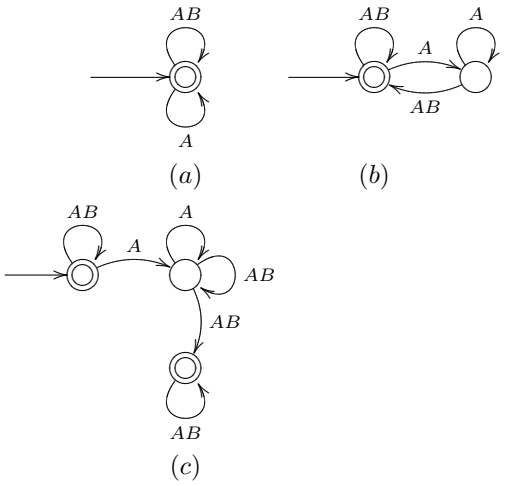


Figure 3. Lossy synchronous channels

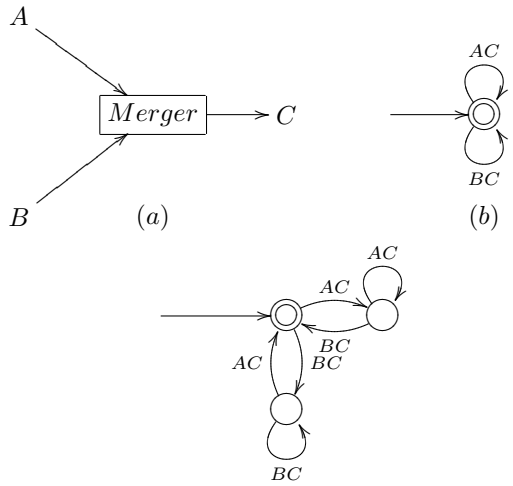


Figure 4. Merger connectors

In Figure 3.c we model a lossy synchronous connector with the strong fairness requirement that only finitely many inputs values can get lost. The automaton needs now three states, but because of the Büchi acceptance condition, the transitions with label A can only be taken a finite number of times.

As another example, consider the *merger* connector among two source ports A and B and one sink port C (Figure 4.a). Intuitively, it transmits synchronously data items from either A or B to the port C. If both the source ports are offering data at the same time then one of them is chosen non-deterministically. The BAR model of a merger over a singleton data set is shown in Figure 4.b. This model allows unfair executions in which data values from the same source is always preferred if both A and B are always offering data values simultaneously. Figure 4.c shows a BAR model that disallows those unfair executions.

3.2. Product, Join, and hiding

Complex component connectors can be obtained by composing simpler ones, and by hiding some ports from the environment. Below we describe these operators on BAR's. We will give few examples in the next section.

Product Since BAR's are ordinary Büchi automata, we can compose them by means of the standard product for Büchi automata, provided they act on the same alphabet. The intuitive meaning of the product is the synchronization of the two component connectors they represent.

Let us to recall the definition of product of Büchi automata which, for simplicity, is given in terms of generalized Büchi automata [20]. A *generalized Büchi automaton* is a Büchi automaton $B = \langle Q, \Sigma, \longrightarrow, Q_0, \mathcal{F}\rangle$ but for the set of final states, that now is a set of sets, that is, $\mathcal{F} \subseteq 2^Q$. A stream $\omega \in \Sigma^\omega$ is accepted by generalized Büchi automaton B if and only if there is an infinite computation π for ω in B such that for every $F \in \mathcal{F}$ at least one of the states in F occurs in π infinitely often.

Definition 4 *Let $B_1 = \langle Q_1, \Sigma, \longrightarrow_1, Q_{01}, F_1\rangle$ and $B_2 = \langle Q_2, \Sigma, \longrightarrow_2, Q_{02}, F_2\rangle$ be two Büchi automata on the same alphabet. The product of B_1 and B_2 is the* generalized *Büchi automaton:*

$$B_1 \times B_2 = \langle Q_1 \times Q_2 . \Sigma, \longrightarrow, Q_{01} \times Q_{02}, \{F_1 \times Q_2, Q_1 \times F_2\}\rangle$$

where the transition relation $\longrightarrow$ is defined by as follows:

$$\frac{q \xrightarrow{a}_1 p \quad s \xrightarrow{a}_2 t}{\langle q, s\rangle \xrightarrow{a} \langle p, t\rangle}.$$

The language of the product of two generalized Büchi automata is the intersection of their respective languages [20].

