A Compendium of Reo Connectors

Compiled by
David Costa and Dave Clarke

October 22, 2005

Abstract

Reo is an exogenous coordination language based on connectors formed by joining channels into circuit-like configurations. The resulting circuits coordinate the flow of data between components connected to the input and output ends of the connector. This document collects together the Reo channels and connectors devised to date. By collecting existing connectors together in a single place, this document provides a useful resource when developing Reo connectors and for generating examples for further developing the tools and techniques surrounding Reo.

Contents

1 Introduction

Reo is a channel-based component coordination model. It has semantics based on co-inductively defined abstract behaviour types (ABT) and constraint automata (CA). We defer discussion of these models until future versions of this document, and suggest that the reader consult the relevant literature on ABTs [4] and CAs [7]. These will appear in Section 2 and 3.

We begin our list of connectors by describing first the basic channels, along with some more exotic ones in Section 4. Section 5 follows with an extensive list of connectors. In future versions of this work, we will include a series of examples in Section 6 and the encoding of a number of software architectures in Section **??**.

2 Abstract Behaviour Types

TODO.

Note that for streams a' gives the tail of the stream. This operation is called the *derivative*. $a^{(k)}$ denotes the k-th derivative of the stream a.

3 Constraint Automata

TODO.

CAs are automata which has edges annotated by a set of node names and a data constraint over that set. The intended meaning is that data flows simultaneously among the nodes in the set (and only among those nodes) and that the data satisfies the given data constraint. Constraint automata abstract away from the direction of the data flow.

4 Channels

Reo assumes the availability of an arbitrary set of channel types, each with its own well-defined behaviour. A channel is a point-to-point medium of communication with its own unique identity and two distinct ends. A channel itself has no direction. There are two types of channel ends: sources and sinks. A source channel end accepts data into its channel. A sink channel end dispenses data out of its channel. Channels are dynamically created and are automatically garbage collected, i.e., they are not explicitly destroyed, when no longer required. Channels are assumed to be mobile, though this has no effect on their intrinsic semantics.

Every channel in Reo has exactly two ends, which may be of the same or different types. Thus, a channel may have a source and a sink end, two source ends, or two sink ends. The behaviour of a channel may depend on such parameters as its synchronizing properties, the number of its source and sink ends, the size of its buffer, its ordering scheme, its lossy policy, etc. There are a number of different properties which (partially) characterise the behaviour of channels. A channel is called *synchronous* if it delays the success of the appropriate pairs of operations on its two ends so that they can succeed only simultaneously[1]; otherwise, it is called *asynchronous*. An asynchronous channel may have a bounded or an unbounded buffer (to hold the data items it has already consumed through its source, but not yet dispensed through its sink) and need not impose an order on the delivery of its contents. A *lossy* channel may deliver only some of the data items that it receives, and lose the rest.

[1]Note that by simultaneity we really mean atomicity. That is, if we say that an event occurs simultaneously on a number of channel ends (or equivalently, nodes), then we mean that the event cannot be interleaved with other events on any of the ends involved [4].

4.1 Synchronous Channel — Sync

Description A synchronous, unbuffered, and ordered channel, with one sink and one source end [1].

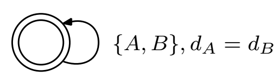
Circuit The `Sync` channel is graphically represented by

ABT The synchronous channel, $\longrightarrow$, is defined, for all timed data streams $\langle\alpha, a\rangle$ and $\langle\beta, b\rangle$, by

$$\langle\alpha, a\rangle \longrightarrow \langle\beta, b\rangle \;\equiv\; \alpha = \beta \;\wedge\; a = b \;.$$

The `Sync` channel produces an output data stream identical to its input data stream ($\alpha = \beta$), and reproduces every element in its output at the same time as its respective input element is consumed ($a = b$).

Constraint Automata The deterministic constraint automaton for the synchronous channel is:



4.2 Synchronous Drain — SyncDrain

Description A synchronous, unbuffered, ordered, and lossy channel, with two source ends [1].

