Universiteit
Leiden
The Netherlands

Model checking of component connectors

Izadi, M.

Citation
Izadi, M. (2011, November 6). *Model checking of component connectors*. *IPA Dissertation Series*. Retrieved from https://hdl.handle.net/1887/18189

Version:	Corrected Publisher's Version
License:	Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden
Downloaded from:	https://hdl.handle.net/1887/18189

Note: To cite this publication please use the final published version (if applicable).

4 Fair Component Connectors

In this chapter, we introduce Büchi automata of records and unconditional fair constraint automata as alternative models for the operational semantics of Reo. We compare the expressiveness of these models with that of the original model of constraint automata discussed in the previous chapter. In the first section, we review some shortcomings of constraint automata and of their TDS based semantics in modeling component connectors and motivate the use of records and Büchi automata of records as operational semantics for Reo. In the second section, we introduce the notions of records, streams and languages of records. We also give a bidirectional translation of TDS-languages and record-based languages. The notion of Büchi automaton of records (BAR) is introduced in the third section and we show that every constraint automaton can be translated into a Büchi automaton of records. In the fourth section, we show that BAR's can be used to model Reo connectors especially connectors with some fairness conditions on their behavior using some examples. Therefore, BARs are semantically more powerful than constraint automata. In the fifth section, a set of composition operators for BARs is introduced. We compare the join operator of BARs with its counterpart for constraint automata and show that the join operation can be obtained using two more basic operations, namely, product and alphabet extension. In the subsequent section, we introduce the notion of unconditional fair constraint automata and compare their expressiveness with that of constraint automata and Büchi automata of records. In the last section, we introduce a version of constraint automaton, called *fair constraint automaton*, whose syntax is the same as constraint automaton except that now it has final (accepting) states, but its semantics is based on the languages of streams of records.

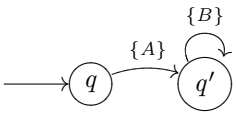
4.1 Introduction

In the previous chapter, we have seen that constraint automata are operational models of Reo connectors. However, we can recognize some shortcomings in using constraint automata as the semantics for Reo. These shortcomings can be categorized in two main groups, those related to the TDS-based semantics of constraint automata and those about the modeling capabilities for Reo connectors. Briefly, the first group shows that the TDS-based semantics of constraint automata is more concrete than what is needed in modeling Reo connectors, whereas the other group shows that constraint automata fail to model all expected behaviors of the connectors. The main shortcomings concerning the TDS-language based semantics of constraint automata can be summarized as follows.

1- Constraint automata are defined as the 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 times).

2- Different than finite and Büchi automata, the simplicity of a constraint automaton is not necessarily reflected by the TDS language it recognizes:

Example 4.1 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 of 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 really irrelevant, yet one needs to describe them all.

In addition, constraint automata fail to model some expected behaviors of Reo connectors. For instance, they cannot model *fairness constraints* over the behaviors of a connector, as well as operations that depend on 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. In this chapter, we concentrate on the issue of fairness constraints. In the next chapter we will discuss context dependencies.

Fairness Constraints Many specification formalisms for reactive and concurrent systems incorporate some notions of fairness constraints such as *unconditional*, *weak*, and *strong fairness*. Informally, the requirement of unconditional fairness disallows executions of the system in which certain sets of actions or situations are taken only finitely many times. In other words, in a model with an unconditional fairness constraint we are interested only in the executions wherein a certain set of actions (or a certain set of the states of the system) are seen infinitely many times. The requirements of weak and strong fairness are conditional. The weak fairness requirement disallows executions in which certain sets of actions or situations are continually enabled but not taken. Namely, the weak fairness requirement states that continually enabled actions or states must occur infinitely often. The requirement of strong fairness disallows executions in which certain sets of actions or states are enabled infinitely often but they are taken only finitely many times. That is, if certain actions or situations are enabled infinitely often then they must occur infinitely often. More formally:

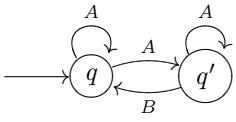
Definition 4.1 A *basic transition system* is a tuple $A = \langle Q, \Sigma, \Delta, q_0 \rangle$ where, Q is a finite set of *states*, Σ is a finite nonempty set of symbols called *alphabet*, $\Delta \subseteq (Q \times \Sigma \times Q)$ is a *transition relation*, and $q_0 \in Q$ is the *initial* state. An infinite *run* of A is an infinite sequence $\rho = q_0, a_0, q_1, a_1, ...$, of alternating states and symbols where, for all i, $(q_i, a_i, q_{i+1}) \in \Delta$. We say that the symbol (action) $a \in \Sigma$ is *enabled* in state $q \in Q$ whenever there exists a transition $(q, a, q') \in \Delta$. Also, we say that the symbol a is taken (or occurs) in position $i \in \mathbb{N}$ of the infinite run $\rho = q_0, a_0, q_1, a_1, ...$ if $a_i = a$. Let ρ be an infinite run in the basic transition system A and $F \subseteq \Sigma$ be a set of symbols. Then,

- ρ is *unconditionally F-fair* if an element in F occurs infinitely often in ρ.
- ρ is *strongly F-fair* if the condition that infinitely often an element in F (not necessarily the same) is enabled implies that an element in F (not necessarily the same) occurs infinitely often in ρ.

- ρ is *weakly F-fair* if the condition that some element in F (not necessarily the same) is eventually always enabled implies that an element in F (not necessarily the same) occurs infinitely often in ρ.

Next we present some simple examples of unconditional fairness constraint in the context of component connectors.

Example 4.2 Consider a channel from port A to port B with a buffer with capacity of one. Suppose that if the buffer is empty, the input data from port A can be saved in the buffer or can get lost. If the buffer is full all other inputs are lost. When the buffer is full, port B is able to get the saved data and then the buffer becomes empty. We call this channel a *Restive-Buffer* channel and model it with the following basic transition system:

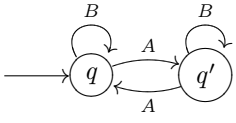


Now, an example of an unconditional fairness constraint is given by considering the (infinite) runs in which the buffer becomes full infinitely many times. Namely, the fair runs are the executions in which state q' (or transition with label B) is taken infinitely many times. In these runs, it is impossible that all input data get lost.

Obviously, if we consider the above transition system as a finite automaton over infinite words (a Büchi automaton) with state q' as final state, the semantics of the model is exactly what we want.

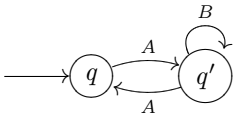
In general, it can be shown that unconditional fairness conditions correspond to the Büchi acceptance condition in the theory of automata on infinite words [101]. Based on this fact, sometimes unconditional fairness conditions are called as Büchi fairness conditions [131].

Example 4.3 Now consider the following basic transition system:



Suppose that we want the model to be weakly fair with respect to the action set $\{B\}$. A transition with action B is continuously enabled in all infinite runs of the above model. A run is weakly fair if it takes transitions with action B infinitely many times. Thus, the run $q, A, q', A, q, A, q', A, \ldots$ and every run in which there are only finitely many transitions with action B are (weakly) unfair with respect to action set $\{B\}$.

Example 4.4 Take the following basic transition system:



Suppose that we want the model to be strongly fair with respect to action set $\{B\}$. In all infinite runs of the above model the transition with action B is enabled infinitely often. An run is strongly fair if it takes the transition with action B infinitely many times. Thus, run $q, A, q', A, q, A, q', A, \ldots$ and every runs in which there are only finitely many transitions with action B are (strongly) unfair with respect to action set $\{B\}$.

In general, strong fairness conditions correspond to the Streett acceptance condition in the theory of automata on infinite words [101]. Interestingly, Streett automata can be efficiently simulated by Büchi automata [130]. Thus, a semantics for component connectors based on Büchi automata is able to specify both unconditional and strong fairness constraints. In this chapter, our main goal is to present this kind of semantics for Reo connectors.

Fair connectors. According to the view point of exogenous coordination, a connector (coordinator) is an open system. By the term of *open system*, we mean that the set of actions (in the case of automata models, the set of transition labels or symbols) are not under the control of the connector. They are fired by the environment. Thus, it makes sense to talk about fairness for the behavior of the system only when there is non-determinism. For deterministic systems/automata, one cannot really talk about their fairness. Whenever a system/automaton has non-deterministic choices it becomes meaningful to expect it to behave fairly. There are different definitions for non-determinism for specification formalisms. To fix our terminology, we call a Reo connector a *non-deterministic connector* if there are possible alternative firings of the ports that the connector can decide to choose (and if the connector *can* decide to make a transition, then staying in the current state is *not* a choice). If a connector is non-deterministic, we can augment its specification or model by fairness constraints.