Every Büchi automaton $\langle Q, \Sigma, \longrightarrow, Q_0, F\rangle$ can be turned into the generalized Büchi automaton $\langle Q, \Sigma, \longrightarrow, Q_0, \{F\}\rangle$ recognizing the same language. However, the product of two such automata is a generalized Büchi automaton. To obtain an ordinary Büchi automaton for the product, one can use the fact that for each generalized Büchi automaton B there is an ordinary Büchi automaton B' such that $L(B) = L(B')$ [20].

Join Using the richer structure of the alphabet of BAR's, we can give a more general definition of product that works even if the alphabets of the two automata are different.

Definition 5 *We define the* join *of two BAR's $B_1 = \langle Q_1, Rec_{\mathcal{N}_1}(\mathcal{D}), \longrightarrow_1, Q_{01}, F_1\rangle$ and $B_2 =$*

$\langle Q_2, Rec_{\mathcal{N}_2}(\mathcal{D}), \longrightarrow_2, Q_{02}, F_2\rangle$ *as the* generalized *Büchi automaton* $B_1 \bowtie B_2$ *given by:*

$$\langle Q_1 \times Q_2, Rec_{\mathcal{N}_1 \cup \mathcal{N}_2}(\mathcal{D}), \longrightarrow, Q_{01} \times Q_{02}, \{F_1 \times Q_2, Q_1 \times F_2\}\rangle$$

where the transition relation $\longrightarrow$ *is defined by the following rules:*

$$\frac{q \xrightarrow{r_1}_1 p \quad s \xrightarrow{r_2}_2 t \quad comp(r_1, r_2)}{\langle q, s\rangle \xrightarrow{r_1 \cup r_2} \langle p, t\rangle},$$

$$\frac{q \xrightarrow{r_1}_1 p \quad dom(r_1) \cap \mathcal{N}_2 = \emptyset}{\langle q, s\rangle \xrightarrow{r_1} \langle p, s\rangle},$$

and dually,

$$\frac{s \xrightarrow{r_2}_2 t \quad dom(r_2) \cap \mathcal{N}_1 = \emptyset}{\langle q, s\rangle \xrightarrow{r_1} \langle q, t\rangle}.$$

Intuitively, two transitions synchronize if they are labeled by compatible records (i.e. on the common ports they communicate the same data values), whereas they interleaves if they are labeled with records not referring to ports of the other automaton.

For BAR's without τ-transitions, the join operator coincides with the product in the case when both automata have the same alphabet.

Lemma 2 *Let* B_1 *and* B_2 *be two BAR's with the same alphabet. Then* $L_{vis}(B_1 \bowtie B_2) = L_{vis}(B_1) \cap L_{vis}(B_2)$.

This implies that our definition of join is correct with respect to the product of ordinary Büchi automata (up to τ-transitions).

Hiding The effect of hiding a port of a component connector is that data flow at that node is no longer observable. In BAR's, the hiding operator removes all information about the hidden port.

Definition 6 *The hiding of a port name* $A \in \mathcal{N}$ *from a BAR* $B = \langle Q, Rec_{\mathcal{N}}(\mathcal{D}), \rightarrow, Q_0, F\rangle$ *is the BAR*

$$B{\downarrow_A} = \langle Q, Rec_{\mathcal{N}\setminus\{A\}}(\mathcal{D}), \longrightarrow' \; Q_0, F\rangle$$

where $q \xrightarrow{r\setminus A}{}' p$ *if and only if* $q \xrightarrow{r} p$.

Note that if the domain of a record labeling a transition contains only the name to be hidden, then the transition becomes an internal one. It is easy to verify that (visibly) language equivalence is a congruence with respect to join and hiding.