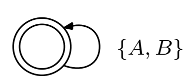
Circuit The `SyncDrain` channel is graphically represented by

ABT The `SyncDrain` channel, $\blacktriangleright\!\!\!-\!\!\!-\!\!\!\blacktriangleleft$, is defined for all timed data streams $\langle\alpha, a\rangle$ and $\langle\beta, b\rangle$, by

$$\blacktriangleright\!\!\!-\!\!\!-\!\!\!\blacktriangleleft(\langle\alpha, a\rangle, \langle\beta, b\rangle;\,) \;\equiv\; a = b \;.$$

It has no sink end, thus it produces no data items. Consequently, every data item written to its source ends is simply lost. `SyncDrain` is synchronous because a write operation on one of its ends remains pending until a write is performed on its other end; only then will both write operations succeed together.

Constraint Automata The deterministic constraint automaton for the synchronous drain channel is:



4.3 Synchronous Spout — SyncSpout

Description A synchronous, unbuffered, and ordered channel, with two sink ends [1].

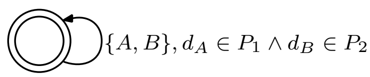
Circuit The `SyncSpout` channel with patterns P_1 and P_2 is graphically represented by

ABT The `SyncSpout` channel, ${}^{P_1}\!\!\longleftrightarrow\!\!{}^{P_2}$, is defined, for all timed data streams $\langle \alpha, a\rangle$ and $\langle \beta, b\rangle$, by

$$
{}^{P_1}\!\!\longleftrightarrow\!\!{}^{P_2}\ (;\langle \alpha, a\rangle, \langle \beta, b\rangle) \ \equiv \ \begin{cases} a(0) = b(0) \ \wedge \ \alpha(0) \ni P_1 \ \wedge \ \beta(0) \ni P_2 \\ {}^{P_1}\!\!\longleftrightarrow\!\!{}^{P_2}\ (;\langle \alpha', a'\rangle, \langle \beta', b'\rangle) \end{cases}
$$

It is an unbounded source of data items that match with its specified patterns, P_1, P_2 and can be taken from its opposite ends only simultaneously. Obviously data items are produced in a non-deterministic order and the data items taken out of the two sinks of this channel are not related to each other.

Constraint Automata The deterministic constraint automaton for the synchronous spout channel is



4.4 Lossy Synchronous Channel — LossySync

Description A synchronous, unbuffered, ordered, lossy channel, which has both a source and a sink end [1].

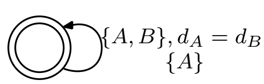
Circuit The `LossySync` channel is graphically represented by

ABT The `LossySync` is defined, $\dashrightarrow$, for all timed data streams $\langle \alpha, a\rangle$ and $\langle \beta, b\rangle$, by

$$
\langle \alpha, a\rangle \dashrightarrow \langle \beta, b\rangle \ \equiv \ \begin{cases} \beta(0) = \alpha(0) \ \wedge \ \langle \alpha', a'\rangle \dashrightarrow \langle \beta', b'\rangle & \text{if } a(0) = b(0) \\ \langle \alpha', a'\rangle \dashrightarrow \langle \beta, b\rangle & \text{otherwise} \end{cases}
$$

A `LossySync` is similar to a `Sync` channel except that it is always ready to consume a data item written to its source end. If a matching read operation is pending at its sink, the data item written to its source is transfered; otherwise, the written data item is lost.

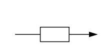
Constraint Automata The deterministic constraint automaton for the lossy synchronous channel is



4.5 FIFO channel with capacity of 1 – FIFO$_1$

Description An asynchronous, buffered and ordered channel, with a source and a sink end [1].