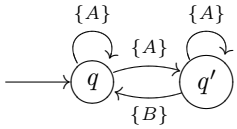
On the other hand, a transition system or automaton can be considered as the model of a *closed system*. For instance, a connector and its environment together can be considered as a closed system and modeled by an automaton. In this case, we can distinguish fair and unfair runs of the system by augmenting its model by fairness constraints.

Thus, in our terminology, speaking about the fair connectors is permitted only for connectors that have non-deterministic choices in their behavior. Also, if an automaton is considered as the model of a connector or an open system and there is non-determinism in its behavior, it can be asked to be fair. However, if we speak generally about fairness for automata models, they should be considered as models of closed systems.

Next, we show that constraint automata are not always able to model the desired fairness conditions, even in the simplest case, namely the unconditional fairness.

Constraint Automata and Fairness. The timed data streams semantics of constraint automata implicitly expresses some unconditional fairness constraints. Let us have an example:

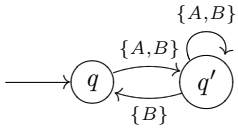
Example 4.5 We model the Restive-Buffer channel that we presented in Example 4.2 with the following constraint automaton:



According to the TDS-languages semantics of constraint automata, in the above automaton port A cannot be fired eventually always because TDS-language semantics forces to assign an infinite sequence of time-data pairs to both ports A and B. Thus, using the TDS based semantics of constraint automata implicitly satisfy the unconditional fairness constraint of Example 4.2.

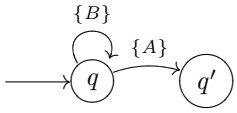
However, there are several cases that the TDS-languages based semantics of constraint automata fails to satisfy some simple fairness conditions:

Example 4.6 Consider the following constraint automaton:



The automaton accepts TDS-languages in which B alone never occurs (even though it is always enabled in state q'). Note that if we consider the above automaton as a Büchi automaton with two simple action names $\{A, B\}$ and $\{B\}$ and two accepting state then action $\{B\}$ can occur alone.

Now, consider the following constraint automaton:



The automaton does not accept any timed data streams tuple, because A cannot appear only once (even if B is initially enabled in state q).

The above example in addition to Example 4.1 show that the TDS-language based semantics of constraint automata sometimes is not able to express fairness constraints but sometimes implicitly it does! Furthermore, timed data streams contain exact time value expressions while in data passage through ports only the temporal orderings of data exchanges are of interest. Thus, TDS-language semantics of constraint automata is more concrete than it is necessary[1].

In this chapter, we introduce the notion of Büchi automaton of records as the alternative operational semantics for Reo with a more standard and simpler semantics which is able to express the desired unconditional and strong fairness conditions. We use records as data

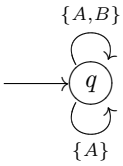
[1] In a later work and in order to define the operational semantics of timed Reo connectors, some of the authors of constraint automata, introduced the notion of *scheduled data streams* [18, 17]. This formalism is similar to our proposed *streams of records* from the view point that both abstract away the exact time stamps and focus only on the ordering.

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 they can not participate in communication. 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 *fairness conditions* admitting only executions for which some actions occur infinitely many times [145]. In the next chapter, we enrich the model to overcome the context dependency problems.

For example, for the lossy synchronous channel which has been introduced in the previous chapter, we will define at least two types of *fair lossy synchronous* channels:

Example 4.7 A lossy synchronous channel from port A to port B behaves as a synchronous channel except that the input data can be non-deterministically lost or delivered to the sink. We call this channel as a *ND-LossySync* channel. If we add the fairness condition that not all data can get lost, we call the channel as a *weak fair LossySync* channel. If we consider stronger fairness condition that only finitely many data can get lost, the channel is called as a *strong fair LossySync* channel.

In [30] a lossy synchronous channel is modeled using the following (deterministic) constraint automaton:



For the moment, suppose that the above model is a basic transition system with infinite traces semantics (or it is a Büchi automaton whose only state is accepting). From the environments viewpoint, firing each one of the two actions can be selected non-deterministically. Thus, it is possible that the transition with action $\{A\}$ is selected forever or the other transition to be selected only finitely many time. In the first case, the model violates both weak and strong fairness conditions. In the other, the model violates the strong fairness constraint. But, what about the above model if it is regarded as a constraint automaton with the TDS-language semantics? As we explained before, the TDS-language semantics forces to assign an infinite sequence of time-data pairs to both ports A and B. Thus, it implicitly satisfies the fairness constraint that not all data at port A get lost.

We also show that every constraint automaton with a *slight correction* of their TDS semantics can be translated into an essentially equivalent Büchi automaton of records. 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 different structures. In fact it is easy to embed a language on streams of records into a language of timed data streams, but not vice-versa. 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 a 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.

4.2 Streams and Languages of Records

Now we introduce 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 from participating communication. 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.

Definition 4.2 Let $\mathcal{N}$ be a finite nonempty set of (port) names and $\mathcal{D}$ a finite nonempty set of data.
(1) We write $Rec_{\mathcal{N}}(\mathcal{D}) = \mathcal{N} \rightharpoonup \mathcal{D}$ for the set of *records* with entries from the set of data $\mathcal{D}$ and labels from the set of names $\mathcal{N}$, consisting of all partial functions from $\mathcal{N}$ to $\mathcal{D}$.
(2) For a record $r \in Rec_{\mathcal{N}}(\mathcal{D})$ we write $dom(r)$ for the domain of r.
(3) 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 $1 \leq i \leq k$. Different than a tuple, the order of the components of a record is irrelevant and its size is not fixed a priori.
(4) We denote by τ the special record with the empty domain, that is $dom(\tau) = \emptyset$.
(5) A *stream of records* over a data set $\mathcal{D}$ and a name set $\mathcal{N}$ is an infinite string of records $w \in Rec_{\mathcal{N}}(\mathcal{D})^{\omega}$.
(6) A *language of (streams of) records* over a data set $\mathcal{D}$ and a name set $\mathcal{N}$ is a set of infinite strings of records $L \subseteq Rec_{\mathcal{N}}(\mathcal{D})^{\omega}$.

We use records as data structures for modeling constrained synchronization of ports in $\mathcal{N}$. Following [127], 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 a (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 the domain $dom(r) \cup \{n\}$ mapping the port n to d and remaining invariant with respect to all other ports. The *hiding* operator '$\setminus$' is used to increase the negative information. For $n \in \mathcal{N}$, the record $r \setminus n$ hides the port n to the environment by setting $dom(r \setminus n) = dom(r) \setminus \{n\}$, and $(r \setminus n).m = r.m$.

Definition 4.3 Let $r_1 \in Rec_{\mathcal{N}_1}(\mathcal{D})$ and $r_2 \in Rec_{\mathcal{N}_2}(\mathcal{D})$.
(1) 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 for all $n \in dom(r_1) \cap dom(r_2)$, $r_1.n = r_2.n$.
(2) 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, for all $n \in dom(r_1)$, $(r_1 \cup r_2).n = r_1.n$ and for all $n \in dom(r_2)$, $(r_1 \cup r_2).n = r_2.n$.

4.2.1 Bidirectional Translation of Record and TDS-Languages

Let us compare the expressiveness of TDS-languages with that of languages of streams of records. First, we introduce a slight modification in the definition of timed data stream:

Definition 4.4 Let $\mathcal{N}$ be a fixed finite set of *port names* and $\mathcal{D}$ 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.

The only difference of the above definition of timed data stream and the original one (see Definition 3.2) is that in the present definition the time value ∞ (infinity) is also allowed. This simplifies our next discussions and will solve some of the problems [2].

For instance in the case of Example 4.1 it is enough to consider the values of all events time but the first in θ_A to be infinity. The definitions of TDS-tuples and TDS-languages remain the same as previously defined.

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 $\theta.time(k)$. 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 $L_{TDS} \subseteq TDS^{\mathcal{N}}$,

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

Example 4.8 Let

A	d	d'	d"	...
	0.5	0.7	1.9	
B	d	d'		...
	0.5	1.2		

[2] In [16], Arbab uses the $\perp$ symbol in a footnote as a special value for the time values in time streams to model *finite behavior*. This is similar to using ∞ as we have here.

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 stamps 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 times when data are exchanged so to respect the relative order of communication imposed by ρ. Formally,

$$\Theta(\rho) = \{\theta \mid \forall k \geq 0{:}(\theta.N(k) = dom(\rho(k)) \wedge \forall n \in dom(\rho(k)){:}\theta.\delta(k)_n = \rho(k).n)\}\,.$$

Example 4.9 For example, for ρ being the stream of records as in Example 4.8 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 4.8 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 $\Theta{:}2^{Rec_{\mathcal{N}}(\mathcal{D})^{\omega}} \rightarrow 2^{TDS^{\mathcal{N}}}$ is an embedding of languages over records into TDS-languages for $\mathcal{N}$.

Lemma 4.1 For each $L \subseteq Rec_{\mathcal{N}}(\mathcal{D})^{\omega}$, $L = \Upsilon(\Theta(L))$.

