Büchi automata for modeling component connectors

Article *in* Software and Systems Modeling · May 2011

DOI: 10.1007/s10270-010-0152-1 · Source: DBLP

3 authors:

Mohammad Izadi
Sharif University of Technology
22 PUBLICATIONS **58** CITATIONS
SEE PROFILE

Marcello Bonsangue
Leiden University
178 PUBLICATIONS **1,504** CITATIONS
SEE PROFILE

Dave Clarke
Uppsala University
149 PUBLICATIONS **2,829** CITATIONS
SEE PROFILE

myjournal manuscript No.
(will be inserted by the editor)

Büchi Automata for Modeling Component Connectors

Mohammad Izadi[1,2], **Marcello Bonsangue**[1], **Dave Clarke**[3]

[1] Leiden Institute of Advanced Computer Science (LIACS), Leiden University, The Netherlands
[2] Institute for Humanities and Cultural Studies (IHCS), Tehran, Iran
[3] Dept. of Computer Science, Katholieke Universiteit Leuven, Belgium

Received: date / Revised version: date

Abstract Reo is an exogenous coordination language for component connectors extending data flow networks with synchronization and context-dependent behavior. The first proposed formalism to capture the operational semantics of Reo is called constraint automaton. In this paper, we propose another 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. It is shown that constraint automata can be recast into our proposed Büchi automata of records. Also, 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.

In contrast to many connector languages for components that focus on stateless connectors in a control flow setting (e.g. BIP [8]), 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 [6], timed data streams (also known as abstract behavioral types) [4], connector colouring [13], structural operational semantics [21] and linear logic [12]. 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 a 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 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 modeling Reo by constraint automata, 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 behavior 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 Reo automata [9] address the context dependency issue, but connector coloring does not include a description of the temporal unfolding of a Reo connector, and Reo automata do not address fair behaviours. Both models are incomplete in that they cannot give semantics to many reasonable connectors.

The main contribution of our work presented in this paper is a novel approach to specify the behavior of a network of components. We use records as data structures for modeling the simultaneous executions of events: ports in the domain of the 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. The behavior of a network of components is given in terms of (infinite) sequences of records, so to specify the order of occurrence of the events. Standard operational models can be used to recognize such languages. For example, we use ordinary Büchi automata as operational devices for recognizing languages of streams of records. Because our model is based on Büchi automata, we can easily express the *fairness condition* admitting only executions for which some actions occur infinitely many times [25].

We show that every constraint automaton can be translated into an essentially equivalent Büchi automaton. The construction of the Büchi automaton is straightforward and the result may appear as not surprising at all. But beware! The languages recognized by the two type of automata have a different structure. In fact it is easy to embed a language on streams of records into a language of timed data streams, but not viceversa. Despite these structural differences, we show that the converse also holds without losing any information as far as constraint automata is concerned. An immediate consequence of this result is that, since Büchi automata enjoy closure properties that constraint automata do not have, our model is more expressive. In fact we give few concrete examples of realistic connectors (not considered in the Reo language until now) that can be specified in our model but not with constraint automata.

The main reason for having time information in the timed data streams is compositionality with respect to the Reo join operator. We introduce a join composition operator for Büchi automata on streams of records and show that it is correct with respect to the join operator for constraint automata. Also, we present a method to recast this join operation using the standard product operator of Büchi automata.

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 [19]. 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.

Related Works. Much work has been done on constraint automata for the verification of properties of Reo circuits by model-checking [5], to synthesize Reo circuits and executable code from constraint automata [2], and to automatically compose constraint automata. Further, several extensions of constraint automata have been defined to cover probabilities [7], real-time [5], and other quality of services of connectors [3]. On the other hand, since the definition of constraint automata [6], there has been no expressivity results with respect to existing automata models. In this paper we recast the theory of constraint automata into that of *ordinary* Büchi automata. The latter is especially important because many of the results, tools and extensions of actual interest for Reo and constraint automata have already been developed [14, 26].

In [16] it is shown that every constraint automaton can be translated into an essentially equivalent Büchi automaton. This way synchronization aspects of Reo connectors as well as fairness are taken into consideration. In [17] Büchi automata are augmented so to express also context dependencies. In the present paper we combine the two above mentioned works and complete them with the proofs of the main theorems involved.

We proceed as follows. In Section 2 we give an informal introduction to Reo and its constraint automata model. Constraint automata are acceptor of timed data streams, i.e. indexed families of pairs of infinite words consisting of the data communicated together with its time-stamp. In Section 3 we remark that time-stamps in timed data streams are only needed for defining a temporal order of data communications, and therefore can be mapped into certain kind of infinite words expressive enough to model both synchronization and temporal dependencies. Büchi automata are a natural candidate for accepting languages over infinite words, and therefore we present them ad as an alternative model to constraint automata. We show that the Büchi automata model is strictly more expressive than the constraint automata model of Reo. Further we show how to adapt two automata so that their composition can be expressed in terms of the ordinary product between Büchi automata. In Section 4 we augment our Büchi automata model of Reo with guards for testing if ports at the environment are ready to communicate or not. We introduce a novel composition operation and show, by means of examples, that it behaves as informally explained in [1].

2 Reo and constraint automata

In this section, we briefly introduce the Reo coordination language and its first proposed operational semantics based on constraint automaton.

2.1 Reo coordination language

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

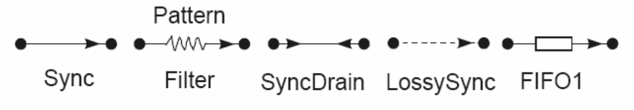


Fig. 1 Some basic Reo channels

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 views the components as computational black boxes. Their internal structure can neither be changed nor any new coordination primitive can be added. A Reo connector *exogenously* orchestrates the data flow between the components and by means of this orchestration it exerts its own coordination strategy. In contrast, there are some other coordination languages and frameworks that exert their coordination strategies *endogenously* which means that they change the internal structure of the components by mens of inserting some primitives into the components' sources [22].

Reo relies on a liberal and simple notion of connector. A connector 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*. Atomic connectors are called *channels*. The notion of channel in Reo is more general than its common interpretation. A channel is a primitive communication medium with exactly two ends, each with its own unique identity. These two ends can be a source and a sink end or both be sources or sinks. Each channel must support a certain set of fully defined primitive operations such as I/O on its ends, synchronization, buffering and so on. More complex connectors can be obtained by composing of the more simpler ones. Thus, in general a connector is a composition of some *channels*. Figure 1 shows the graphical representations of some primitive Reo channels whose composition allows for expressing a rich set of coordination strategies [1].

The *synchronous channel* (Sync) 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* (LossySync) is similar to synchronous channel, but it never delays the port at the source end. If the port at its sink is not ready to accept data, then the data item at the source is lost. In the case when the sink is ready to accept data, the *non-deterministic* lossy synchronous channel may still lose the data item at the source, while the *context-sensitive* lossy synchronous channel will immediately dispense it through its sink. In fact, the latter is able to sense the absence of a request for data from the port connected at the sink in order for the data received from the source to get lost.

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

A *FIFO1 channel* 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 [6].

Another very useful channel for the design of complex coordination in Reo is the *synchronous drain* (SyncDrain). 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.

More complex connectors are constructed in Reo out of the simpler ones by joining them. The *join* of two connectors consists in plugging them together on their common nodes. If a node is connected to the sink of multiple channels, when data flows, it nondeterministically chooses one of them, whereas if it is connected to the source of multiple channels then the data flowing is transferred to all of them provided that all of them can accept it. Thus while a sink node acts as a nondeterministic *merger*, while a source node acts as a *data replicator*. The behaviors on all other ports after a join remain unchanged.

External components are connected only to sink or source nodes, they cannot connect to mixed nodes, i.e. nodes that are source end for one channel and sink end for another one.

Figure 2 shows two examples of complex connectors obtained by the composition of some of the above Reo channels to implement an exclusive router and a shift-lossy FIFO1 connector. Here, we use them only as examples of the appealing visual representations of Reo connectors. The enclosing thick boxes in these figures represent *hiding* which means that the structure of nodes and edges inside the box are hidden to the environment and cannot be changed. The only observable nodes are the nodes at the border of the box which can be used by other entities to interact with the connector. The intuitive behavior of the exclusive router (Figure 2.a) is that if a data item is communicated at the external source node F, the connector delivers it to only one of its sink nodes, either B or E, depending on which of them is ready to accept data. If both B and E are willing to accept data, the exclusive router nondeterministically chooses one of them. More precisely, assume a data item is ready to be sent through port F. If both E and B are not ready to receive data, then F is blocked because the node X cannot deliver the data to all channels having X as source (in fact it is the synchronous drain between X and Z that prevents the data transfer). If B is ready to accept data

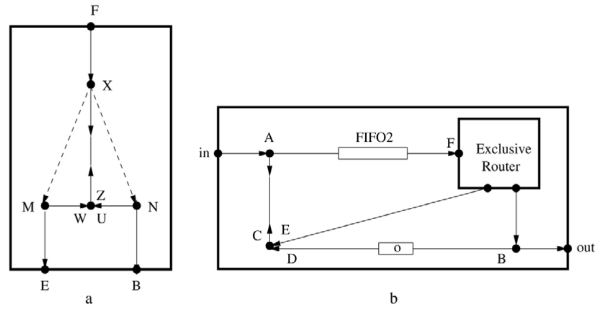


Fig. 2 Exclusive router (a) and shift-lossy (b) connectors

but E is not, then the data at F is replicated by the node X and transferred to N and finally to B. The data copied at X and passing through the lossy synchronous channel between X and M is lost, while the synchronous drain between X and Z forces the (possibly non-deterministic) lossy synchronous channel between X and N to choose for not losing the data item. A similar behavior happen when E is ready to accept but B is not, or when both B and E are ready to accept data.

A shift-lossy channel (Figure 2.b) behaves in the the same way as a FIFO1 channel, except that writing to its source node is 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. We leave to the reader a detailed description of the channel behavior. In Figure 2.b, FIFO2 is a FIFO channel with a buffer capacity of two cells, and the exclusive router is the connector in Figure 2.a. For more examples and a more detailed description of several other Reo circuits we refer to [1,2,6].