4. Modeling context-dependencies

In this section we augment our model for component connectors so to take into account context dependencies like the ones of the lossy synchronous channel: if the port connected at the sink is ready for a sending data but the port at the source it is not ready for receiving it, then the data at the sink is lost. In the previous section we have ignored such a requirement and modeled the loosing of data by means of a (fair) non-deterministic choice with a BAR. In this section, we extend Büchi automata of records with the capability of modeling coordination strategies based on pending and ignored ports. The idea is to enrich the states of a BAR automaton with expressions for testing if the ports at the environment are ready to communicate or not. Intuitively, a transition

$$q \xrightarrow{r} p$$

can be taken only if the ports of the system successfully pass the test associated with a state q. This implies that we must be able to safely eliminate states associated with tests that always fails, and that passing a test has to guarantee that at least as many ports are ready to communicate as needed by every outgoing transitions.

More formally, we consider the set $\mathcal{N}$ of port names as our *primitive test* symbols. We define the set $Exp_{\mathcal{N}}$ of expression for Boolean tests for $\mathcal{N}$ by the grammar

$$e ::= 1 \mid A \mid e.e \mid \bar{e}\,,$$

where $A \in \mathcal{N}$. Given a set $N \subseteq \mathcal{N}$ of ports (ready to communicate) we define when N passes the test e, denoted by $N \models e$, as follows:

$$\begin{array}{lll} N \models 1 & & \\ N \models A & \text{iff} & A \in N \\ N \models e_1.e_2 & \text{iff} & N \models e_1 \text{ and } N \models e_2 \\ N \models \bar{e} & \text{iff} & N \not\models e \end{array}$$

Informally, every collection of port ready to communicate passes the test 1 while every collection of ports ready to communicate containing A passes the primitive test A. The conjunction of two tests e_1 and e_2 is the test $e_1.e_2$, while the negation of a test e is denoted by $\bar{e}$. The other boolean connectives can be defined as derived operators, for instance we define $e_1 + e_2$ as an abbreviation for $\overline{\bar{e_1}.\bar{e_2}}$. We use $\equiv$ to denote the propositional logic equivalence on $Exp_{\mathcal{N}}$.

Given a record $r \in Rec_{\mathcal{N}}(\mathcal{D})$, let $wp(r)$ be the *weakest precondition* for r to be executed. It is defined inductively on the size of $dom(r)$ as the following expression (up to $\equiv$):

$$\begin{array}{l} wp(\tau) = 1 \\ wp(r) = A \wedge wp(r \setminus A) \quad \text{if } A \in dom(r) \end{array}$$

Intuitively, the expression $wp(r)$ is a test checking if all the ports synchronized by r are ready to communicate.

We are now ready to introduce our extension of BAR's for modeling both synchronization and context dependencies.

Definition 7 *An* augmented Büchi automaton of records *(abbreviated by ABAR) is a pair* $\langle B, l\rangle$ *consisting of a BAR* $B = \langle Q, Rec_{\mathcal{N}}(\mathcal{D}), \rightarrow, Q_0, F\rangle$ *and labeling function* $l{:}Q \rightarrow Exp_{\mathcal{N}}$ *such that for all* $q \in Q$*, if* $q \xrightarrow{r} p$ *then* $l(q)$ *implies* $wp(r)$.

As a consequence of the above definition, if $l(q) = \bar{A}$, then all transitions outgoing from q must be internal, i.e., they must be labelled by τ. Similarly, all transitions outgoing from a state labeled by 1 must be internal.