Proof.
Let $\rho \in L$ be a stream of records. Since $\rho = \Upsilon(\Theta(\rho))$ we have $L \subseteq \Upsilon(\Theta(L))$.
Now Let $\rho \in \Upsilon(\Theta(L))$. There are a stream of records $\rho' \in L$ and a TDS-tuple $\theta \in TDS^{\mathcal{N}}$ such that $\rho = \Upsilon(\theta)$ and $\theta = \Theta(\rho')$. Thus, θ is a proper time assignment into ρ' and ρ is the time abstraction of θ. Obviously it should be $\rho = \rho'$. Thus, $\Upsilon(\Theta(L)) \subseteq L$. □

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

4.3 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 [138].

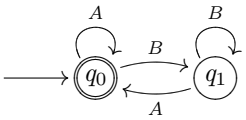


Figure 4.1: A Büchi automaton for L in Example 4.10

4.3.1 Büchi Automata: A Review

A *Büchi automaton* is a non-deterministic finite state automaton which takes infinite words as input. A word is accepted if the automaton goes through some designated *final* or *accepting* state infinitely often while reading the word. More formally:

Definition 4.5
(1) A *Büchi automaton* is a tuple $B = \langle Q, \Sigma, \Delta, Q_0, F\rangle$ 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.
(2) 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.
(3) 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.
(4) We say that two Büchi automata B_1 and B_2 are *(language-based) equivalent* if $L(B_1) = L(B_2)$.
(5) Let $B = \langle Q, \Sigma, \Delta, Q_0, F\rangle$ be a Büchi automaton. B is called as a *deterministic* Büchi automaton if $|Q_0| \leq 1$ and the transition relation Δ can be considered as a function of the form $\Delta{:}(Q \times \Sigma) \to Q$.

If we regard the state space of a Büchi automaton as a graph, an accepting computation (or run) traces an infinite path which start at some state $q_0 \in Q_0$, reaches an accepting state $q_F \in F$ and, thereafter, keeps looping back to q_F infinitely often. In graphical representation of Büchi automata accepting states are distinguished from other states by drawing them with a double circle.

Example 4.10 Consider the alphabet $\Sigma = \{A, B\}$. Let $L \subseteq \Sigma^\omega$ consist of all infinite words α such that there are infinitely many occurrences of A in α. Figure 4.1 shows a Büchi automaton recognizing L. The initial state is marked by an arrow without a source. There is only one accepting state q_0 which is indicated by a double circle. In this automaton, all transitions labeled A lead into the accepting state and, conversely, all transitions coming into the accepting state are labeled A. From this, it follows that the automaton accepts an infinite word if and only if it has infinitely many occurrences of A.

The complement of L, which we denote $\bar{L}$, is the set of all infinite words α such that α has only finitely many occurrences of A. An automaton recognizing $\bar{L}$ is shown in Figure 4.2.

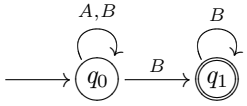


Figure 4.2: A Büchi automaton for $\bar{L}$ in Example 4.10

The automaton guesses a point in the input beyond which it will see no more A's - such a point must exist in any input with only a finite number of A's. Once it has made this guess, it can process only B's - there is no transition labeled A from the second state, so if it reads any more A's it gets stuck.

In the above example, notice that the automaton recognizing L is deterministic while the automaton for $\bar{L}$ is non-deterministic. It can be shown that the non-determinism in the second case is unavoidable - that is, there is *no* deterministic automaton recognizing $\bar{L}$. This means that Büchi automata are fundamentally different than their counterparts on finite inputs: we know that over finite words, deterministic automata are as powerful as non-deterministic automata. In other words, non-deterministic Büchi automata are strictly more powerful than deterministic Büchi automata: there are languages recognized by non-deterministic Büchi automata that cannot be recognized by any deterministic Büchi automaton [138].

Generalized Büchi Automata In several applications, other types of automata on infinite objects are useful. In fact, there are several variants of automata on infinite words that are equally expressive as nondeterministic Büchi automata, although they use more general acceptance conditions. For some of these automata, the deterministic version has the full power of nondeterministic Büchi automata. Muller, Rabin and Streett automata are examples of these types of automata on infinite words [138]. Also, Büchi automaton itself has some slight variants, called *generalized* and *alternating Büchi automata*, both of which are equally expressive as nondeterministic Büchi automata. For the purpose of this thesis, it suffices to consider *generalized (nondeterministic) Büchi automata*. The difference between a Büchi automaton and a generalized Büchi automaton is that the acceptance condition of the generalized one requires to visit several sets (of final states) $F_1, \ldots, F_k$ infinitely often. More formally:

Definition 4.6
(1) A *generalized Büchi automaton* is a Büchi automaton $B = \langle Q, \Sigma, \Delta, Q_0, \mathcal{F} \rangle$ but for the set of final states, that now is a set of sets, that is, $\mathcal{F} \subseteq 2^Q$.
(2) A stream $\omega \in \Sigma^\omega$ is accepted by generalized Büchi automaton B if and only if there is an infinite computation π for ω in B such that for every $F \in \mathcal{F}$ at least one of the states in F occurs in π infinitely often.

The definitions of languages recognized by generalized Büchi automata and their equivalence are the same as for the case of Büchi automata.

Example 4.11 Figure 4.3 shows a generalized Büchi automaton over the alphabet set $\Sigma = \{A, B, C\}$ with the acceptance sets $F_1 = \{q_1\}$ and $F_2 = \{q_2\}$. The accepted language

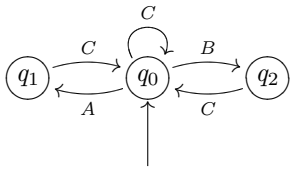


Figure 4.3: A generalized Büchi automaton with the set of accepting sets $\mathcal{F} = \{\{q_1\}, \{q_2\}\}$.

consists of all infinite words over the alphabet set $\Sigma = \{A, B, C\}$ such that both A and B hold infinitely often (possibly at different positions).

Remark 4.1 The set $\mathcal{F}$ of accepting sets of a generalized Büchi automaton may be empty. If $\mathcal{F} = \emptyset$ the stream ω is accepted if and only if there exists an infinite computation for ω in the automaton. Note the difference with the case of an ordinary Büchi automaton whose set of final states is empty. For a Büchi automaton whose set of final states is empty, there are *no* accepting computations and the language of the automaton is empty. Contrary to that, *every* infinite computation of a generalized Büchi automaton with $\mathcal{F} = \emptyset$ is accepting.

Clearly, every Büchi automaton is a generalized Büchi automaton with a singleton set of final states, containing the original set of of final states. Conversely, every generalized Büchi automaton can be transformed into an equivalent Büchi automaton:

Lemma 4.2 [3] Let $B = \langle Q, \Sigma, \Delta, Q_0, \mathcal{F} \rangle$ be a generalized Büchi automaton. Then, there exists a Büchi automaton B' such that $L(B) = L(B')$.

Proof. Based on Remark 4.1, if $\mathcal{F} = \emptyset$ then B accepts all infinite strings over the alphabet Σ. In this case, B is equivalent with the Büchi automaton that has only one state, say q, that is both initial and final, and for every $a \in \sigma$, there is a self-transition (q, a, q) in B.

Now, we assume that $\mathcal{F} \neq \emptyset$. Let $\mathcal{F} = \{F_0, \ldots, F_{k-1}\}$, where $k \geq 0$. The basic idea of the construction of B' is to create k copies of B such that the accepting set F_i of the ith copy is connected to the corresponding states of the $i + 1$th copy. The acceptance condition for B' consists of the requirement that an accepting state of the first copy is visited infinitely often. This ensures that all other accepting sets F_i of the k copies are visited infinitely often too.

Now we can define ordinary Büchi automaton $B' = \langle Q', \Sigma, \Delta', Q'_0, F' \rangle$ such that:

- $Q' = Q \times \{0, \ldots, k-1\}$,
- $Q'_0 = Q_0 \times \{0\}$,
- $F' = F \times \{0\}$.

The transition relation $\Delta' \subseteq (Q' \times \Sigma \times Q')$ is defined as follows. For all $q \in Q$, $A \in \Sigma$, and $i \in [0..k-1]$:

[3] This lemma is a known result in the literature of Büchi automata. In the rest of this chapter we will use the construction procedure that is introduced in its proof.

- if $q \notin F_i$, then for all $q' \in Q$ that $(q, A, q') \in \Delta$, $(\langle q, i\rangle, A, \langle q', i\rangle) \in \Delta'$,
- else, for all $q' \in Q$ that $(q, A, q') \in \Delta$, $(\langle q, i\rangle, A, \langle q', (i+1)\ mod\ k\rangle) \in \Delta'$.