2.2 Timed data streams

Constraint automata are a formalism introduced in [6] for describing all possible data flow among the ports of an open components-based software system. For example, a compositional semantics for a large subset of the glue-code language Reo [1] can be given in terms of constraint automata [6]. Next we present the basic theory of constraint automata as acceptors of timed data streams. To begin with, we recall the notion of *time data stream* presented in [4].

Let S be any set. The set of all finite sequences (words) over S is denoted by S^*. We define the set S^ω of all *streams* (infinite sequences) over S as the set of functions $w{:}\mathbb{N} \to S$. For a stream $w \in S^\omega$ we call $w(0)$ the initial value of w. The (stream) *derivative* w' of a stream w is defined as $w'(k) = w(k+1)$. We write $w^{(i)}$ for the i-th derivative of w which is defined by $w^{(0)} = w$ and $w^{(i+1)} = w^{(i)\prime}$. Note that $w^{(i)}(k) = w(i+k)$, for all $k, i \geq 0$. Now, let $\mathcal{N}$ be a fixed finite set of *port names* and $\mathcal{D}$ be a non-empty set of *data* that can be communicated through those ports. The set TDS of all (infinite) *timed data streams* over $\mathcal{D}$ consists of all pairs $\langle \alpha, a \rangle \in \mathcal{D}^\omega \times \mathbb{R}_+^\omega$ such that

1. for all $k \geq 0$ either $a(k) = \infty$ or $a(k) < a(k+1)$, and
2. $\lim_{k\to\infty} a(k) = \infty$.

where $\mathbb{R}_+ = [0, \infty]$ is the set of all positive real numbers including zero and infinity. Informally, a timed data stream contains a pair of streams: a stream of data items called *data stream* together with a stream, say *time stamp*, consisting of strictly increasing positive real numbers converging to infinity. The time stamp indicates for each data item $\alpha(k)$ the moment $a(k)$ at which it is communicated. Data item $\alpha(k)$ with time stamp $a(k) = \infty$ are, by convention, never communicated. Note that, by definition, if $a(k) = \infty$ then also $a(i) = \infty$ for all $i \geq k$.

Let $\mathcal{N}$ be the set of all ports of a coordination system. With each port $A \in \mathcal{N}$, we associate a timed data stream recording both the data communicated and the time when the communication happens. Thus, we can define a *TDS-tuple* as a indexed family with $|\mathcal{N}|$ elements such that each element is a timed data stream recording the communications for one of the ports in $\mathcal{N}$. Now, let $TDS^{\mathcal{N}}$ be the set of all TDS-tuples consisting of one timed data stream for each port in $\mathcal{N}$. We use a family-notation $\theta = (\theta|_A)_{A\in\mathcal{N}}$ for the elements of $TDS^{\mathcal{N}}$ (i.e. functions from $\mathcal{N}$ to TDS), where $\theta|_A$ stands for the application of θ along the port A. Thus, $\theta|_A$ is the timed data stream belongs to port A in TDS-tuple θ. A *TDS-language* for $\mathcal{N}$ is any subset of $TDS^{\mathcal{N}}$.

Simultaneous exchange of data between a set of ports can be detected by inspecting the time when communication happens. For this purpose, for $\theta \in TDS^{\mathcal{N}}$ we define $\theta.time$ to be a stream in $\mathbb{R}_+^\omega$ obtained by merging the "time stamps" streams in $\theta|_A$ associated to each name A in $\mathcal{N}$ in increasing order. More formally, for every timed data stream $t = \langle \alpha, a \rangle$ we define $\pi_l(t) = \alpha$ and $\pi_r(t) = a$. Let θ be TDS-tuple. The time stream $\theta.time$ is defined as follows:

$$\begin{aligned}
&\theta.time(0) = min\{\pi_r(\theta|_A)(0) \mid A \in \mathcal{N}\},\\
&\theta.time(i+1) = min\{\pi_r(\theta|_A)(k) \mid \pi_r(\theta|_A)(k) > \theta.time(i),\\
&\qquad\qquad k \in \mathbb{N}, A \in \mathcal{N}\},
\end{aligned}$$

where we assume that $min\, \emptyset = \infty$ (similar to the empty conjunction being the top element in a complete lattice).

Next we define the stream $\theta.N$ over $2^{\mathcal{N}}$ by setting, for each $k \in \mathbb{N}$, the set $\theta.N(k)$ of *active ports* at time $\theta.time(k)$ to be

$$\{A \mid A \in \mathcal{N}, \exists i \in \mathbb{N}{:}\pi_r(\theta|_A)(i) = \theta.time(k) < \infty\}.$$

Intuitively, the above set consists of all the ports exchanging a data item at time $\theta.time(k)$. The restriction of $\theta.time(k) < \infty$ is to ensure that we consider only data that is communicated within finite time.

We denote the data communicated by a port $A \in \theta.N(k)$ by

$$\theta.\delta(k)_A = \pi_l(\theta|_A)(i)$$

for the unique index i such that $\pi_r(\theta|_A)(i) = \theta.time(k)$.

Timed data streams do not distinguish between input and output actions. They only report the data *observed* at the ports.

In addition, because timed data streams consist of two *infinite* data and time streams, we can only describe finite sequence by saturating them with data communication that cannot happen, i.e. happening at time stamp ∞. In the remainder of this paper we will only consider TDS-tuples θ such that $\theta.time(k) \neq \infty$ for all $k \in \mathbb{N}$. Since, by convention, data with ∞ as time stamp is never communicated, the above restriction implies that we will consider only TDS-tuples that have an infinite number of actual communications.

2.3 Constraint automata and their composition

A constraint automaton is a labeled transition system in which each transition label contains two parts: a finite set N of port names which is a subset of the set of all ports $\mathcal{N}$, and a proposition g on the data. Both parts act as constraints: the set of ports $N \subseteq \mathcal{N}$ constrains which ports of the system should be active if the transition is taken, whereas the proposition g constrains the data that could be communicated through the ports in N.

The set $DC(\mathcal{N}, \mathcal{D})$ of *data constraints* over a finite set N of port names and a finite set $\mathcal{D}$ of data is defined by the following grammar:

$$g \ ::= \ true \mid d_A = d \mid g_1 \vee g_2 \mid \neg g \qquad d \in \mathcal{D}, \ \ A \in N \,.$$

Now we can define the notion of constraint automaton formally.

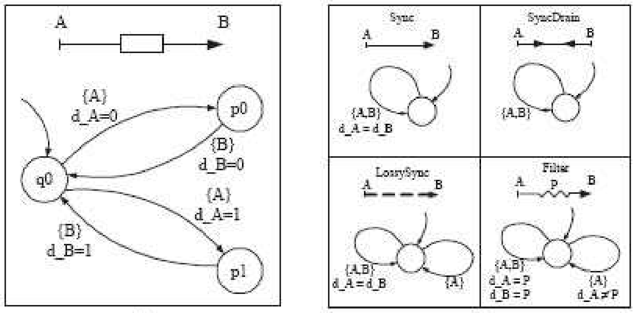
Definition 1 *A* constraint automaton *over finite data set $\mathcal{D}$ is a quadruple $A = \langle Q, \mathcal{N}, \longrightarrow, Q_0 \rangle$ where, Q is a set of states, $\mathcal{N}$ is a finite set of names, $\longrightarrow \subseteq Q \times 2^{\mathcal{N}} \times DC(\mathcal{N}, \mathcal{D}) \times Q$ is a set of transitions and $Q_0 \subseteq Q$ is a set of initial states.*

We write $p \xrightarrow{N,g} q$ instead of $(p, N, g, q) \in \longrightarrow$ and call N the name set and g the guard of the transition. For every transition label (N, g), it is required that $N \neq \emptyset$ and $g \in DC(N, Data)$.

A constraint automaton is said to be *finite* if its sets of states and transitions are finite, and to be *deterministic* if its set of initial states Q_0 is a singleton and for each state q, set of port names N and data assignment $\delta{:}N \to \mathcal{D}$ there is at most one transition $q \xrightarrow{N,g} q'$ with $\delta \models g$.

Example 1 Figure 3 shows the constraint automata models of the set of Reo channels which we introduced in the previous section [6]. Figure 3.a is the constraint automaton model of a FIFO1 channel from a source node A to sink B over the data set $\{0, 1\}$. Figure 3.b shows the constraint automata models of the remaining channels with the consideration that $d_A = d_B$ is an abbreviation for $\vee_{d \in \mathcal{D}}(d_A = d \wedge d_B = d)$ where $\mathcal{D}$ is the set of data $\{0, 1\}$. In the case of the filter channel it must that $P \in \mathcal{D}$.

Like ordinary automata are acceptors of finite strings, constraint automata are acceptors of tuples of timed data streams. Informally, each element of the tuple is associated to a port of the system and corresponds to the streams of observed data communicated through this port together with the time when the data has been observed.



(a) (b)

Fig. 3 Constraint automata models of some Reo channel

Definition 2 *Let $A = \langle Q, \mathcal{N}, \longrightarrow, Q_0 \rangle$ be a constraint automaton and $\theta \in TDS^{\mathcal{N}}$ be a TDS-tuple.*

(i) An infinite computation *for θ in A is an infinite sequence of alternating states and transition labels such as $\pi = q_0, (N_0, g_0), q_1, (N_1, g_1), ...$, in which, for all i, $q_i \in Q$ and $q_i \xrightarrow{N_i, g_i} q_{i+1}$ such that $N_i = \theta.N(i)$, $\theta.\delta(i) \models g_i$. Also, π is an* initial infinite computation, *if $q_0 \in Q_0$.*

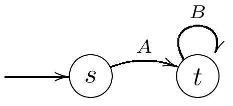
(ii) A TDS-tuple θ is accepted *by A if and only if there is an initial infinite computation for θ in A. The language of constraint automaton A is*

$$\mathcal{L}(A) = \{\theta \in TDS^{\mathcal{N}} \mid A \text{ accepts } \theta\}.$$

The above definition of $\mathcal{L}(A)$ can also be formally defined by means of the greatest fixed point of a suitably chosen monotone operator [6]. For the purpose of this paper we found it easier to reason with the accepted language characterized by means of the (standard) notion of accepted computations.