ABAR's are acceptors of infinite guarded strings [14]. In our case, an infinite guarded string over the alphabet $Rec_{\mathcal{N}}(\mathcal{D})$ is an alternating infinite sequence $N_0 r_0 N_1 r_1 \cdots$ where $r_i \in Rec_{\mathcal{N}}(\mathcal{D})$ and the N_i's are subset of ports in $\mathcal{N}$. Intuitively, a guarded string represents an execution of the system, where for each step it records the ports ready for a communication and the actual data flow among a subset of them. More formally, we define an *infinite computation* for a guarded string $N_0 r_0 N_1 r_1 \cdots$ in an ABAR $\langle B, l\rangle$ to be an infinite sequence $q_0, r_0, q_1, r_1, ...$, of alternating states and records in which $q_0 \in Q_0$, $N_i \models l(q_i)$ and $q_i \xrightarrow{r_i} q_{i+1}$ for all $i \in \mathbb{N}$. The *language accepted* by an ABAR $\langle B, l\rangle$ consists of all infinite guarded strings γ such that there is an infinite computation for ω in B with at least one of the final states occurring infinitely often. The language of an ABAR $\langle B, l\rangle$, denoted by $GL(B)$, is the set of all infinite guarded strings accepted by it.

Note that the condition of an ABAR $\langle B, l\rangle$ that for all state q, if $q \xrightarrow{r} p$ then $l(q)$ implies $wp(r)$ means that for every guarded string $N_0 r_0 N_1 r_1 \cdots$ accepted, $dom(r_i) \subseteq N_i$ for all $i \geq 0$.

We say that two ABAR's B_1 and B_2 are *language equivalent* if $GL(B_1) = GL(B_2)$. Given an ABAR $\langle B, l\rangle$ we can construct a language equivalent ABAR $\langle B', l'\rangle$ such that $l'(q) \not\equiv 0$ for all states q of B'. In fact, we can safely delete these inconsistent states from the set of states of B because no set of names N will ever pass the test 0 (not even the empty set of names).

An augmented Büchi automaton of records can be considered as a Büchi automaton of records, if we ignore the labeling function. Conversely, every Büchi automaton of records B can be transformed into a canonical ABAR $\langle B, l\rangle$ by assigning to each state q of B the conjunction of all $wp(r)$ for each record r labeling outgoing transitions from q. Transforming a BAR into an ABAR and back will obtain the same BAR, while the converse holds only for ABAR without states with negative tests. We say that two ABARs $\langle B_1, l_1\rangle$ and $\langle B_2, l_2\rangle$ are *visibly equivalent* if they have no inconsistent state and $L_{vis}(B_1) = L_{vis}(B_2)$.

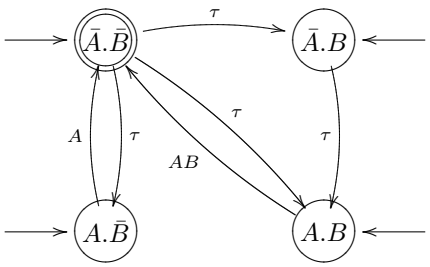


Figure 5. A lossy synchronous channel

Although ABAR's are as expressive as BAR's, in terms of the visible language they recognize, they are more concrete. We will use this extra information when composing them. For the moment let us remark that for an ABAR $\langle B, l\rangle$ we can give a formal definition of pending and ignored ports. Given a set N of ports, we say that $A \in N$ is *ignored* by a transition $q \xrightarrow{r} p$ if $N \models l(q)$ but $A \notin dom(r)$, that is, the port A may be ready to communicate but it is excluded by r. Similarly, we say that a port A is *pending* in a state q if it is ignored by all transitions outgoing from q.

Next we present few examples of component connectors modeled by ABAR's. Figure 5 shows an ABAR model for a lossy synchronous channel over a singleton data domain connecting a port A with a port B. Note that the port A is never pending, while in the upper rightmost state the port B is pending.

In Figure 6 we show two ABAR models of a synchronous channel with source end B and sink C over a singleton data set. The first model (c) is the canonical extension of the BAR model in (b). Note that both ports B and C cannot be pending, whereas in the model in (d) they both can be pending in the initial state. This is of course reflected by the guarded languages they recognize. While the ABAR in (c) accepts only the infinite guarded string

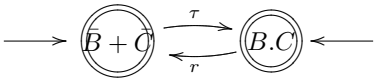
$$\{B, C\}[B = d, C = d]\{B, C\}[B = d, C = d] \cdots ,$$

the other automaton accepts infinitely many strings, including

$$\{\}\tau\{B, C\}[B = d, C = d]\{B\}\tau\{B, C\}[B = d, C = d] \cdots .$$

It is easy to see that the two automata are visibly equivalent.