Now, we can simply show that $L(B) = L(B')$ (for more detail of the proof see for example [29]). □

4.3.2 Büchi Automata on Streams of Records

In the rest of this chapter, we work with Büchi automata whose alphabet sets are defined as sets of records over some sets of port names and data:

Definition 4.7 Let $\mathcal{N}$ be a finite set of port names and $\mathcal{D}$ a finite set of data. Also, let $B = \langle Q, \Sigma, \Delta, Q_0, F\rangle$ be a Büchi automaton over the alphabet $\Sigma = Rec_{\mathcal{N}}(\mathcal{D})$. We call B as a *Büchi automaton (on streams) of records*, abbreviated by BAR.

In the following example we show that the basic channels of Reo can be modeled by BARs. Thus, not only it will be an example for BARs, but also this example shows the expressive power of BARs as the semantic model of component connectors.

Example 4.12 In Figure 4.4 we show BAR models of the 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. The non-deterministic lossy synchronous channel (ND-LossySync) that we model in Figure 4.4.d is the same as we introduced in Example 4.7. As we mentioned in that example, using fairness assumptions, at least two other versions of the lossy synchronous channel can be defined. Later in this chapter, we will show that using BARs, we are also able to model these two fair lossy synchronous channels (see Section 4.4).

Also, it can be shown that the source and sink nodes in the Reo terminology should be modeled as special kinds of connectors. A source node acts as a *duplicator* channel while a sink node acts as a *merger* (for more detail see Chapter 3). For simplicity of our discussions, in the following example we use duplicator and merger connectors to explicitly show their behavior as Reo primitive and show that they can be modeled by BARs.

Example 4.13 A *duplicator* is a connector with a source and two sink ends. Whenever an entity at the source is ready to put data and the entities at both sinks are ready to get it, data will be delivered from the source to the sinks of this connector synchronously. Thus, a duplicator can be modeled as we have shown in Figure 4.5. Again we assume that the data set is $\mathcal{D} = \{d, d'\}$.

Now, consider the *merger* connector with two source ports A and B and one sink port B. Intuitively, it transmits synchronously data item from either A or B to the port C. If both the source ports A and B offer data at the same time then only one of them is chosen non-deterministically. The Büchi automaton of records model of this connector, when the data set is $\mathcal{D} = \{d, d'\}$, is shown in Figure 4.6.

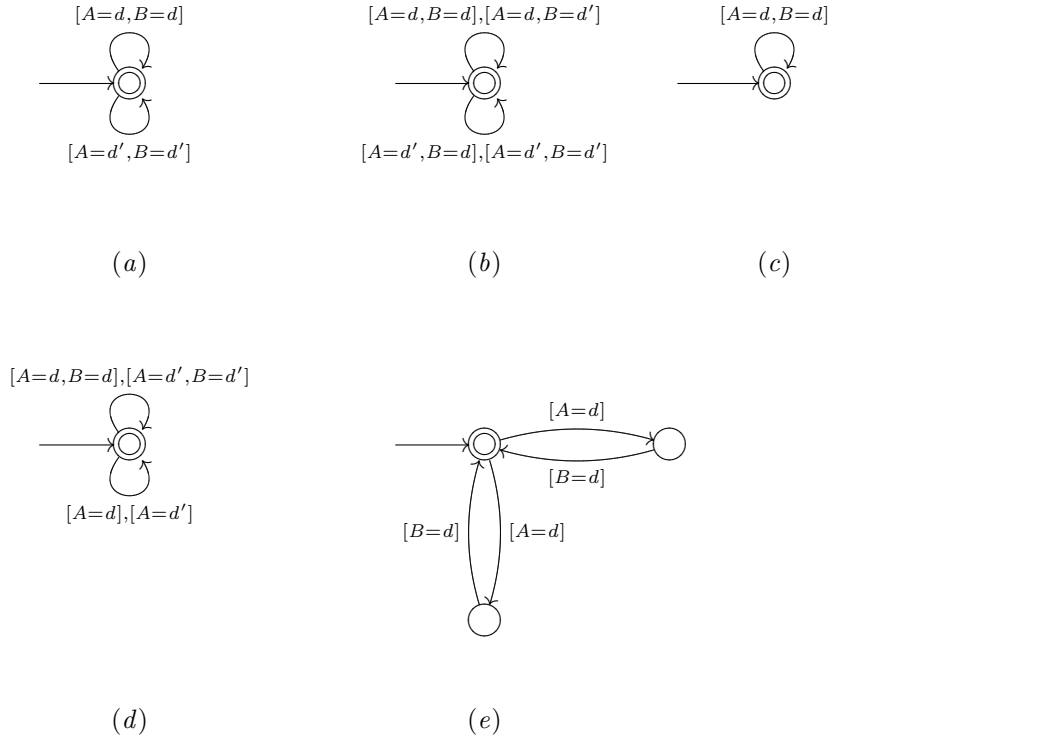


Figure 4.4: BAR models of basic Reo channels: a) Sync channel b) SyncDrain channel, c) Filter channel, (d) ND-LossySync channel, and (e) FIFO1 channel.

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. Since they are externally invisible we may ignore them. However, if we remove all τ symbols from a stream of records ω, the resulting sequence need not to be infinite anymore. For example, removing all τ's from the stream consisting of only τ symbols will result in the empty (and hence finite) string.

Definition 4.8 Let B be a Büchi automaton of records. The *visible language* of B is defined as:

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

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

Note that $L_{vis}(B)$ contains only infinite sequences and therefore is a subset of the set of sequences obtained from removing the τ's from the streams in $L(B)$. 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

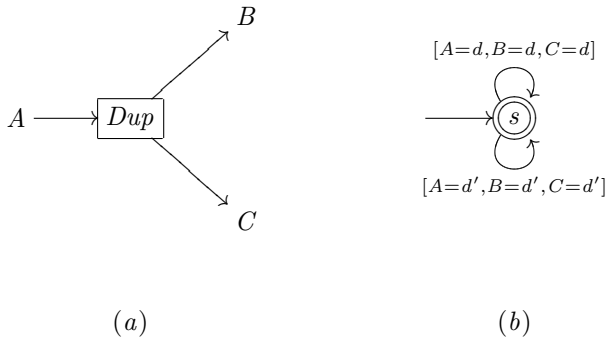


Figure 4.5: A duplicator channel and its BAR model

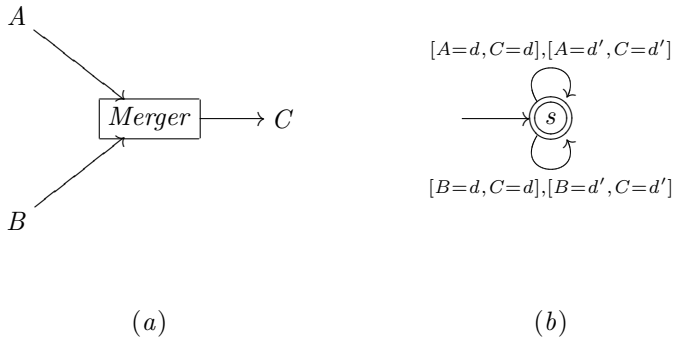


Figure 4.6: An (unfair) merger channel and its BAR model

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 4.14 In Figure 4.7 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 [66], we can construct a Büchi automaton recognizing $L_{vis}(B)$.

Lemma 4.3 For every Büchi automaton of records B there is a Büchi automaton of records B' (without τ-transition) such that, $L_{vis}(B) = L(B')$.

Proof. Let $B = \langle Q, \Sigma, \Delta, Q_0, F\rangle$ be the Büchi automaton of records over the alphabet $\Sigma = Rec_{\mathcal{N}}(\mathcal{D})$. Using B, we construct the following BAR without τ-transitions:

$$B' = \langle Q', \Sigma', \Delta', Q_0', F'\rangle$$

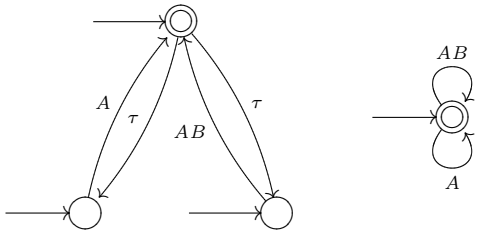


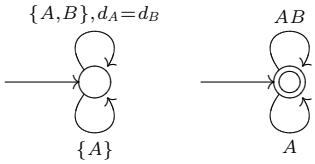
Figure 4.7: Two visibly equivalent Büchi automata of records.

such that,

- $Q = Q'$,
- $\Sigma' = \Sigma - \{\tau\}$,
- $Q_0 = Q'_0$,
- $F' = F \cup \{q | \exists q_F \in F, (q \xrightarrow{\tau^+} q_F) \in \Delta\}$,
- $(q, a, q') \in \Delta' \iff (q \xrightarrow{\tau^* a \tau^*} q') \in \Delta$

where, by $(q \xrightarrow{\tau^+} q_F) \in \Delta$ we mean that using the transition relation Δ there is a finite path π from q to q_F such that $\pi = q \xrightarrow{\tau} q_F$ or for $k \geq 1$, $\pi = q \xrightarrow{\tau} q_1 \xrightarrow{\tau} \ldots q_k \xrightarrow{\tau} q_F$; and by $(q \xrightarrow{\tau^* a \tau^*} q') \in \Delta$ we mean that there is a finite path π from q to q' such that there are states q_1 and q_2 (not necessarily distinct from each other or from q and q') where, $\pi = q \xrightarrow{\tau^*} q_1 \xrightarrow{a} q_2 \xrightarrow{\tau^*} q'$ in B.