Using the Rabin-Scott powerset construction as for finite automata, it is easy to see that for every constraint automaton A there is a deterministic constraint automaton A' such that $\mathcal{L}(A) = \mathcal{L}(A')$ [6]. Further, in a finite constraint automaton A all transitions with unsatisfiable guards can be removed without any effect on the TDS language accepted by A, where a guard g of a transition $q \xrightarrow{N,g} q'$ is said to be semantically unsatisfiable for N if there is no data assignment for elements of N which satisfies g (take, for example, g to be $\neg true$). In the rest of this paper we assume without any loss of generality that all guards in a constraint automaton are satisfiable with respect to the set of names of the transition they belongs to.

Differently from finite and Büchi automata on languages, the simplicity of a constraint automaton is not reflected in the TDS language it recognizes. Consider for example the following constraint automaton on two ports A and B over a singleton data set:



While the automaton describes only a *single* event happening at port A, a TDS-tuple θ accepted by the automaton consists a pair of two *infinite* sequence of events θ_A and θ_B, one describing the data flow at port A and the other the flow at port B, such that all events in θ_B happen between the first and the second event in θ_A. All events but the first in θ_A are not really relevant, yet one needs to describe them all.

Constraint automata can be composed by means of a join operator, the semantic counterpart of the join operator in Reo [6]. Differently from the ordinary product for finite automata, the composition of two constraint automata is allowed even if they have different alphabets. In fact, the resulting constraint automaton has transitions when data occur at the ports belonging to only one of the automaton, without involving the transitions or states that it inherits from the other automaton (because at that point in time, there is no data on any of its corresponding ports). More formally, the join operation for constraint automata is defined as follows:

Definition 3 *Suppose that* $A_1 = \langle Q_1, \mathcal{N}_1, \longrightarrow_1, Q_{01}\rangle$ *and* $A_2 = \langle Q_2, \mathcal{N}_2, \longrightarrow_2, Q_{02}\rangle$ *are two constraint automata both over data set* $\mathcal{D}$. *The* join *of* A_1 *and* A_2 *produces a constraint automaton*

$$A_1 \bowtie_C A_2 = \langle Q_1 \times Q_2, \mathcal{N}_1 \cup \mathcal{N}_2, \longrightarrow, Q_{01} \times Q_{02}\rangle$$

in which transition relation $\longrightarrow$ *is defined using the follow rules:*

$$\frac{q \xrightarrow{N_1,g_1}_1 q', p \xrightarrow{N_2,g_2}_2 p', N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1}{\langle q,p\rangle \xrightarrow{N_1 \cup N_2, g_1 \wedge g_2} \langle q',p'\rangle},$$

$$\frac{q \xrightarrow{N_1,g_1}_1 q', N_1 \cap \mathcal{N}_2 = \emptyset}{\langle q,p\rangle \xrightarrow{N_1,g_1} \langle q',p\rangle}, \qquad \frac{p \xrightarrow{N_2,g_2}_1 p', N_2 \cap \mathcal{N}_1 = \emptyset}{\langle q,p\rangle \xrightarrow{N_2,g_2} \langle q,p'\rangle}.$$

Basically, the two automata have to agree on the data exchanged on the common ports (that is, the names used in the transition of the first automaton known to the second automaton are exactly the same as the names used by the transition of the second automaton known to the first one), and each maintains its own behavior on the other ports (as described by the last two rules).

The join of two automata is the realization at the automata-level of the join at the Reo connector level of the source nodes of a connector with the common nodes of the other. The joining of sink nodes can be modeled by means of the automata representing a merger connector as given in Figure 9.b. The join of two constraint automata using the operation defined in Definition 3 is correct with respect to the join of their accepted TDS-languages, where the join of two TDS-languages is basically the same as defined in the theory of relational databases [6]: the projection on the common indexes (port names) of the resulting language must agree with that of the two original languages, while the projection on each indexes in one language but not in the other must agree with the projection on the same index of the language where the index belongs to.

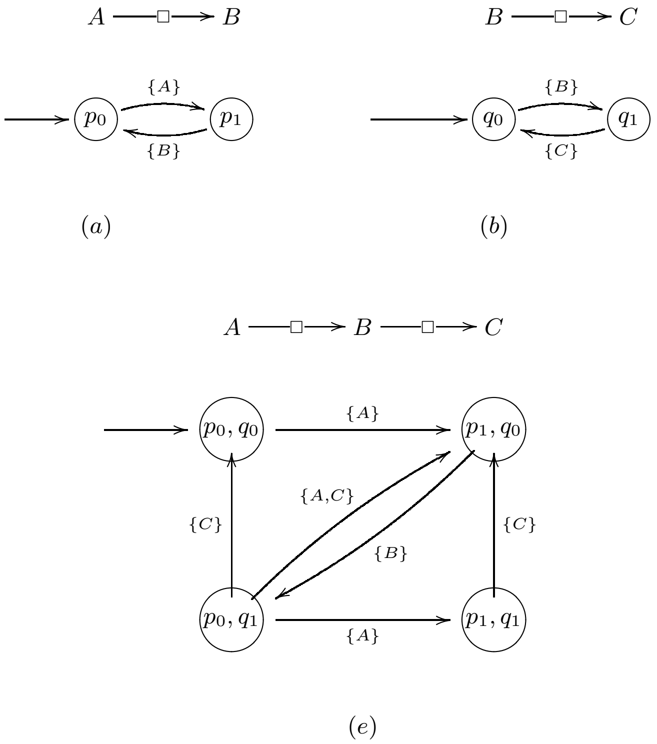


Fig. 4 Joining of constraint automata models of two FIFO1 channels

Example 2 Let us join the constraint automata representing two FIFO1 channels, one from source A to sink B and the other from source B to sink C. The automata models of the two channels and the resulted automaton are illustrated in Figure 4. For simplicity we assumed that the data set is $\mathcal{D} = \{d\}$. Thus, every transition label contains only the set of port names that participate in firing the transition.

Constraint automata are, in general, not closed under complement. Informally this is due to the fact that constraint automata do not have ”final” states. If we augment the definition of constraint automaton by a set of final states and use Büchi acceptance condition (a timed data stream is accepted if at least one of the correspondent runs for it contains one of the final states infinitely many times), we refer to the resulting automaton as a *Büchi constraint automaton*. Obviously, a constraint automaton is a Büchi constraint automaton in which all states are accepting. Its complement, however does not need to satisfy this property.

3 Büchi automata of record languages

Constraint automata are acceptors of timed data streams. However, timed data streams are much more concrete than constraint automata because they record the actual times when communications happen, whereas constraint automata record just the temporal order of data communications and not their time stamps.

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.

3.1 Streams and languages of records

Next we introduce an alternative way to model temporal ordering of data occurrences using streams of records. After introducing the notions of record and record-based languages, we introduce a bidirectional translation between TDS-languages and record-based languages.

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 $1 \leq i \leq k$. Differently from a tuple, the order of the components of a record is irrelevant and its size is not fixed a priori. We denote by τ the special record with empty domain, that is $dom(\tau) = \emptyset$.

We use records as data structures for modeling constrained synchronization of ports in $\mathcal{N}$. Following [23], 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 '\' 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 $(r \setminus n).m = r.m$.

Definition 4 *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 $\forall n \in dom(r_2){:}(r_1 \cup r_2).n = r_2.n$.*

Let us compare the expressiveness of TDS-languages with that of languages of streams of records. Given a TDS-language L for $\mathcal{N}$ we can abstract from its timing information to obtain a set of streams over $Rec_{\mathcal{N}}(\mathcal{D})$. For a TDS-tuple $\theta \in TDS^{\mathcal{N}}$, the idea is to construct a stream of records $\Upsilon(\theta) \in Rec_{\mathcal{N}}(\mathcal{D})^{\omega}$, where, for each k, the record $\Upsilon(\theta)(k)$ contains all ports and data exchanged at time $\theta.time(k)$, that here we assume to be always different from ∞. In fact, we define for each $n \in \theta.N(k)$ and $k \in \mathbb{N}$,

$$\Upsilon(\theta)(k).n = \theta.\delta(k)_n .$$

Note that $dom(\Upsilon(\theta)(k)) = \theta.N(k)$. As usual, we extend this construction to sets, namely, for every $L \subseteq TDS^{\mathcal{N}}$,

$$\Upsilon(L) = \bigcup \{\Upsilon(\theta) \mid \theta \in L\} .$$

Example 3 Let

A	d	d'	d”	...
	0.5	0.7	1.9	

B	d	d'	...
	0.5	1.2	

be a TDS-tuple over port set $\{A, B\}$. Then,

$$\rho = [A = d, B = d][A = d'][B = d'][A = d"] \cdots$$

is its correspondent stream of records. The time stamp are used only to determine the ordering of data communications.

Conversely, any stream of records $\rho \in Rec_{\mathcal{N}}(\mathcal{D})^{\omega}$ generates a TDS-language $\Theta(\rho)$ by *guessing* the time when data is exchanged so to respect the relative order of communication imposed by ρ. Formally,

$$\Theta(\rho) = \{\theta \in TDS^{\mathcal{N}} \mid \forall k \geq 0{:}(\theta.N(k) = dom(\rho(k)) \text{ and } \forall n \in dom(\rho(k)){:}\theta.\delta(k)_n = \rho(k).n)\} .$$

For example, for ρ being the stream of records as in the example above, the following TDS-tuple

A	d	d'	d”	...
	1	10.4	23.6	

B	d	d'	...
	1	10.5	

is in the language $\Theta(\rho)$. Clearly, also the TDS-tuple in Example 3 is an element of the same language.