Also other basic connectors of Reo can be modeled by ABARs. A synchronous drain (and similarly for the synchronous spout) between two ports B and C can be modeled as a synchronous channel, but for the data values passing through the two ports that in this case needs not to be the same:



,

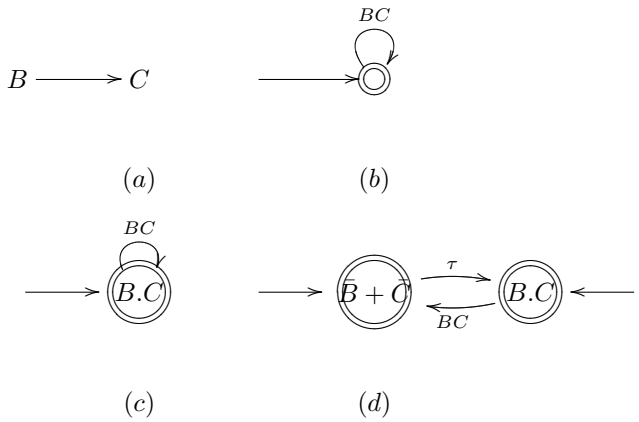


Figure 6. Models of the synchronous channel

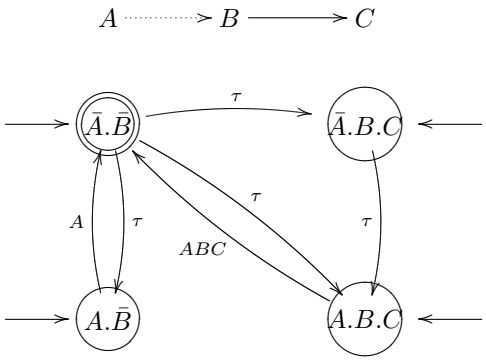
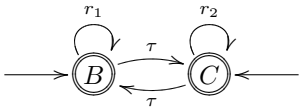


Figure 7. The join of a lossy synchronous channel with a synchronous channel

where $dom(r) = \{B, C\}$. The asynchronous version of a drain between B and C can be modeled by the following ABAR:



where $dom(r_1) = \{B\}$ and $dom(r_2) = \{C\}$.

5. Composing ABAR's

In this section we give a definition of product and join of two ABAR's. We first give a direct definition of the join and then we prove it equivalent to another definition given in term of the product.

Definition 8 *Let $\langle B_1, l_1\rangle$ and $\langle B_2, l_2\rangle$ be two ABAR over the same alphabet, say $Rec_{\mathcal{N}}(\mathcal{D})$. Their* product *is defined as the ABAR $\langle B, l\rangle$, where $B = B_1 \times B_2$ and $l(\langle q, p\rangle) = l_1(q).l_2(p)$.*

Similarly, we define the join of two ABARs in terms of the join of their underlying BAR's.

Definition 9 *Let $\langle B_1, l_1\rangle$ and $\langle B_2, l_2\rangle$ be two ABAR over the same alphabet, say $Rec_{\mathcal{N}}(\mathcal{D})$. Their* join $\langle B_1, l_1\rangle \bowtie \langle B_2, l_2\rangle$ *is defined as the ABAR $\langle B, l\rangle$, where $B = B_1 \bowtie B_2$ and $l(\langle q, p\rangle) = l_1(q).l_2(p)$.*

It is easy to check that the join of ABARs is again an ABAR. In fact, if $q_1 \xrightarrow{r_1} p_1$ is a transition in $\langle B_1, l_1\rangle$ and $dom(r_1)$ has no name in common with those used by another ABAR $\langle B_2, l_2\rangle$, then $l_1(q_1).l_2(q_2)$ implies $wp(r_1)$ for all state q_2 of B_2. Similarly, if $q_2 \xrightarrow{r_2} p_2$ is another transition in $\langle B_2, l_2\rangle$ such that $comp(r_1, r_2)$, then $l_1(q_1).l_2(q_2)$ implies $wp(r_1 \cup r_2)$.