Now, we can show that $L_{vis}(B) = L(B')$. Obviously, $L_{vis}(B) \subseteq (\Sigma')^\omega$. First, let $\rho \in L_{vis}(B)$, there is $\omega \in \Sigma^\omega$ in $L(B)$ such that $\rho = vis(\omega)$. Thus, there is an accepting infinite computation $\pi = q_0, a_0, q_1, a_1, \ldots$ such that at least one of the accepting states, say $q_F \in F$, occurs infinitely often (looping style) in π and $\omega = a_0 a_1 \ldots$. Consider π' as the computation obtained by replacing all finite subcomputations of the form $q_i \quad q_j$ in π with q_i. Because both ρ and ω are infinite words, by the definition of B', π' is an accepting computation for ρ in B'. Thus, $\rho \in L(B')$. Conversely, suppose that $\rho \in L(B')$. Thus, there is an accepting infinite computation $\pi' = q'_0, a_0, q'_1, a_1, \ldots$ in B'. Using the definition of δ', for every triple (q_i, a, q_j) in computation π' there is a computation fragment $q_i \xrightarrow{\tau^* a \tau^*} q_j$ in B. Replace all triples of the form (q_i, a, q_j) in computation π' with one of the corresponding computation fragments $q_i \xrightarrow{\tau^* a \tau^*} q_j$ and call the resulting computation π. Obviously, using the definitions of Δ' and F', it is necessary that π be an accepting infinite computation for an infinite word $\omega \in \Sigma^\omega$ such that $\rho = vis(\omega)$. Thus, $\omega \in L(B)$ and $\rho \in L_{vis}(B)$. □



(a) (b)

Figure 4.8: Models of a *non-deterministic* lossy synchronous channel by a) a constraint automaton and b) a Büchi automaton of records.

4.3.3 Recasting Constraint Automata into Büchi Automata

Now we show that for every constraint automaton A over a name set $\mathcal{N}$ and a 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 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 4.9 For every constraint automaton $A = \langle Q, \mathcal{N}, \longrightarrow, Q_0 \rangle$ over a finite data set $\mathcal{D}$ and a finite name set $\mathcal{N}$, we define $B(A)$ to be the Büchi automaton of records $\langle Q, Rec_{\mathcal{N}}(\mathcal{D}), \Delta, Q_0, F \rangle$, where $F = Q$ and Δ is the following set of transitions:

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

Example 4.15 Consider the constraint automaton depicted in Figure 4.8(a). It models a non-deterministic lossy synchronous channel from the source A to the sink B: data in $\mathcal{D}$ either flow from A to B or they get lost after they are read by A [30]. Figure 4.8(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 4.4 are obtained as the translations of the constraint automata models of the same channels in Figure 3.3. Note that in Figure 4.4 the data set is $\mathcal{D} = \{d, d'\}$.

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

Theorem 4.4 Let $A = \langle Q, \mathcal{N}, \longrightarrow, Q_0 \rangle$ be a finite constraint automaton. Then,

$$\Upsilon(L_{TDS}(A)) = L(B(A)) \text{ and } \Theta(L(B(A))) = L_{TDS}(A).$$

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 all whose 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 where each tuple (q_i, r_i, q_{i+1}) is a transition in $B(A)$. By construction, 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 the constraint automaton A and that for all TDS-tuples $\theta \in TDS^{\mathcal{N}}$ 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, $r \in \Upsilon(L_{TDS}(A))$ and $L(B(A)) \subseteq \Upsilon(L_{TDS}(A))$.

Conversely, let $r = r_0, r_1, \cdots$ be a stream of records in $\Upsilon(L_{TDS}(A))$. 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 $\theta \in L_{TDS}(A)$, there is a computation $\pi = q_0, (N_0, g_0), q_1, (N_1, g_1), \cdots$ in the constraint automaton A, starting from an initial state q_0 where $\theta.N(i) = N(i)$ and $\theta.\delta(i) \models g_i$, for all $i \geq 0$. By construction, there is a 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 $\Upsilon(L_{TDS}(A)) \subseteq L(B(A))$.

Next we prove the rightmost equality. Let $\theta \in TDS^{\mathcal{N}}$ be a timed data stream accepted by the constraint automaton A, that is $\theta \in L_{TDS}(A)$. 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, $L_{TDS}(A) \subseteq \Theta(L(B(A)))$.

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 the 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 $\theta \in L_{TDS}(A)$. Therefore, $\Theta(L(B(A))) \subseteq L_{TDS}(A)$. □

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 (language) complement while constraint automata are not.

4.4 Modeling Fair Reo Connectors

As we mentioned in the introduction, for several connectors we can consider some fairness conditions. In this section, we present some useful *fair* connectors that can be modeled by

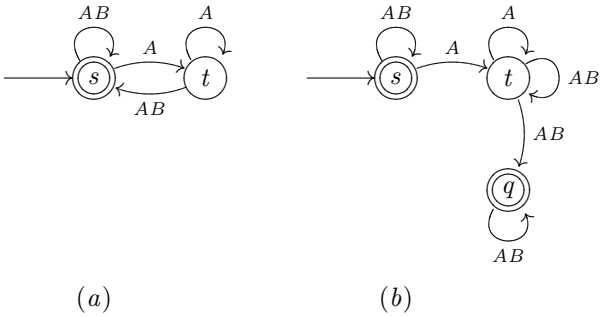


Figure 4.9: Models of a fair non-deterministic lossy synchronous channel with a) a weak fairness condition, b) a strong fairness condition.

Büchi automata of records.

Example 4.16 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 4.9.a. It is a connector similar to the non-deterministic lossy synchronous channel depicted in Figure 4.8.b but with this extra property that not all data can get lost. Still infinitely many data can get lost, while the non-deterministic lossy synchronous channel modeled by Büchi automaton of records in Figure 4.9.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* [101]: in each infinite execution of the system, some actions should occur infinitely many times.

Example 4.17 Consider the *merger* connector with two source ports A and B and one sink port B (see Figure 4.6(a)). Intuitively, it transmits synchronously a data item from either A or B to the port C. If both the source ports A and B offer data at the same time then only one of them is chosen non-deterministically. The Büchi automaton of records over the data set $\mathcal{D} = \{d\}$ corresponding to the constraint automaton model of merger introduced in [30] is shown in Figure 4.10(a). Both models allow unfair executions where data from the same source is always preferred if both A and B always offer data simultaneously. Figure 4.10(b) 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 [30].

4.5 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 BARs. We will give few examples in the following section.

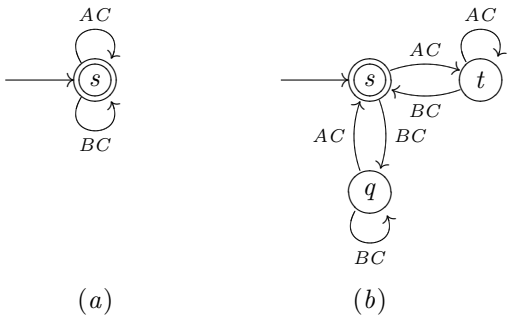


Figure 4.10: Models of a merger connector: (a) unfair version, (b) fair version

4.5.1 Product and Join

Since BARs 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.

Recall the definition of the product of Büchi automata which, for simplicity, we give in terms of generalized Büchi automata as defined in Definition 4.6:

Definition 4.10 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:

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

The language of the product of two Büchi automata is the intersection of their respective languages [138].

Note that 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 $L(B) = L(B')$ (see Lemma 4.2) .

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

Definition 4.11
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

BARs. We define the *join* of B_1 and B_2 as the *generalized* Büchi automaton $B_1 \bowtie B_2$ given by:

$$B_1 \bowtie B_2 = \langle Q_1 \times Q_2, Rec_{\mathcal{N}_1 \cup \mathcal{N}_2}(\mathcal{D}), \longrightarrow, Q_{01} \times Q_{02}, \{F_1 \times Q_2, Q_1 \times F_2\} \rangle$$

where the transition relation $\longrightarrow$ is defined by the following rules:
Rule 1:

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

Rule 2:

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

and dually,

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

where by the proposition $comp(r_1, r_2)$ we mean that records r_1 and r_2 are compatible (see Definition 4.3).

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

Example 4.18 For example, consider Figure 4.11. Figure 4.11(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 Figure 4.11(c).