We extend Θ to languages $L \subseteq Rec_{\mathcal{N}}(\mathcal{D})^{\omega}$ by setting

$$\Theta(L) = \bigcup \{\Theta(\rho) \mid \rho \in L\}.$$

The function $\Theta{:}2^{Rec_{\mathcal{N}}(\mathcal{D})^{\omega}} \to 2^{TDS^{\mathcal{N}}}$ is an embedding of languages over records into TDS-languages for $\mathcal{N}$.

Lemma 1 *For each $L \subseteq Rec_{\mathcal{N}}(\mathcal{D})^{\omega}$, $L = \Upsilon(\Theta(L))$.*

The counterpart of the above lemma for TDS-languages does not hold, because a tuple of time data streams $\theta \in TDS^{\mathcal{N}}$ contains specific time information that get lost when mapped into a stream of record $\Upsilon(\theta)$. In the next section we will see that for constraint automata the information lost in the above translation is never used.

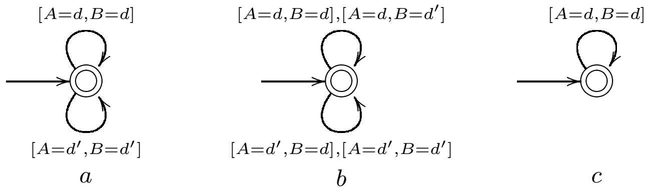

[$A{=}d,B{=}d$]
[$A{=}d',B{=}d'$]
a
[$A{=}d,B{=}d$],[$A{=}d,B{=}d'$]
[$A{=}d',B{=}d$],[$A{=}d',B{=}d'$]
b
[$A{=}d,B{=}d$]
c

Fig. 5 BAR models of some Reo channels: a) Sync, b) SyncDrain, c) Filter.

3.2 Büchi automata of records

Sets of streams of records are just languages of infinite strings, and as such some of them can be recognized by ordinary Büchi automata. Next we recall some basic definitions and facts on Büchi automata [26].

Definition 5 *A* Büchi automaton *B is a tuple $\langle Q, \Sigma, \Delta, Q_0, F\rangle$ where, Q is a finite set of* states, *Σ is a finite nonempty set of symbols called* alphabet, *$\Delta \subseteq (Q \times \Sigma \times Q)$ 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.*

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 $(q_i, a_i, q_{i+1}) \in \Delta$ for all i. 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 automata with as alphabet a subset of $Rec_{\mathcal{N}}(\mathcal{D})$, 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).

Example 4 In Figure 5 we show BAR models of some basic Reo channels. We assume that all channels are from port A to port B and the data set is $\mathcal{D} = \{d, d'\}$. Sometimes instead of drawing separate loops on the same vertex, we draw one loop with several labels separated by commas. For the case of filter we assume that the filter value is d.

In general, a Büchi automaton of records may contain transitions labeled by τ. These can be considered as internal actions, as no port of the system can be involved in a communication. Similar to weak bisimulation, we may ignore internal steps, thus considering two systems equivalent only with respect to their sequences of externally visible actions.

Definition 6 *Let B be a Büchi automaton of records. The* visible language *of B is defined as:*

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

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

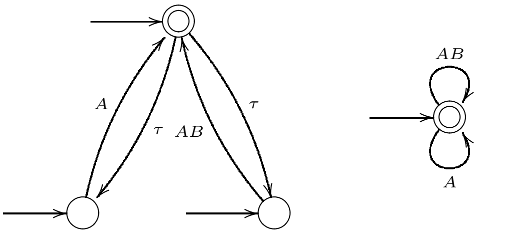


Fig. 6 Two visibly equivalent Büchi automata of records.

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 $L(B)$. For example, if $L(B) = \{[A = d] \cdot [A' = d'] \cdot \tau^\omega\}$, then $L_{vis}(B) = \emptyset$, because removing all τ's from a stream consisting of infinitely many τ's will result in a finite string, and thus not in $Rec_{\mathcal{N}}(\mathcal{D})^\omega$. Clearly, $L_{vis}(B) = L(B)$ if B does not have τ-transitions.

Example 5 In Figure 6 two visibly equivalent BAR models are illustrated. To simplify the figure, we use a singleton data set $\mathcal{D} = \{d\}$ and denote a record labeling a transition only by the domain where it is defined.

By a simple generalization of the standard algorithm for eliminating the ϵ-transitions of an ordinary finite automaton over finite words [15], we can construct a Büchi automaton recognizing $L_{vis}(B)$.

Lemma 2 *For every Büchi automaton of records B there is a Büchi automaton of records B' (without τ-transition) such that, $L_{vis}(B) = L(B')$*

3.3 Recasting constraint automata into Büchi automata

Now we show that for every constraint automaton A over name set $\mathcal{N}$ and data set $\mathcal{D}$ we can construct a Büchi automaton of records. The key observation is that for each transition labeled (N, g) in A, there is a set of (total) data assignments $\{\delta{:}N \to \mathcal{D} \mid \delta \models g\}$. Every data assignment in this set can be seen as a partial function from $\mathcal{N}$ to $\mathcal{D}$, with as domain $N \subseteq \mathcal{N}$, that is, it is a record in $Rec_{\mathcal{N}}(\mathcal{D})$. We can thus construct a Büchi automaton of records $B(A)$ with the same (initial) states as A, with all states as final, and with transitions labeled by each of the above data assignment for every transition in A.

Definition 7 *Let $A = \langle Q, \mathcal{N}, \longrightarrow, Q_0\rangle$ be a constraint automaton over finite data set $\mathcal{D}$ and finite name set $\mathcal{N}$. We define $B(A) = \langle Q, Rec_{\mathcal{N}}(\mathcal{D}), \Delta, Q_0, F\rangle$ to be a Büchi automaton of records, where, $F = Q$ and,*

$$\Delta = \{(q, r, q') \mid \exists q \xrightarrow{(N,g)} q', \exists \delta{:}N \to \mathcal{D}{:} \\ \delta \models g, dom(r) = N \, \textit{and} \, \forall n \in N{:}r.n = \delta(n)\}.$$

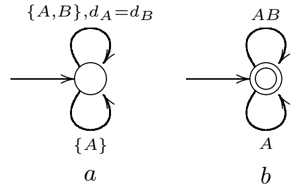


Fig. 7 Models of a *non-deterministic* lossy synchronous channel by a) a constraint automaton and b) a Büchi automaton of records.

Example 6 Consider the constraint automaton depicted in Figure 7(a). It models a non-deterministic lossy synchronous channel from the source A to the sink B: data in $\mathcal{D}$ either flows from A to B or it gets lost after it is read by A [6]. Figure 7(b) shows the corresponding Büchi automaton on streams of records. Again, to simplify the figure, we use a singleton data set $\mathcal{D} = \{d\}$ and denote records only by the domains where they are defined.

All Büchi automata of records in Figure 5 are obtained as the translation of the constraint automata models of the same channels in Figure 3. Note that in Figure 5 the data set is $\mathcal{D} = \{d, d'\}$.

The following theorem shows that timed data streams are not different from streams of records, at least as far as finite constraint automata are concerned.

Theorem 1 *Let $A = \langle Q, \mathcal{N}, \longrightarrow, Q_0\rangle$ be a finite constraint automaton then,*

$$\Upsilon(\mathcal{L}(A)) = L(B(A)) \textit{ and } \Theta(L(B(A))) = \mathcal{L}(A).$$

Proof We start by proving the leftmost equality. Let $r = r_0, r_1, \cdots$ be a stream of records in $L(B(A)) \subseteq Rec_{\mathcal{N}}(\mathcal{D})^\omega$. Because $B(A)$ is a Büchi automaton such that all its states are final, there is an infinite computation $\pi = q_0, r_0, q_1, r_1, \cdots$ in $B(A)$, starting from an initial state q_0 and such that each tuple (q_i, r_i, q_{i+1}) is a transition in $B(A)$. By construction of for each transition (q_i, r_i, q_{i+1}) in the Büchi automaton $B(A)$, there is a transition (q_i, N_i, g_i, q_{i+1}) in the constraint automaton A, with a data assignment $\delta_i{:}N_i \rightarrow \mathcal{D}$ such that $\delta \models g_i$ and $\forall n \in N_i, r.n = \delta(n)$. This implies that the stream $\pi' = q_0, (N_0, g_0), q_1, (N_1, g_1), \cdots$ is an infinite computation in constraint automaton A and that for all TDS-tuple $\theta \in TDS^{\mathcal{N}}$ with $r = \Upsilon(\theta)$ it holds that $\theta.N(i) = N_i$ and $\theta.\delta(i) \models g_i$, for all $i \geq 0$. Thus, $r \in \Upsilon(\mathcal{L}(A))$ and $L(B(A)) \subseteq \Upsilon(\mathcal{L}(A))$.

Conversely, let $r = r_0, r_1, \cdots$ be a stream of records in $\Upsilon(\mathcal{L}(A))$. Then there is a TDS-tuple $\theta \in L_{TDS}(A)$ such that $r = \Upsilon(\theta)$ and for each $n \in \theta.N(k)$ and $k \in \mathbb{N}$, $r(k).n = \theta.\delta(k)_n$. Because $\theta \in \mathcal{L}(A)$, there is an infinite computation $\pi = q_0, (N_0, g_0), q_1, (N_1, g_1), \cdots$ in the constraint automaton A, starting from an intial state q_0 and such that $\theta.N(i) = N(i)$ and $\theta.\delta(i) \models g_i$, for all $i \geq 0$. By construction, there is an infinite computation $\pi = q_0, r_0, q_1, r_1, \cdots$ in $B(A)$ and data assignments $\delta_i{:}N \rightarrow \mathcal{D}$ such that, for all $i \geq 0$, $\delta_i \models g_i$ and $r_i.n = \delta_i(n)$. Since in $B(A)$ all infinite runs starting from an initial state are accepting, $r \in L(B(A))$, and hence $\Upsilon(\mathcal{L}(A)) \subseteq L(B(A))$. Next we prove the rightmost equality. Let $\theta \in TDS^{\mathcal{N}}$ be a timed data stream accepted by the constraint automata A, that is $\theta \in \mathcal{L}(A)$. By definition of acceptance there exists an infinite computation $\pi = q_0, (N_0, g_0), q_1, (N_1, g_1), \cdots$ in A such that, $q_0 \in Q_0$ and, for all $i \geq 0$, $(q_i, (N_i, g_i), q_{i+1})$ is a transition in A, $N_i = \theta.N(i)$, and $\theta.\delta(i) \models g_i$. But then, by construction, there is an infinite computation $\pi' = q_0, r_0, q_1, r_1, \cdots$ in the Büchi automaton $B(A)$ such that for all $i \geq 0$, there is a data assignment $\delta_i{:}N \rightarrow \mathcal{D}$ such that $\delta_i \models g$ and $\forall n \in N, r_i.n = \delta(n)$. Thus, $r = r_0, r_1, \cdots \in L(B(A))$ and $\theta = \Theta(r)$. Therefore, $\mathcal{L}(A) \subseteq \Theta(L(B(A)))$.