As for BAR's, the join of two ABAR's with the same alphabet coincides with their product. In general, the join operator is not a congruence with respect to the visible equivalence. To see this, it is enough to take two visibly equivalent ABARs with one state labeled in one automaton with $A.B$ and in the other automaton by $A.\bar{B}$. The join of one of them with an automaton with a state labeled by B is different than the join of the other.

Let us now give an example of connector composition. Consider the lossy synchronous channel from a port A to a port B given in Figure 5 and the synchronous channel from B to C as modeled in Figure 6.d. Their join is the ABAR shown in Figure 7. In this figure, we ignored two states with label 0 (false) and two other initial states which have only outgoing transitions with label τ into some other initial states. Note that the resulted automaton is very similar to the model of a lossy synchronous channel between the port A and C, except that we can still observe the data flowing through the port B. After hiding it, the two automata will be language equivalent.

Definition 10 *Let $\langle B, l\rangle$ be an ABAR. The hiding of a port A results in the ABAR $\langle B{\downarrow_A}, l'\rangle$ where $l'(q)$ is the expression $l(q)$ with 1 substituted for A.*

For example, in Figure 8 we consider the composition of a lossy synchronous channel as modeled in Figure 5 with a FIFO1 channel, after hiding the common port B (FIFO1 model is the same as shown in Figure 9.b). Note that in the resulting connector if the buffer of the FIFO1 channel is empty no data value from port A is lost, whereas this happen when the buffer is full.

5.1. Splitting the join

Next we give an alternative way to calculate the join of two Büchi automata of records. The idea is to use the

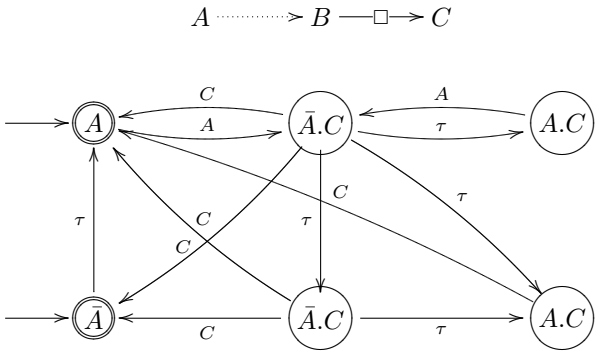


Figure 8. The join of a lossy synchronous channel with a FIFO1 channel

product after we have extended the alphabets of the two automata to a minimal common alphabet. First of all we concentrate on how to extend a Büchi automata of records B with an extra port name, not necessarily present in the alphabet of B. If the port is new, the resulting automata will have to guess the right behavior non-deterministically, by allowing or not the simultaneous exchange of data with the other ports known by the automata.

Definition 11 *Let $B = \langle Q, Rec_{\mathcal{N}}(\mathcal{D}), \longrightarrow, Q_0, F\rangle$ be a Büchi automaton of records and A be a (port) name. We define the extension of B with respect to A as the BAR*

$$B{\uparrow}A = \langle Q, Rec_{\mathcal{N}\cup\{A\}}(\mathcal{D}), \longrightarrow', Q_0, F\rangle\,,$$

where $\longrightarrow'$ is the least relation including $\longrightarrow$ and such that if $A \in \mathcal{N}$ then

1- $q \overset{[A=d]}{\longrightarrow'} q$ for $q \in Q$ and $d \in \mathcal{D}$;

2- $q \overset{r[A:=d]}{\longrightarrow'} p$ for $q \xrightarrow{r} p$ and $d \in \mathcal{D}$.

To extend a BAR B using one extra port name A, we use the same structure of B and add only some new transitions to it representing the guesses of the new behavior of the automaton with respect to the new port A. There are three kinds of guess: the environment does not use the name A in a communication (explaining why $\longrightarrow \subseteq \longrightarrow'$), or the environment use the name n for a communication but no other port of B is used (explaining the addition of a new loop transition on each state labeled by a record with A as the only name in the domain), or the environment use the name n in combination with the name constrained by B (corresponding to the new transitions of the form $q \overset{r[A:=d]}{\longrightarrow'} p$. Recall here that $r[A:=d]$ is the extension of record r by adding the new field $A = d$ to it).