For Büchi automata without τ-transitions, the join operator coincides with the product in case both automata have the same alphabet. In this case, the language of the product is just the intersection of the languages of the two automata.

Lemma 4.5 Let B_1 and B_2 be two Büchi automata of records with the same alphabet $\Sigma = Rec_{\mathcal{N}}(\mathcal{D})$ (over the same data sets and the same port sets). Then,

$$L_{vis}(B_1 \bowtie B_2) = L_{vis}(B_1) \cap L_{vis}(B_2).$$

Proof. Let B_1' and B_2' be BARs without τ-transitions, respectively, the visibly equivalents of B_1 and B_2 after applying the τ-transitions elimination procedure that we introduced in the proof of Lemma 4.3. We know that for $i \in \{1, 2\}$, $L_{vis}(B_i) = L(B_i')$. Thus, it is enough to show that $L_{vis}(B_1 \bowtie B_2) = L(B_1' \times B_2')$.

Let $\omega \in L(B_1' \times B_2')$. Thus, ω has no τ symbol and $\omega \in L(B_1') \cap L(B_2')$. Because there is an accepting computation for ω in B_1', there is an infinite word $\rho_1 \in \Sigma^\omega$ such that $\rho_1 \in L(B_1)$ and $\omega = vis(\rho_1)$. Similarly, there is an infinite word $\rho_2 \in \Sigma^\omega$ such that $\rho_2 \in L(B_1)$ and $\omega = vis(\rho_2)$. Because ρ_1 and ρ_2 are visibly equivalent (namely, ignoring all τ symbols, both

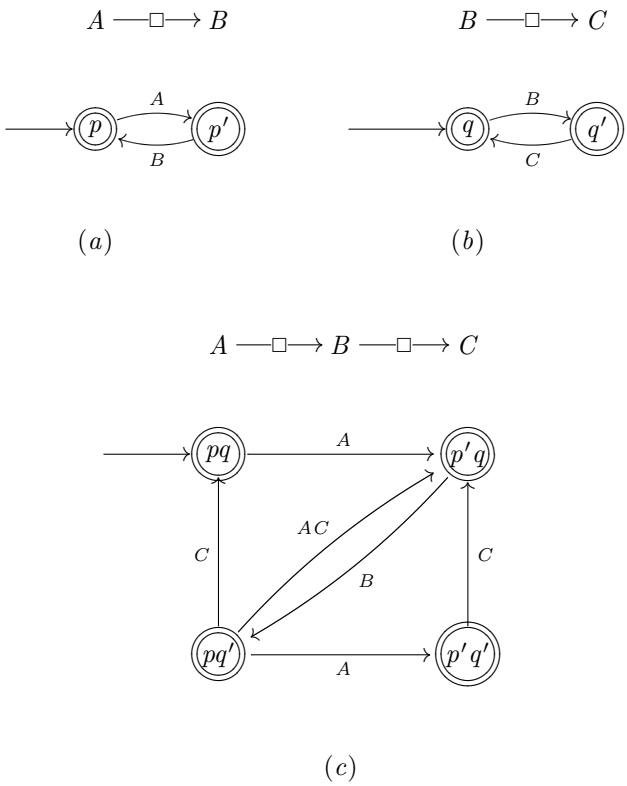


Figure 4.11: Composing two FIFO1 channels

become the same infinite word) and both are over the same alphabet, the first rule of the join operation (see Rule 1 in Definition 4.11) is applicable only on two τ-transitions or two transitions with the exact same labels (a τ-transition can not be synchronized with a transition with a label other than τ). Similarly, Rule 2 is applicable only on τ-transitions. Thus, there is an accepting infinite word $\rho_3 \in L(B_1 \bowtie B_2)$ such that $vis(\rho_3) = vis(\rho_2) = vis(\rho_1) = \omega$. Therefore, $\omega \in L_{vis}(B_1 \bowtie B_2)$.

Conversely, suppose that $\omega \in L_{vis}(B_1 \bowtie B_2)$. Thus, there is $\rho \in L(B_1 \bowtie B_2)$ such that $\omega = vis(\rho)$. Because B_1 and B_2 are over the same alphabets (same set of data and same set of names), again in the join operation, Rule 1 can be applied only on two τ-transitions or on two transitions with the exact same labels (a τ-transition cannot be synchronized with a transition with a label other than τ) and Rule 2 is applicable only on τ-transitions. Thus, there is an infinite word, say $\rho_1 \in L(B_1)$, that $vis(\rho_1) = vis(\rho)$. Similarly, there is an infinite word, say $\rho_2 \in L(B_2)$, that $vis(\rho_2) = vis(\rho)$. Thus, $\omega \in L(B_1') \cap L(B_2')$. Therefor, $\omega \in L(B_1' \times B_2')$. □

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 4.6 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 $A_1 = \langle Q_1, \mathcal{N}_1, T_1, Q_{01}\rangle$ and $A_2 = \langle Q_2, \mathcal{N}_2, T_2, Q_{02}\rangle$. Using Definition 3.8

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

where T is the set of all transitions obtained using rules presented in Definition 3.8. Using Definition 4.9, $B(A_1 \bowtie_C A_2)$ 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 the transition $(\langle s, t\rangle, N, g, \langle s', t'\rangle) \in T$ and $\delta{:}N \to D$ such that $\delta \models g$ and for all n in N, $r.n = \delta(n)$.

Further, let

$$B(A_1) = \langle Q_1, Rec_{\mathcal{N}_1}(\mathcal{D}), \Delta_1, Q_{01}, Q_1\rangle \text{ and } B(A_2) = \langle Q_2, Rec_{\mathcal{N}_2}(\mathcal{D}), \Delta_2, Q_{02}, Q_2\rangle$$

with Δ_1 and Δ_2 obtained as described in Definition 4.9. Using Definition 4.11, $B(A_1) \bowtie B(A_2)$ 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 4.11. We need to prove that $\Delta_C = \Delta_B$

First, we prove $\Delta_C \subseteq \Delta_B$. 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 $\delta{:}N \to \mathcal{D}$, such that $\delta \models g$ and $\forall n \in N, r.n = \delta(n)$. We have three cases:

1) If $(\langle s, t\rangle, N, g, \langle s', t'\rangle) \in T$ is obtained using the first rule in Definition 3.8, 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 the 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 the 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 4.11, $(\langle s, t\rangle, r, \langle s', t'\rangle) \in \Delta_B$.

2) If $(\langle s, t\rangle, N, g, \langle s', t'\rangle) \in T$ is obtained using the second rule in Definition 3.8, then, there is $(s, N, g, s') \in T_1$ such that, $t = t'$ and $N \cap \mathcal{N}_2 = \emptyset$. Thus, based on the definition of Δ_1, we have $(s, r, s') \in \Delta_1$. Because $dom(r) \subseteq N$, therefore, $dom(r) \cap \mathcal{N}_2 = \emptyset$. Using Rule 2 in Definition 4.11, we conclude that $(\langle s, t\rangle, r, \langle s', t'\rangle) \in \Delta_B$.

3) If $(\langle s, t\rangle, N, g, \langle s', t'\rangle) \in T$ is obtained using the third rule in Definition 3.8. The proof is similar to the previous case because the third rule is the dual of the second one.

It remains to prove $\Delta_B \subseteq \Delta_C$. Let $(\langle s, t\rangle, r, \langle s', t'\rangle) \in \Delta_B$. We have three cases:

1) If $(\langle s, t\rangle, r, \langle s', t'\rangle) \in \Delta_B$ is obtained using the first rule in Definition 4.11, then, there are

$(s, r_1, s') \in \Delta_1$ and $(t, r_2, t') \in \Delta_2$ 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 the 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 $\delta_1{:}N_1 \rightarrow \mathcal{D}$ and $\delta_2{:}N_2 \rightarrow \mathcal{D}$ 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 $dom(r_1) \cap \mathcal{N}_2 = dom(r_2) \cap \mathcal{N}_1$, therefore, $N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1$ and using the first rule in Definition 3.8, we have $(\langle s, t\rangle, N, g, \langle s', t'\rangle) \in \Delta_B$. 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 4.11, then, there is a $(s, r, s') \in \Delta_1$ such that $t = t'$ and $dom(r) \cap \mathcal{N}_2 = \emptyset$. Based on the definition of Δ_1, there is a $(s, N, g, s') \in T_1$ and there are data assignments $\delta{:}N \rightarrow \mathcal{D}$ 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 3.8, we have $(\langle s, t\rangle, N, g, \langle s', t'\rangle) \in \Delta_B$. Thus,$(\langle s, t\rangle, r, \langle s', t'\rangle) \in \Delta_C$.
3) Again, the remaining case can be treated similarly. □

4.5.2 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 automaton of records B with an extra port name, not necessarily present in the alphabet of B. If the port is new, the resulting automaton will have to guess the right behavior non-deterministically, by allowing or not the simultaneous exchange of data with the other ports known to the automaton.