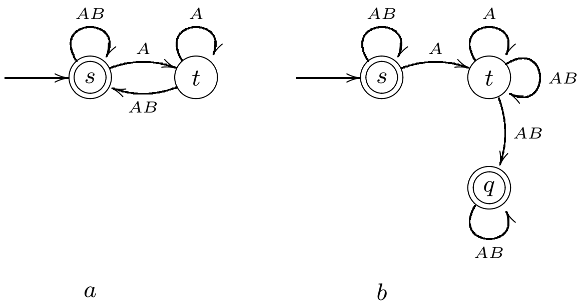


Fig. 8 Models of a fair lossy synchronous channel with a) a weak fairness condition, b) a strong fairness condition.

Conversely, let $\theta \in TDS^{\mathcal{N}}$ be such that $\theta \in \Theta(L(B(A)))$. Then there is a stream of records $r = r_0 r_1 \cdots \in L(B(A))$, with $\theta = \Theta(r)$, that is, for all $k \geq 0$, $\theta.N(k) = dom(r_k)$ and $\forall n \in dom(r_k)$, $\theta.\delta(k)_n = r_k.n$. Because $r \in L(B(A))$, there is an infinite computation $\pi = q_0, r_0, q_1, r_1, \cdots$ in $B(A)$ with $q_0 \in Q_0$ and such that for all $i \geq 0$, the triple (q_i, r_i, q_{i+1}) is a transition in $B(A)$. By construction of the Büchi automaton $B(A)$ from the constraint automaton A, there is an infinite computation $\pi' = q_0, (N_0, g_0), q_1, (N_1, g_1), \cdots$ in A such that for all $i \geq 0$, there is a data assignment $\delta_i{:}N \rightarrow \mathcal{D}$ which $\delta_i \models g_i$ and $\forall n \in N_i, r_i.n = \delta(n)$. Thus, $\theta = \Theta(r)$ and $\theta \in \mathcal{L}(A)$. Therefore, $\Theta(L(B(A))) \subseteq \mathcal{L}(A)$. □

It follows that Büchi automata of records are at least as expressive as constraint automata. They are actually more expressive, because Büchi automata of records are closed under complement while constraint automata are not [6].

3.4 Fair Reo connectors

In this section, we present some useful connectors that can be modeled by Büchi automata of records while this is not the case if we consider constraint automaton.

Example 7 Consider the connector (over a singleton data domain) between two ports A and B with the behavior described by the Büchi automaton of records in Figure 8.a. It is a connector similar to the non-deterministic lossy synchronous channel depicted in Figure 7.b but with this extra property that not all data can get lost. Still infinitely many data can get lost,

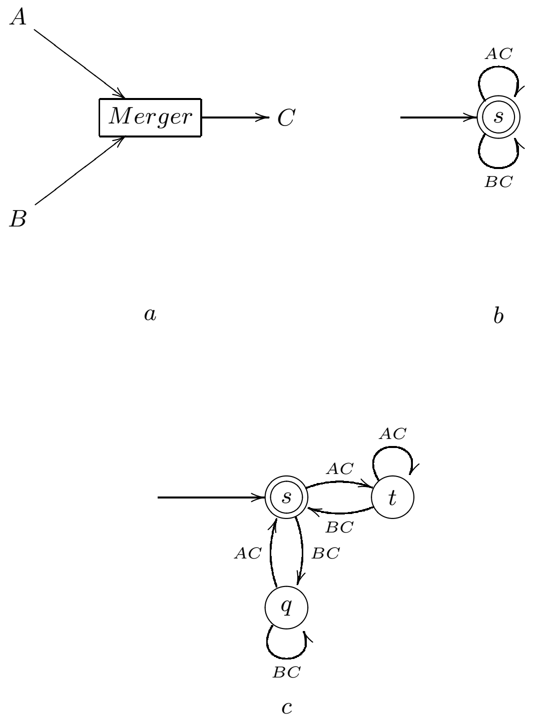


Fig. 9 Models of a merger connector

while the non-deterministic lossy synchronous channel modeled by Büchi automaton of records in Figure 8.b allows for loosing only *finitely many* data at the port A.

Because Büchi automata of records are Büchi automata, we can express *unconditional fairness conditions* [20]: in each infinite execution of the system, some actions should occur infinitely many times.

Example 8 Consider the *merger* connector among two source ports A and B and one sink port C (see Figure 9(a)). Intuitively, it transmits synchronously data item from either A or B to the port C. If both the source ports A and B are offering data at the same time than only one of them is chosen non-deterministically. The Büchi automaton of records corresponding to its constraint automaton model introduced in [6] is shown in Figure 9(b). Both models allow unfair executions where data from the same source is always preferred if both A and B are always offering data simultaneously. Figure 9(c) shows a Büchi automaton that disallows those unfair executions. Because constraint automata do not distinguish between accepting and non-accepting states, they cannot express this kind of fairness conditions [6].

3.5 Composition of Büchi automata of records

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 (synchronous) 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 [26]. 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, $\mathcal{F} \subseteq 2^Q$. A stream $w \in \Sigma^\omega$ is accepted by generalized Büchi automaton B if and only if there is an infinite computation π for w in B such that for every $F \in \mathcal{F}$ at least one of the states in F occurs in π infinitely often.

Definition 8 *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 as follows:*

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

The language of the product of two generalized Büchi automata is the intersection of their respective languages [26].

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 $L(B) = L(B')$ [26].

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 9 *Let* $B_1 = \langle Q_1, Rec_{\mathcal{N}_1}(\mathcal{D}), \longrightarrow_1, Q_{01}, F_1\rangle$ *and* $B_2 = \langle Q_2, Rec_{\mathcal{N}_2}(\mathcal{D}), \longrightarrow_2, Q_{02}, F_2\rangle$ *be two BAR's. We define the* join *of* B_1 *and* B_2 *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:*

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

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

and dually,

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

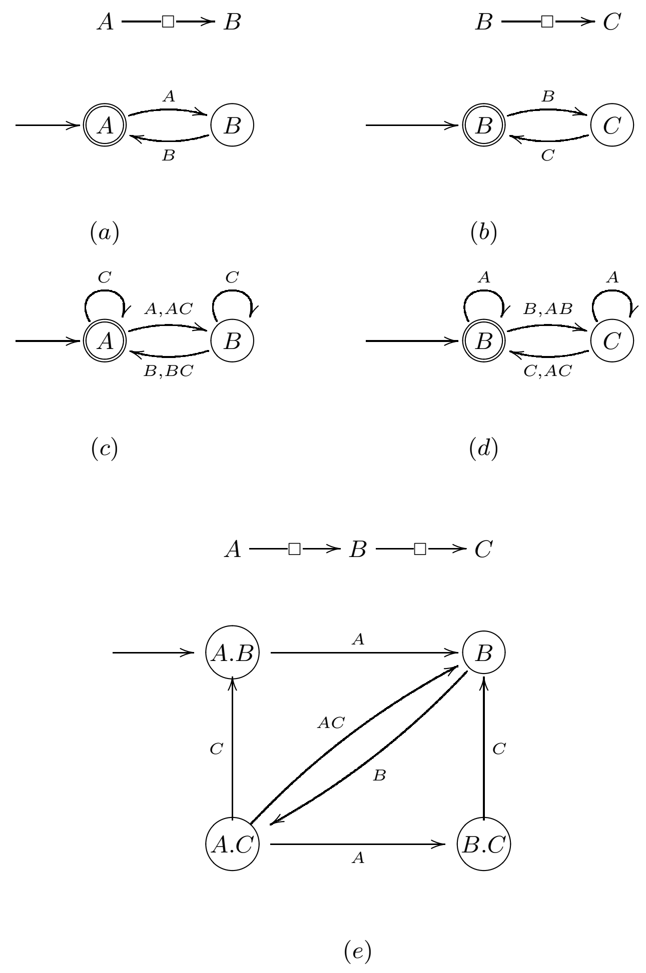


Fig. 10 Composing two FIFO1 channels

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 interleave if they are labeled with records not referring to ports of the other automaton.

Example 9 For example, consider Figure 10 (at this moment ignoring the labels on the states). Figure 10(a) shows the Büchi automaton of records modeling a FIFO1 channel between ports A and B (using as data set $\mathcal{D} = \{d\}$) and (b) a FIFO1 between ports B and C over the same data set. The join of these two automata is shown in (e). In Figure 10(e) the generalized set of final states is $\{\{AB, B\}, \{AB, AC\}\}$ where we refer to each state by its label.

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

Lemma 3 *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). On the other hand, our definition of join is correct (even structurally, and not only language theoretically) also with respect to the join of constraint automata.

Theorem 2 *Let A_1 and A_2 be two constraint automata. Then,*

$$B(A_1) \bowtie B(A_2) = B(A_1 \bowtie_C A_2)$$

.