The operation of name extension is not sensible with respect to the order of different applications, in the sense that $(B{\uparrow}A_1){\uparrow}A_2 = (B{\uparrow}A_2){\uparrow}A_1$, for two ports A_1 and A_2. Therefore we can define the extension of a BAR with respect to a finite set of port names N, denoted by $B{\uparrow}N$ by inductively extending the automaton B by one port in N at a time.

Given two Büchi automata of records B_1 and B_2 we can extend each them with respect to the port names of the other, so that they become two Büchi automata over the same alphabet. We can thus take their ordinary product, obtaining as result the join of the two Büchi automata B_1 and B_2.

Theorem 1 *Let $\langle B_1, l_1\rangle$ and $\langle B_2, l_2\rangle$ be two ABAR's over alphabet sets $Rec_{\mathcal{N}_1}(\mathcal{D})$ and $Rec_{\mathcal{N}_2}(\mathcal{D})$ respectively. Then, $\langle B_1{\uparrow}\mathcal{N}_2 \times B_2{\uparrow}\mathcal{N}_1, l'\rangle = \langle B_1, l_1\rangle \bowtie \langle B_2, l_2\rangle$, where $l'(\langle q_1, q_2\rangle) = l_1(q_1) \wedge l_2(q_2)$.*

For example, Figures 9.a and 9.b show the models of two FIFO1 channels (as always in this paper over a singleton data set $\mathcal{D} = \{d\}$). The extension of the first automaton with the port name C is given in (c), while the extension of the second automaton with the port name A is given in (d). Their product is the automaton in (e).

6. Concluding remarks

In this paper, we introduced a simple mathematical model addressing the full semantical features of Reo. Like constraint automata and unlike connector coloring, our model can express the behavior of a connector in term of ports synchronization and exclusion. Like connector coloring and unlike constraint automata our model can express context dependent behavior. Differently from all other models of Reo, our model allows for the specification of fairness constraints.

One of the main benefits of our automata theoretic framework for modeling networks of component connectors comes from the area of model checking. We can use Büchi automata for expressing properties (directly or after translating from linear temporal logics [19, 7]). Existing model checkers for Büchi automata, such as SPIN [11] and NuSMV [6], could be used directly for networks of connectors instead of re-inventing similar tools for constraint automata. It is our plan to investigate this direction in the near future.

References

[1] Arbab F., *Reo: a channel-based coordination model for component composition*, Math. Struc. in Computer Science, **14(3)**, (2004), 329-366.

[2] Arbab F. and Rutten J.J.M.M., *A Coinductive calculus of component connectors*, Lecture Notes in Computer Science, **2755**, Springer-Verlag (2003), 35-56.