Definition 4.12 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

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

Note that in forthcoming discussions and proofs sometimes we refer to the second and third component-sets of $\widehat{\Delta}$ by Δ' and Δ''. Namely, we define $\widehat{\Delta} = \Delta \cup \Delta' \cup \Delta''$ 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 with 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 kinds of guess: the environment does not use the name n in a communication (explaining why $\Delta \subseteq \widehat{\Delta}$); or the environment uses the name n for a communication but no other port of B is

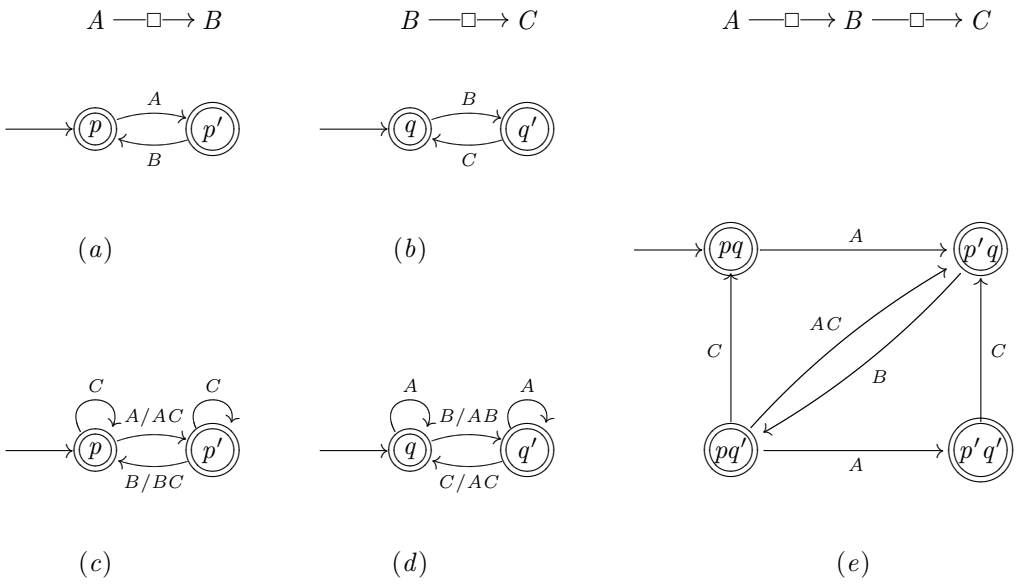


Figure 4.12: Direct and indirect joining of two FIFO1 buffers

used (explaining the addition of a new loop transition on each state labeled by a record with n as its only name in the domain); or the environment uses 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 4.19 For example, in Figure 4.12(c) we show the extension of the automaton has been shown in Figure 4.12(a) with respect to the new port name C. In this figure, $p \xrightarrow{A/AC} p'$ means that there are two transitions $p \xrightarrow{A} p'$ and $p \xrightarrow{AC} p'$. Also, Figure 4.12(d) is the extension of Figure 4.12(b) with 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 names 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 the result of the join of the two Büchi automata B_1 and B_2.

Theorem 4.7 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,

$$B_1{\uparrow}\mathcal{N}_2 \times_B B_2{\uparrow}\mathcal{N}_1 = B_1 \bowtie_B B_2.$$

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 4.11, $B_1 \bowtie B_2$ 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 Definition 4.12, 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 Definition 4.10. We need to prove $\Delta_{\times} = \Delta_{\bowtie}$. We start by showing that $\Delta_{\times} \subseteq \Delta_{\bowtie}$:
Let $(\langle s, t\rangle, r, \langle s', t'\rangle) \in \Delta_{\times}$. Using Definitions 4.10 and 4.12 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 4.11, 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 $r \in Rec_{\mathcal{N}_1 \setminus \mathcal{N}_2}(\mathcal{D})$. Thus, $dom(r) \cap \mathcal{N}_2 = \emptyset$. Therefore, using the second rule in Definition 4.11, $(\langle s, t\rangle, r, \langle s', t\rangle)$ is in $\Delta_{\bowtie}$.
3) $(s, r, s') \in \Delta_1$ and $(t, r, t') \in \Delta_2''$. According to the definition of Δ_2'', there is a $(t, r', t') \in \Delta_2$ such that $dom(r) = dom(r') \cup N'$ for some $N' \subseteq \mathcal{N}_1 \setminus \mathcal{N}_2$ and $\forall n \in dom(r'){:}r(n) = r'(n)$. Therefore, $dom(r) \cap \mathcal{N}_2 = dom(r') \cap \mathcal{N}_1 = dom(r')$, r and r' are compatible and $r \cup r' = r$. Thus, using Definition 4.11 Rule 1, $(\langle s, t\rangle, r, \langle s', t'\rangle) \in \Delta_{\bowtie}$.
4) $(s, r, s') \in \Delta_1'$ and $(t, r, t') \in \Delta_2$. The proof of this case is symmetric to the proof of case 2.
5) $(s, r, s') \in \Delta_1'$ and $(t, r, t') \in \Delta_2'$. This case is impossible, because, by the definition of Δ_1', $dom(r) \subseteq \mathcal{N}_2 \setminus \mathcal{N}_1$ and by definition of Δ_2', $dom(r) \subseteq \mathcal{N}_1 \setminus \mathcal{N}_2$ 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 symmetric to the 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 the definition of Δ'', there are records r' and r'' such that $dom(r) = dom(r') \cup N' = dom(r'') \cup N''$ for $N' \subseteq \mathcal{N}_2 \setminus \mathcal{N}_1$ and $N'' \subseteq \mathcal{N}_1 \setminus \mathcal{N}_2$. 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 $\forall n \in dom(r''){:}r(n) = r''(n)$,

we have $r = r' \cup r''$. Thus, using Definition 4.11, 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 4.11, 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 4.11, there is a $(s, r, s') \in \Delta_1$ such that $dom(r) \cap \mathcal{N}_2 = \emptyset$ and $t = t'$. Because $r \in Rec_{\mathcal{N}_1}(\mathcal{D})$ and $dom(r) \cap \mathcal{N}_2 = \emptyset$, we have $r \in Rec_{\mathcal{N}_1 \setminus \mathcal{N}_2}(\mathcal{D})$. Based on the definition of Δ', $(t, r, t) \in \Delta_2'$. Thus $(s, r, s') \in \widehat{\Delta_1}$ and $(t, r, t') \in \widehat{\Delta_2}$. Therefore, using the definition of Büchi product, $(\langle s, t\rangle, r, \langle s', t'\rangle) \in \Delta_{\times}$. □

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 4.20 The join of the Büchi automata of records shown in Figures 4.12(a) and (b) is the automaton shown in 4.12(e). This automaton, in turn, is the *product* of the automata depicted in Figures 4.12(c) and 4.12(d). The resulting automaton models a two-cell queue. 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 item through the port A.

4.5.3 Hiding of Port Names

The effect of hiding a port of a component connector is that data flow through that node is no longer observable. In BARs, the hiding operator removes all information about the hidden port.

Definition 4.13 Let $B = \langle Q, Rec_{\mathcal{N}}(\mathcal{D}), \rightarrow, Q_0, F\rangle$ be a BAR or generalized BAR. The hiding of a port name $A \in \mathcal{N}$ from B is the following BAR or generalized 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 $q \overset{r}{\longrightarrow} p$.

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.

The hiding operation is interesting when it is used after joining the Büchi automata of records that model some Reo connectors. In such cases, we are generally interested to hide the common or intermediate port names. In other words, by joining of connectors, we normally construct more complicated connectors in which the common ports of the elementary connectors become internal nodes and the other ports become the interfaces of the new connector.

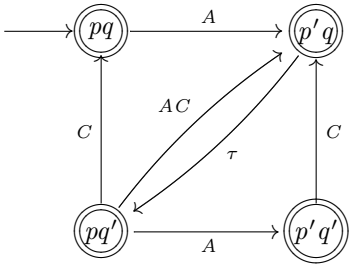


Figure 4.13: The resulting BAR after hiding B in Figure 4.12(e).

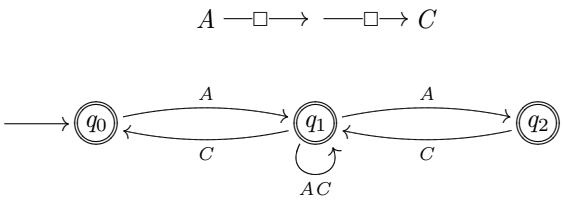


Figure 4.14: The resulting BAR after eliminating τ-transitions in Figure 4.13.

Thus, after joining of connectors, we can hide the effects of the common (intermediate) ports. Let us have an example:

Example 4.21 Figure 4.12(e) shows a BAR which is the resulting automaton after joining the two automata models of two FIFO1 channels (one from port A to port B and the other from port B to port C) that were illustrated in Figures 4.12(a) and (b). Now, B is an intermediate port. We hide port name B in the BAR illustrated in Figure 4.12(e). The resulting automaton is illustrated in Figure 4.13.