Proof Let $A_1 = \langle Q_1, \mathcal{N}_1, T_1, Q_{01}\rangle$ be a constraint automaton and $A_2 = \langle Q_2, \mathcal{N}_2, T_2, Q_{02}\rangle$ be another one. Using Definition 3

$$A_1 \bowtie_C A_2 = \langle Q_1 \times Q_2, \mathcal{N}_1 \cup \mathcal{N}_2, T, Q_{01} \times Q_{02}\rangle,$$

where T is the set of all transitions obtained using rules presented in Definition 3. Using Definition 7, $B(A_1 \bowtie_C A_2)$ is

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

where Δ_C is the set of transitions $(\langle s,t\rangle, r, \langle s',t'\rangle)$ such that there exists $(\langle s,t\rangle, N, g, \langle s',t'\rangle) \in T$ and $\delta{:}N \rightarrow D$ such that $\delta \models g$ and for all n in N, $r.n = \delta(n)$.

Further, let $B(A_1) = \langle Q_1, Rec_{\mathcal{N}_1}(\mathcal{D}), \Delta_1, Q_{01}, Q_1\rangle$ and $B(A_2) = \langle Q_2, Rec_{\mathcal{N}_2}(\mathcal{D}), \Delta_2, Q_{02}, Q_2\rangle$, with Δ_1 and Δ_2 obtained as described in Definition 7. Using Definition 9, $B(A_1) \bowtie B(A_2)$ is the automaton

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

with Δ_B the set of all transitions obtained using the rules in Definition 9. We need to prove that $\Delta_C = \Delta_B$

To begin with, we prove that $\Delta_C \subseteq \Delta_B$.
Let $(\langle s,t\rangle, r, \langle s',t'\rangle) \in \Delta_C$. There is $(\langle s,t\rangle, N, g, \langle s',t'\rangle) \in T$ and data assignment $\delta{:}N \rightarrow \mathcal{D}$, such that $\delta \models g$ and $\forall n \in N, r.n = \delta(n)$. We have two cases:
1) If $(\langle s,t\rangle, N, g, \langle s',t'\rangle) \in T$ is obtained using the first rule in Definition 3, then, there are $(s, N_1, g_1, s') \in T_1$ and $(t, N_2, g_2, t') \in T_2$ such that, $N = N_1 \cup N_2$, $N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1$, $\emptyset \neq N_1 \subseteq \mathcal{N}_1$ and $\emptyset \neq N_2 \subseteq \mathcal{N}_2$. Let $\delta|_{N_1}$ and $r|_{N_1}$ be respectively the restricted versions of δ and r for domain $N_1 \subseteq N$. Obviously, $\delta|_{N_1} \models g_1$ and $\forall n \in N_1, r|_{N_1}.n = \delta|_{N_1}(n)$. Similarly, $\delta|_{N_2} \models g_2$ and $\forall n \in N_2, r|_{N_2}.n = \delta|_{N_2}(n)$. Thus, based on definitions of Δ_1 and Δ_2, we conclude that $(s, r|_{N_1}, s') \in \Delta_1$ and $(t, r|_{N_2}, t') \in \Delta_2$. Because $r|_{N_1}$ and $r|_{N_2}$ both are restricted versions of r, $r|_{N_1}$ and $r|_{N_2}$ are compatible and $r|_{N_1} \cup r|_{N_2} = r$. Thus, using the first rule in Definition 9, $(\langle s,t\rangle, r, \langle s',t'\rangle) \in \Delta_B$.
2) If $(\langle s,t\rangle, N, g, \langle s',t'\rangle) \in T$ is obtained using the second rule in Definition 3, then, there is $(s, N, g, s') \in T_1$ such that, $t = t'$ and $N \cap \mathcal{N}_2 = \emptyset$. Thus, based on definition of Δ_1, $(s, r, s') \in \Delta_1$. Because $dom(r) \subseteq N$, thus, $dom(r) \cap \mathcal{N}_2 = \emptyset$. Using rule 2 in Definition 9, we conclude that $(\langle s,t\rangle, r, \langle s',t'\rangle) \in \Delta_B$.

It remains to prove $\Delta_B \subseteq \Delta_C$. Let $(\langle s,t\rangle, r, \langle s',t'\rangle) \in \Delta_B$. We have two cases:
1) If $(\langle s,t\rangle, r, \langle s',t'\rangle) \in \Delta_B$ is obtained using the first rule in Definition 9, then, there are $(s, r_1, s') \in \Delta_1$ and $(t, r_2, t') \in$

Δ_2 such that, $r = r_1 \cup r_2$, records r_1 and r_2 are compatible, $dom(r_1) \cap \mathcal{N}_2 = dom(r_2) \cap \mathcal{N}_1$, $r_1 \neq \tau$ and $r_2 \neq \tau$. Thus, based on definitions of Δ_1 and Δ_2, we conclude that there are $(s, N_1, g_1, s') \in T_1$ and $(t, N_2, g_2, t') \in T_2$ and data assignments $\delta_1{:}N_1 \to \mathcal{D}$ and $\delta_2{:}N_2 \to \mathcal{D}$ such that $\delta_1 \models g_1$, $\delta_2 \models g_2$, $\forall n \in N_1, r.n = \delta_1(n)$ and $\forall n \in N_2, r.n = \delta_2(n)$. Let $N = N_1 \cup N_2$, $g = g_1 \wedge g_2$ and $\delta = \delta_1 \cup \delta_2$. Because $dom(r_1) \cap \mathcal{N}_2 = dom(r_2) \cap \mathcal{N}_1$, thus, $N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1$ and using the first rule in Definition 3, $(\langle s, t\rangle, N, g, \langle s', t'\rangle) \in \Delta_B$. Obviously, $\delta \models g$ and $\forall n \in N, r.n = \delta(n)$. Thus, by construction $(\langle s, t\rangle, r, \langle s', t'\rangle) \in \Delta_C$.
2) If $(\langle s, t\rangle, r, \langle s', t'\rangle) \in \Delta_B$ is obtained using the second rule in Definition 9, then, there is $(s, r, s') \in \Delta_1$ such that $t = t'$ and $dom(r) \cap \mathcal{N}_2 = \emptyset$. Based on the definition of Δ_1, there is $(s, N, g, s') \in T_1$ and there are data assignments $\delta{:}N \to \mathcal{D}$ such that, $\delta \models g$ and $\forall n \in N, r.n = \delta(n)$. Because $dom(r) \cap \mathcal{N}_2 = \emptyset$, $N \cap \mathcal{N}_2 = \emptyset$ and using the second rule in Definition 3, $(\langle s, t\rangle, N, g, \langle s', t'\rangle) \in \Delta_B$. Thus,$(\langle s, t\rangle, r, \langle s', t'\rangle) \in \Delta_C$. □

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

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

where $q \overset{r\setminus A}{\longrightarrow'} p$ 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.

3.6 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 standard 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 correct behavior non-deterministically, by allowing or not the simultaneous exchange of data with the other ports known by the automata.

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

$$B{\uparrow}n = \langle Q, Rec_{\mathcal{N}\cup\{n\}}(\mathcal{D}), \widehat{\Delta}, Q_0, F\rangle$$

where $\widehat{\Delta} = \Delta$ if $n \in \mathcal{N}$ and otherwise $\widehat{\Delta} = \Delta \cup \Delta' \cup \Delta''$ where $\Delta' = \{(q, [n = d], q) | q \in Q, d \in \mathcal{D}\}$ and $\Delta'' = \{(q, r[n{:}= d], q') | (q, r, q') \in \Delta, d \in \mathcal{D}\}$.

Intuitively, to extend Büchi automaton of records B using one extra port name n, 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 n. There are three kind of guess: the environment does not use the name n in a communication (explaining why $\Delta \subseteq \widehat{\Delta}$), 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 n 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,r[n:=d],q') in Δ'. Recall here that $r[n{:}= d]$ is the extension of record r by adding the new field $n = d$ to it).

Example 10 For example, in Figure 10(c) we show the extension of the automaton that has been shown in Figure 10(a) with respect to the new port name C. In this Figure, $A \xrightarrow{A,AC} B$ means that there are two transitions $A \xrightarrow{A} B$ and $A \xrightarrow{AC} B$. Figure 10(d) shows the extension of the automaton in Figure 10(b) with respect to the port name A.

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

Given two Büchi automata of records B_1 and B_2 we can extend each of 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 3 *Let B_1 and B_2 be two Büchi automata of records over alphabet sets $Rec_{\mathcal{N}_1}(\mathcal{D})$ and $Rec_{\mathcal{N}_2}(\mathcal{D})$ respectively. Then,$B_1{\uparrow}\mathcal{N}_2 \times B_2{\uparrow}\mathcal{N}_1 = B_1 \bowtie B_2$.*

Proof Let $B_1 = \langle Q_1, Rec_{\mathcal{N}_1}(\mathcal{D}), \Delta_1, Q_{01}, F_1\rangle$ and $B_2 = \langle Q_2, Rec_{\mathcal{N}_2}(\mathcal{D}), \Delta_2, Q_{02}, F_2\rangle$. Using Definition 9, $B_1 \bowtie B_2$ is

$$\langle Q_1 \times Q_2, Rec_{\mathcal{N}_1\cup\mathcal{N}_2}(\mathcal{D}), \Delta_{\bowtie}, Q_{01} \times Q_{02}, F\rangle,$$

where $F = \{F_1 \times Q_2, Q_1 \times F_2\}$ and $\Delta_{\bowtie}$ is the transition relation. Based on the Definition 11, we have

$$B_1{\uparrow}\mathcal{N}_2 = \langle Q_1, Rec_{\mathcal{N}_1\cup\mathcal{N}_2}(\mathcal{D}), \widehat{\Delta_1}, Q_{01}, F_1\rangle$$

and

$$B_2{\uparrow}\mathcal{N}_1 = \langle Q_2, Rec_{\mathcal{N}_1\cup\mathcal{N}_2}(\mathcal{D}), \widehat{\Delta_2}, Q_{02}, F_2\rangle$$

where $\widehat{\Delta_1}$ and $\widehat{\Delta_2}$ are the transition relations. Their product is the Büchi automaton $B_1{\uparrow}\mathcal{N}_2 \times B_2{\uparrow}\mathcal{N}_1$ given by