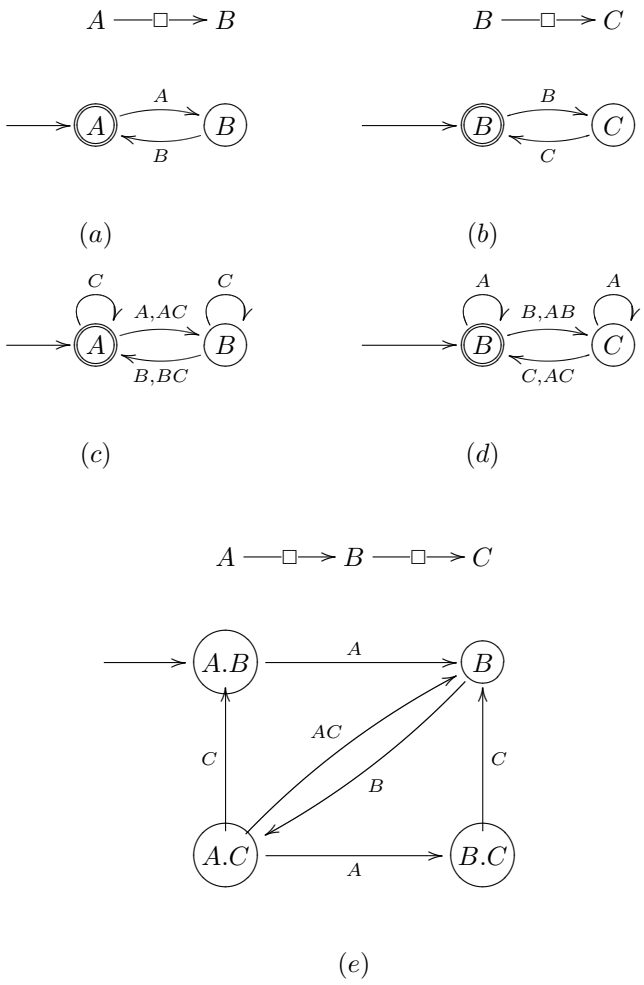


Figure 9. Joining two FIFO1 channels

[3] Arbab, F., Baier, C., de Boer, F., and Rutten, J.J.M.M. *Models and temporal logics for timed component connectors*. In *Proc. of the IEEE International Conference on Software Engineering and Formal Methods (SEFM)*, IEEE Computer Society, pp. 198–207, 2004.

[4] Baier C., Sirjani M., Arbab F., and Rutten J.J.M.M., *Modelling component connectors in Reo by constraint automata*, Science of Computer Programming, **61**, (2006), 75-113.

[5] Arbab F., Baier C., de Boer F., Rutten J., Sirjani M., *Synthesis of Reo circuites for implementation of component-connector automata specifications*, Proceedings of CORDINATION 2005, LNCS, **3454**, Springer-Verlag, (2005), 236-251.

[6] Cimatti A., Clarke E.M., Giunchiglia E., Giunchiglia F., Pistore M., Roveri M., Sebastiani R., and Tacchella A., *NuSMV 2: an open source tool for symbolic model checking*, in Proceedings of the 14th CAV, Springer's LNCS **2404**, (2002), 359-364.

[7] Clarke E., Grumberg O., and Peled D., *Model checking*, The MIT Press, 1999.

[8] Clarke D., *Coordination: Reo, nets, and logic*, unpublished notes, (2008).

[9] Clarke D., Costa D., and Arbab F., *Connector colouring I: synchronisation and context dependency*, Science of Computer Programming, **66(3)**, (2007), 205-225.

[10] Costa D. and Clarke D., *Intensional constraint automata*, unpublished note, 2008.

[11] Holzmann G.J., *The model checker SPIN*, IEEE Transactions on software engineering, **23(5)**, (1997), 279-295.

[12] Hopcroft J.E., Motwani R., and Ullman J.D., *Introduction to automata theory, languages, and computation*, 3rd edition, Addison-Wesley (2006).

[13] Izadi M. and Bonsangue M.M., *Recasting constraint automata into Büchi automata* in Proc. ICTAC 2008, to appear in Springer's LNCS, (2008).

[14] Kaplan D.M., *Regular expressions and the equivalence of programs*, Journal of Compiting System Science, **3**, (1969) 361-386.

[15] Kozen D., *Automata on guarded strings and applications*, Matématica Contemporânea, **24** (2003), 117-139.

[16] Mousavi M.R., Sirjani M., and Arbab F., *Formal sematics and analysis of component connectors in Reo*, in Proc. of FOCLASA 2005, Elsevier's ENTCS **154**, (2005), 83-99.

[17] Remy D., *Efficient representation of extensible records*, Proc. ACM SIGPLAN Workshop on ML and its applications, (1994), 12-16.

[18] Szyperski C., Gruntz D., and Murer S. *Component software: beyond object-oriented programming (second edition)*, Addison-Wesley, 2002.

[19] Vardi M., *An automata-theoretic approach to linear temporal logic*, Lecture Notes in Computer Science, **1043**, Springer-Verlag (1996), 238-266.

[20] Thomas W., *Automata on infinite objects*, J. van Leeuwen (editor), "Handbook of Theoretical Computer Science", **vol. B**, Elsevier, (1990), 133-191.