Now, we convert the generalized BAR model in Figure 4.13 to an equivalent BAR without τ-transition using the constructions that we introduced in the proofs of Lemma 4.2 (to convert generalized BAR to ordinary BAR) and Lemma 4.3 (to eliminate τ-transitions). The reachable part of the resulted automaton after conversion is illustrated in Figure 4.14.

As we expect, Figure 4.14 is a model of the visible behavior of a FIFO channel whose buffer capacity is two (say, FIFO2). States q_0, q_1 and q_2, respectively, represent the configurations of the connector where the buffer is empty, the buffer contains one data item, and the buffer is full.

4.6 Fair Constraint Automata

As we explained earlier, the timed data streams based semantics of constraint automata is more concrete than necessary. On the other hand, languages of streams of records that we have used as the semantics of BARs are more understandable and a more suitable semantics for connectors. In this section, we introduce a version of constraint automaton, called *fair*

constraint automaton, whose syntax is the same as constraint automaton except that now it has final (accepting) states, but its semantics is based on the languages of streams of records. As for the case of Büch automata, by adding sets of final states to constraint automata, *unconditional* fairness constraints over sets of states/transitions become expressible.

Definition 4.14 Let $\mathcal{D}$ be a fixed finite set of data. A *fair constraint automaton* (abbreviated as FCA) over a data set $\mathcal{D}$ is of the form $C = \langle Q, \mathcal{N}, \longrightarrow, Q_0, F \rangle$ where:

- Q is a finite set of states,
- $\mathcal{N}$ is a finite set of names,
- $\longrightarrow \subseteq\ Q \times 2^{\mathcal{N}} \times DC \times Q$ is a set of transitions, where, DC is the set of all data constraints over names set $\mathcal{N}$ and data set $\mathcal{D}$ as defined in Definition 3.5,
- $Q_0 \subseteq Q$ is the set of initial states,
- $F \subseteq Q$ is the set of accepting (final) states.

We write $p \xrightarrow{N,g} q$ instead of $\langle p, N, g, q \rangle \in \longrightarrow$ and call N the name set and g the guard of the transition.

A fair constraint automaton $C = \langle Q, \mathcal{N}, \longrightarrow, Q_0, F \rangle$ is *deterministic* if $|Q_0| \leq 1$ and for every state q, set of port names N, and data assignment $\delta{:}N \rightarrow \mathcal{D}$, there is at most one transition $q \xrightarrow{N,g} q'$ with $\delta \models g$.

Now, we define the semantics of FCAs using the languages of streams of records.

Definition 4.15 Let $C = \langle Q, \mathcal{N}, \longrightarrow, Q_0, F \rangle$ be an FCA over a data set $\mathcal{D}$. Also, suppose that $\Sigma = Rec_{\mathcal{N}}(\mathcal{D})$ and $\omega \in \Sigma^{\omega}$.
1- An *infinite computation* in C is an infinite sequence of alternating states and data constraints of the form $\pi = q_0, (N_0, g_0), q_1, (N_1, g_1), \ldots$ where $q_0 \in Q_0$ and for all subsequence $q_i, (N_i, g_i), q_{i+1}$ in π, $\langle q_i, N_i, g_i, q_{i+1} \rangle \in \longrightarrow$.
2- An infinite computation $\pi = q_0, (N_0, g_0), q_1, (N_1, g_1), \ldots$ is a *computation for the stream* $\omega = r_0 r_1 \ldots \in \Sigma^{\omega}$ if and only if for all $i \geq 0$, there is a data assignment $\delta{:}N_i \rightarrow \mathcal{D}$ such that $\delta \models g_i$, $dom(r_i) = N_i$ and $\forall n \in N_i{:}\ r.n = \delta(n)$.
3- Let $\pi = q_0, (N_0, g_0), q_1, (N_1, g_1), \ldots$ be a computation for the stream $\omega \in \Sigma^{\omega}$. We say that π is an *accepting computation* for ω if at least one of the accepting states, say $q_F \in F$, occurs infinitely many times in π.
4- We define the language of FCA C as follows:

$$L(C) = \{\omega \in Rec_{\mathcal{N}}(\mathcal{D})\omega \mid \text{there is an accepting computation } \pi \text{ for } \omega \text{ in } C.\}$$

5- Two FCAs C_1 and C_2 are equivalent if $L(C_1) = L(C_2)$.
6- The notions of *generalized fair constraint automata* (GFCA) and their accepted languages are defined similarly such as generalized Büchi automata and their accepted languages.

Obviously, because the semantics of both Büchi automaton of records and unconditional fair constraint automaton are based on languages of streams of records, each BAR B over

a name set $\mathcal{N}$ and a data set $\mathcal{D}$ can be considered as an FCA if we replace every transition label $r \in Rec_{\mathcal{N}}(\mathcal{D})$ with (N, g) where, $N = dom(r)$ and g is the data constraint $\bigwedge_{n \in dom(n)}(d_n = r.n)$. Using this simple conversion of BAR into FCA, we can show that all BARs that we introduced as models of Reo connectors can be considered as FCA models for them.

Conversely, if C is an FCA over a finite name set $\mathcal{N}$ and a finite data set $\mathcal{D}$ then C is equivalent with a BAR B all whose components are the same as C except that each transition $q \xrightarrow{N,g} q'$ of C is replaced with a set of transitions of the form $q \xrightarrow{r} q'$ where r satisfies the following conditions: $dom(r) = N$ and there exists a data assignment $\delta{:}N \to \mathcal{D}$ such that $\delta \models g$ and $\forall n \in N,\ \delta(n) = r.n$.

Every constraint automaton A can be converted to an FCA $C(A)$ all of whose states are accepting and with the same components as A. By this conversion, we have the following direct consequence:

Theorem 4.8 Let $A = \langle Q, \mathcal{N}, \longrightarrow, Q_0 \rangle$ be a finite constraint automaton over a data set $\mathcal{D}$. For its corresponding FCA $C(A) = \langle Q, \mathcal{N}, \longrightarrow, Q_0, F \rangle$ over the same data set $\mathcal{D}$ in which all shared components are the same and $F = Q$, we have:

$$\Upsilon(L_{TDS}(A)) = L(C(A)) \text{ and } \Theta(L(C(A))) = L_{TDS}(A).$$

Proof. Let B be the BAR all whose components are the same as $C(A)$ except that each transition $q \xrightarrow{N,g} q'$ of $C(A)$ is replaced with a set of transitions of the form $q \xrightarrow{r} q'$ where r satisfies the following conditions: $dom(r) = N$ and there exists a data assignment $\delta{:}N \to \mathcal{D}$ such that $\delta \models g$ and $\forall n \in N,\ \delta(n) = r.n$. Obviously, we have $L(B) = L(C(A))$, and using Definition 4.9, B is the corresponding BAR for A. Based on Theorem 4.4, $\Upsilon(L_{TDS}(A)) = L(B)$ and $\Theta(L(B)) = L_{TDS}(A)$. Thus, the result holds. □

Same as the case of constraint automata, fair constraint automata can be composed by a join operator:

Definition 4.16 Let $C_1 = \langle Q_1, \mathcal{N}_1, \longrightarrow_1, Q_{01}, F_1 \rangle$ and $C_2 = \langle Q_2, \mathcal{N}_2, \longrightarrow_2, Q_{02}, F_2 \rangle$ be two FCAs both over the set of data $\mathcal{D}$. The *join* of C_1 and C_2 is the GFCA:

$$C_1 \bowtie C_2 = \langle Q_1 \times Q_2, \mathcal{N}_1 \cup \mathcal{N}_2, \longrightarrow, Q_{01} \times Q_{02}, \mathcal{F} \rangle$$

where, the transition relation $\longrightarrow$ is exactly as defined for the join of constraint automata (see Definition 3.8) and the set of sets of accepting states is

$$\mathcal{F} = \{Q_1 \times F_2, F_1 \times Q_2\}.$$

Based on the above definition, we have the following theorem:

Theorem 4.9 Let $C_1 = \langle Q_1, \mathcal{N}, \longrightarrow_1, Q_{01}, F_1 \rangle$ and $C_2 = \langle Q_2, \mathcal{N}, \longrightarrow_2, Q_{02}, F_2 \rangle$ be two FCAs both over the same set of names $\mathcal{N}$ and the same set of data $\mathcal{D}$. Then,

$$L_{vis}(C_1 \bowtie C_2) = L_{vis}(C_1) \cap L_{vis}(C_2).$$

Proof. The theorem is a direct consequence of Lemma 3.1(2), Lemma 4.5, and Theorem 4.8. □

Also, the hiding operator over FCAs is the same as the hiding operator over constraint automata (see Definition 3.10).