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

where $\Delta_\times$ is defined according to the Definition 8. We need to prove $\Delta_\times = \Delta_\bowtie$. We start by showing that $\Delta_\times \subseteq \Delta_\bowtie$: Let $(\langle s,t\rangle, r, \langle s',t'\rangle) \in \Delta_\times$. By using Definition 8) and Definition 11) we have,

$$(\langle s,t\rangle, r, \langle s',t'\rangle) \in \Delta_\times \iff (s,r,s') \in \widehat{\Delta_1} \wedge (t,r,t') \in \widehat{\Delta_2}$$

$$\iff (s,r,s') \in \Delta_1 \cup \Delta_1' \cup \Delta_1'' \wedge (t,r,t') \in \Delta_2 \cup \Delta_2' \cup \Delta_2''$$

We need to consider nine different cases:

1. $(s,r,s') \in \Delta_1$ and $(t,r,t') \in \Delta_2$. Obviously, using the first rule in Definition 9, we have $(\langle s,t\rangle, r, \langle s',t'\rangle) \in \Delta_\bowtie$.
2. $(s,r,s') \in \Delta_1$ and $(t,r,t') \in \Delta_2'$. By the definition of Δ_2', $t = t'$ and $r \in Rec_{\mathcal{N}_1 \backslash \mathcal{N}_2}(\mathcal{D})$. Thus, $dom(r) \cap \mathcal{N}_2 = \emptyset$. Therefore, using the second rule in Definition 9, $(\langle s,t\rangle, r, \langle s',t\rangle) \in \Delta_\bowtie$.
3. $(s,r,s') \in \Delta_1$ and $(t,r,t') \in \Delta_2''$. According to definition of Δ_2'', there is $(t,r',t') \in \Delta_2$ such that $dom(r) = dom(r') \cup N'$ for some $N' \subseteq \mathcal{N}_1 \backslash \mathcal{N}_2$ and for all names in $dom(r')$ it holds $r(n) = r'(n)$. Thus, $dom(r) \cap \mathcal{N}_2 = dom(r') \cap \mathcal{N}_1 = dom(r')$, r and r' are compatible and $r \cup r' = r$. Thus $(\langle s,t\rangle, r, \langle s',t'\rangle) \in \Delta_\bowtie$, using Definition 9 Rule 1.
4. $(s,r,s') \in \Delta_1'$ and $(t,r,t') \in \Delta_2$. The proof of this case is the symmetric proof of case 2.
5. $(s,r,s') \in \Delta_1'$ and $(t,r,t') \in \Delta_2'$. This case is impossible, because, by definition of Δ_1', $dom(r) \subseteq \mathcal{N}_2 \backslash \mathcal{N}_1$ and by definition of Δ_2', $dom(r) \subseteq \mathcal{N}_1 \backslash \mathcal{N}_2$ and $dom(r) \neq \emptyset$. Obviously, these conditions are contradictory.
6. $(s,r,s') \in \Delta_1'$ and $(t,r,t') \in \Delta_2''$. This case is impossible. Its proof is similar to case 5.
7. $(s,r,s') \in \Delta_1''$ and $(t,r,t') \in \Delta_2$.The proof of this case is the symmetric proof of case 3.
8. $(s,r,s') \in \Delta_1''$ and $(t,r,t') \in \Delta_2'$. This case is impossible. Its proof is similar to case 5.
9. $(s,r,s') \in \Delta_1''$ and $(t,r,t') \in \Delta_2''$. According to definition of Δ'', there are records r' and r'' such that $dom(r) = dom(r') \cup N' = dom(r'') \cup N''$ for $N' \subseteq \mathcal{N}_2 \backslash \mathcal{N}_1$ and $N'' \subseteq \mathcal{N}_1 \backslash \mathcal{N}_2$. By a simple set theoretic justification, it can be shown that, $dom(r') \cap \mathcal{N}_2 = dom(r'') \cap \mathcal{N}_1$ and because, $\forall n \in dom(r') {:} r(n) = r'(n)$ and $\forall n \in dom(r'') {:} r(n) = r''(n)$ thus $r = r' \cup r''$. Thus, using Definition 9 Rule 1, $(\langle s,t\rangle, r, \langle s',t'\rangle) \in \Delta_\bowtie$.

Next we prove that $\Delta_\bowtie \subseteq \Delta_\times$. Let $(\langle s,t\rangle, r, \langle s',t'\rangle) \in \Delta_\bowtie$. We have two cases.

1. If $(\langle s,t\rangle, r, \langle s',t'\rangle) \in \Delta_\bowtie$ is obtained using the first rule of Definition 9, there are $(s,r_1,s') \in \Delta_1$ and $(t,r_2,t') \in \Delta_2$ such that r_1 and r_2 are compatible, $r = r_1 \cup r_2$ and $dom(r_1) \cap \mathcal{N}_2 = dom(r_2) \cap \mathcal{N}_1$. Obviously, $(s,r,s') \in \Delta_1''$ and $(t,r,s') \in \Delta_1''$. Thus, $(\langle s,t\rangle, r, \langle s',t'\rangle) \in \Delta_\times$.

2. If $(\langle s,t\rangle, r, \langle s',t'\rangle) \in \Delta_\bowtie$ is obtained using the second rule of Definition 9, there is $(s,r,s') \in \Delta_1$ such that $dom(r) \cap \mathcal{N}_2 = \emptyset$ and $t = t'$. Because r is in $Rec_{\mathcal{N}_1}(\mathcal{D})$ and $dom(r) \cap \mathcal{N}_2 = \emptyset$ thus, $r \in Rec_{\mathcal{N}_1 \backslash \mathcal{N}_2}(\mathcal{D})$. Based on the definition of Δ', we have that $(t,r,t) \in \Delta_2'$. Thus, $(s,r,s') \in \widehat{\Delta_1}$ and $(t,r,t') \in \widehat{\Delta_2}$. Therefore, using the definition of Büchi product, we obtain that the transition $(\langle s,t\rangle, r, \langle s',t'\rangle)$ is in $\Delta_\times$. □

Therefore, to join two Büchi automata of records, one can first extend them to a common set of ports and then compose the resulting Büchi automaton using the standard Büchi product operation. Based on the previous theorem, the automata produced by both methods are structurally, and thus also language theoretically, the same.

Example 11 For example, the join of the Büchi automata of records shown in Figures 10(a) and (b) is the automaton shown in (e). This automaton, in turn, is the *product* of the automata depicted in (c) and (d) which are the extended versions of (a) and (b), respectively. The resulting automaton models a FIFO2 channel . Note that one of the diagonal transitions corresponds to the move of data from one cell to the other, while the other diagonal models the simultaneous consumption of data from port C and the insertion of a new data to the port A.

4 Modeling context-dependencies

In the previous section we have addressed one specific shortcoming of the constraint automata as model of Reo networks, namely the impossibility to model desirable fairness constraints. Next we address another deficiency of constraint automata, that is, their impossibility to model behavior that depend upon pending I/O operations on the ports of the connector. This latter property is called *context dependency*, which manifests itself when the behavior of a connector can change depending upon not only the presence of requests on a connector boundary, but also on their absence.

4.1 Augmented Büchi automata of records

In order to take into account context dependencies in Reo we naturally augment our Büchi automata model for component connector. The prototypical Reo connector featuring a context depended behavior is the *context dependent* lossy synchronous channel (not to be confused with the previous nondeterministic and fair lossy synchronous channels): if the port connected at the source is ready to send data but the port at the sink is not ready to receive, then the data at the source is lost. Until now, such requirements have been ignored and lossy synchronous channels have been modeled by constraint automata or BAR's using a (fair) non-deterministic choice. While this is sufficient for modeling Reo networks like the the exclusive router presented in Figure 2, in general, the presence of context dependent lossy synchronous channels increase the expressivity of the Reo models [1].

Next 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

$$e ::= 1 \,|\, A \,|\, e \cdot e \,|\, \bar{e}\,,$$

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:

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

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 $e_1 \cdot e_2$, 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 the disjunction e_1, e_2 of two tests e_1 and e_2 as an abreviation for $\overline{\bar{e_1} \cdot \bar{e_2}}$. Similarly, we let 0 be the false test defined as $\bar{0}$. We use $\equiv$ to denotes 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 $\equiv$):

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

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 12 *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 [18]. 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 it accepts.

Note that the condition of an ABAR $\langle B, l\rangle$ that for each 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 $L_{vis}(B_1) = L_{vis}(B_2)$.

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.

4.2 Context-dependent Reo connectors

Next we present few examples of component connectors modeled by ABARs. Figure 11 shows an ABAR model for a context dependent lossy synchronous channel over a singleton data domain. It connects the source port A with the sink port B. Note that the port A is never pending because if A is enabled in a state then it can be executed. However port B can pend its request to receive data if no data is available at the source A (as described by the expression labelling the upper rightmost state)

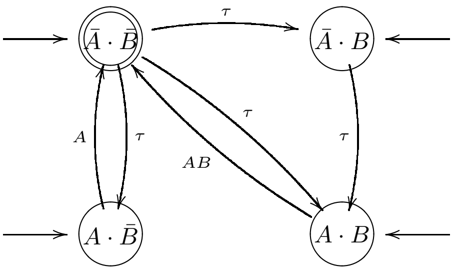
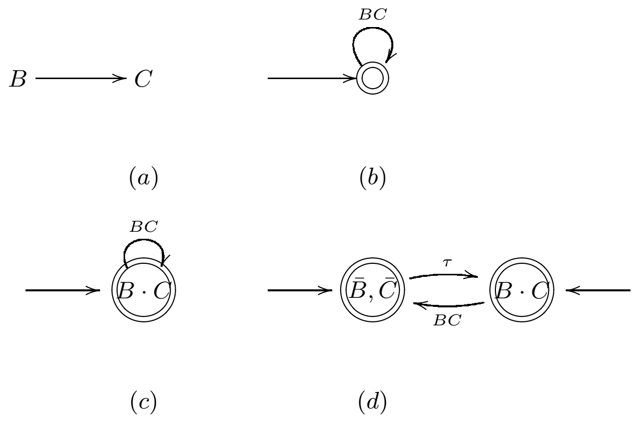


Fig. 11 A context dependent lossy synchronous channel



(a) (b)

(c) (d)

Fig. 12 Models of the synchronous channel

In Figure 12 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) accept only the infinite guarded string

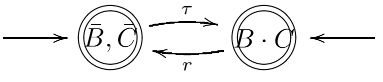
$$\{B,C\}[B=d,C=d]\{B,C\}[B=d,C=d]\cdots,$$

the other automaton accepts infinitely many strings, including

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

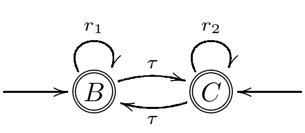
It is easy to see that the two automata are visibly equivalent.

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



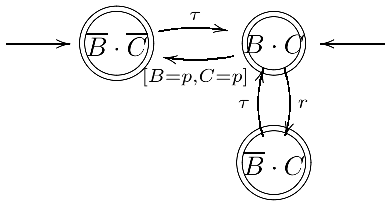
,

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 $dom(r_2)=\{C\}$.

Finally, a context dependent filter channel from B to C is a synchronous channel which allows for the communication only of a specific data item p. Other data value get lost. We can model this channel using record $[B=p,C=p]$ where p is the special value of the filter. Thus, the ABAR model of a context dependent filter channel is:



where $dom(r)=\{B\}$ and $r.B\neq p$.

4.3 Composing ABAR's

Next 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.

We define the join of two ABAR's in terms of the join of their underlying BAR's.

Definition 13 *Let $\langle B_1,l_1\rangle$ and $\langle B_2,l_2\rangle$ be two ABAR's. 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 $l(\langle q,p\rangle)=l_1(q)\cdot l_2(p)$.*

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 $l_1(q_1)\cdot l_2(q_2)$ 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 $l_1(q_1)\cdot l_2(q_2)$ 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 automata 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 11 and the synchronous channel from B to C as modeled in Figure 12.d. Their join is the ABAR shown in Figure 13. Note that it is very similar to the model of an 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 14 *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.*

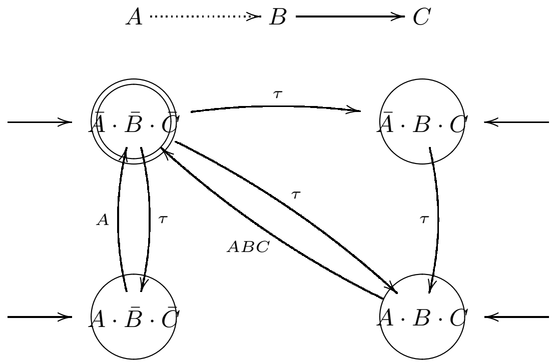

$A \dashrightarrow B \longrightarrow C$
$\bar{A}\cdot\bar{B}\cdot\bar{C}$
τ
$\bar{A}\cdot B\cdot C$
A
τ
τ
ABC
τ
$A\cdot\bar{B}\cdot\bar{C}$
$A\cdot B\cdot C$

Fig. 13 The composition of a context dependent lossy synchronous channel with a context depended synchronous channel

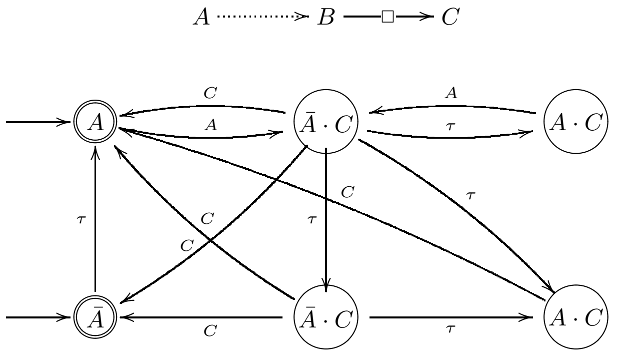


Fig. 14 The composition of a context dependent lossy synchronous channel with a FIFO1 channel

For example, in Figure 14 we consider the composition of a lossy synchronous channel as modeled in Figure 11 with a FIFO1 channel, after hiding the common port B (FIFO1 model is the same as shown in Figure 10.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 happens when the buffer is full.

4.4 Splitting the join

Next we show that the procedure of splitting the join into name extension and production that we have introduced for BAR's is also applicable for the case of ABAR's.

Theorem 4 *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 $l'(\langle q_1, q_2\rangle) = l_1(q_1)\cdot l_2(q_2)$.

Proof The proof is a simple extension of the proof of Theorem 3.

For example, consider Figure 10 (now with the labels of the states). Figures 10(a) and 10(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 10(c), while the extension of the second automaton with the port name A is given in 10(d). Their product is the automaton in 10(e) which can be obtained by either using the direct or the splitting definitions of the join operation.

5 Conclusion and discussion

In this paper, we introduced a simple mathematical model addressing the full semantical features of Reo. Like constraint automata [6] and unlike connector coloring [13], our model can express the behavior of a connector in terms 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 [25,11]). Existing model checkers for Büchi automata, such as SPIN [14] and NuSMV [10], could be used directly for networks of connectors instead of reinventing 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*, Mathematical Structure in Computer Science, **14(3)**, (2004), 329-366.
2. Arbab F., Baier C., de Boer F., Rutten J., Sirjani M., *Synthesis of Reo circuites for implementation of component-connector automata specifications*. In Proceedings of Coordination Languages and Models (CORDINATION 2005), LNCS, **3454**, Springer-Verlag, (2005), 236-251.
3. Arbab F., Chothia T., Meng S., and Moon Y.-J., *Component Connectors with QoS Guarantees*. In Proc. of Coordination Languages and Models (CORDINATION 2007)/, LNCS **4467** Springer-Verlag (2007), 286-304.
4. Arbab F. and Rutten J., *A Coinductive calculus of component connectors*. In Proc. of Recent Trends in Algebraic Development Techniques, (WADT 2002) LNCS **2755**, Springer-Verlag (2002), 34-55.
5. Arbab, F., Baier, C., de Boer, F., and Rutten, J. *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 (2004), 198–207.
6. Baier C., Sirjani M., Arbab F., and Rutten J., *Modelling component connectors in Reo by constraint automata*. Science of Computer Programming, **61**, (2006), 75-113.
7. Baier, C., and Wolf, V., *Stochastic reasoning about channel-based component connectors*. In Proc. of Coordination Languages and Models (CORDINATION 2006), LNCS **4037**, Springer-Verlag (2006), 1-15..
8. Bliudze, S. and Sifakis, J. *The Algebra of Connectors - Structuring Interaction in BIP*. IEEE Transactions on Computers, **57:10**, IEEE Computer Society (2008), 1315-1330.
9. Bonsangue, M., Clarke, D., and Silva A. *Automata for Context-Dependent Connectors*. In Proc. of Coordination Languages and Models (CORDINATION 2009), LNCS **5521**, Springer-Verlag (2009), 184-203.

10. Cimatti A., Clarke E., Giunchiglia E., Giunchiglia F., Pistore M., Roveri M., Sebastiani R., and Tacchella A., *NuSMV 2: an open source tool for symbolic model checking*. In Proc. of the 14th CAV, LNCS **2404**, Springer-Verlag (2002), 359-364.
11. Clarke E., Grumberg O., and Peled D., *Model checking*. The MIT Press, (1999).
12. Clarke D., *Coordination: Reo, nets, and logic*. In Proc. Formal Methods on Components and Objects (FMCO 2008), LNCS **5382**, Springer-Verlag (2008), 226–256.
13. Clarke D., Costa D., and Arbab F., *Connector colouring I: synchronisation and context dependency*. Science of Computer Programming, **66(3)**, Elsevier (2007), 205-225.
14. Holzmann G.J., *The model checker SPIN*. IEEE Transactions on software engineering, **23(5)**, IEEE Computer Society (1997), 279-295.
15. Hopcroft J., Motwani R., and Ullman J., *Introduction to automata theory, languages, and computation*, 3rd edition, Addison-Wesley (2006).
16. Izadi M., Bonsangue M., *Recasting constraint automata into Büchi automata*. In Proc. ICTAC 2008, LNCS **5160**, Springer-Verlag (2008), 156-170.
17. Izadi M., Bonsangue M., Clarke D., *Modeling Component Connectors: Synchronisation and Context-Dependency*. In Proc. 6th IEEE Int. Conf. on Software Engineering and Formal Methods (SEFM 2008), IEEE Computer Society, (2008), 303-312.
18. Kaplan D., *Regular expressions and the equivalence of programs*. Journal of Compiting System Science, **3**, (1969) 361-386.
19. Kozen D., *Automata on guarded strings and applications*, Matématica Contemporânea, **24** (2003), 117-139.
20. Kupferman O., Vardi M., *Verification of Fair Transition Systems*. In Proceedings of the 8th International Conference on Computer Aided Verification CAV, LNCS, Springer (1996).
21. Mousavi M., Sirjani M., and Arbab F., *Formal sematics and analysis of component connectors in Reo*. In Proc. of FOCLASA 2005, ENTCS **154**, Elsevier (2005), 83-99.
22. Papadopoulos G., Arbab F., *Coordination Models and Languages*. Advances in Computers **46**, Academic Press, 1998.
23. Remy D., *Efficient representation of extensible records*. In Proc. ACM SIGPLAN Workshop on ML and its applications, (1994), 12-16.
24. Szyperski C., Gruntz D., and Mürer S. *Component software: beyond object-oriented programming (second edition)*, Addison-Wesley, 2002.
25. Vardi M., *An automata-theoretic approach to linear temporal logic*. In LNCS **1043**, Springer-Verlag (1996), 238-266.
26. Thomas W., *Automata on infinite objects*. In Handbook of Theoretical Computer Science, Volume B, Elsevier, (1990), 133-191.