Circuit The `FIFO`$_1$ channel is graphically represented by

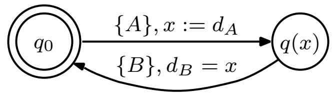


ABT `FIFO`$_1$ channel, $\multimap\!\!\square\!\!\rightarrow$, is defined, for all timed data streams $\langle \alpha, a\rangle$ and $\langle \beta, b\rangle$, by

$$
\langle \alpha, a\rangle \multimap\!\!\square\!\!\rightarrow \langle \beta, b\rangle \equiv \alpha = \beta \wedge a < b < a'
$$

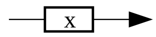
What goes in, comes out: $\alpha = \beta$, but later: $a < b$. Moreover, at any moment the next data item can be input only after the present data item has been output: $b < a'$, which is equivalent to $b(n) < a(n+1)$, for all $n \geq 0$.

Constraint Automata The deterministic parameterized constraint automaton[7] for $\texttt{FIFO}_1$ is

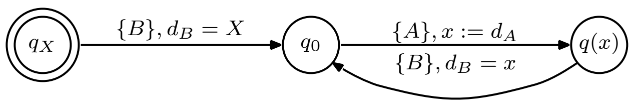


where $q(x)$ is used to denote that q is a location with parameter list $v(q) = x$, while q_0 is a location with an empty parameter list.

Variations $\texttt{FIFO}_X$ is a $\texttt{FIFO}_1$ buffer in which the buffer is initially non-empty. It contains the element X. Graphically represented by:



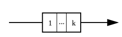
The parameterized constraint automaton for the $\texttt{FIFO}_X$ is in this case slightly different



4.6 FIFO channel with capacity of k — $\textbf{FIFO}_k$

Description An asynchronous, buffered and ordered channel which has a source and a sink end [1].

Circuit The $\texttt{FIFO}_\texttt{k}$ channel is graphically represented by

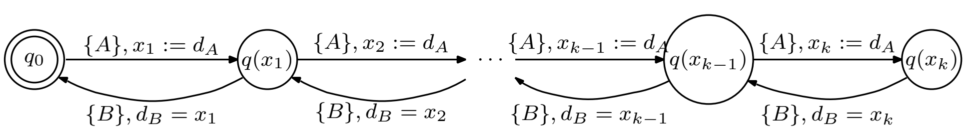


ABT $\texttt{FIFO}_\texttt{k}$ channel, is defined for any $k \geq 1$, for all timed data streams $\langle \alpha, a \rangle$ and $\langle \beta, b \rangle$, by

$$\langle \alpha, a \rangle \quad \langle \beta, b \rangle \equiv \alpha = \beta \wedge a < b < a(k)$$

This models a k-bounded FIFO buffer, generalizing the $\texttt{FIFO}_1$ buffer above. What goes in, comes out: $\alpha = \beta$, but later: $a < b$. Moreover, at any moment the kth-next data item can be input only after the present data item has been output: $b < a^{(k)}$ (which is equivalent to $b(n) < a(n + k)$, for all $n \geq 0$).

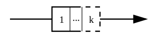
Constraint Automata The deterministic parameterized constraint automaton for the asynchronous bounded FIFO channel with capacity of k is:



4.7 Shift bounded FIFO channel – ShiftFIFO$_k$

Description An asynchronous, buffered, ordered, and lossy channel which has a source and a sink end [1].

Circuit The `ShiftFIFO`$_k$ channel is graphically represented by

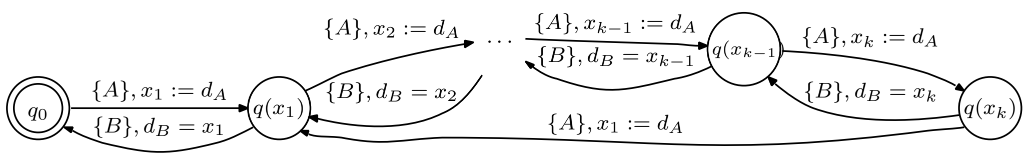


ABT The `ShiftFIFO`$_k$ channel, $\xrightarrow{\boxed{1|\cdots|k}}$, for any $k \geq 1$, is defined, for all timed data streams $\langle \alpha, a \rangle$ and $\langle \beta, b \rangle$, by

$$\langle \alpha, a \rangle \xrightarrow{\boxed{1|\cdots|k}} \langle \beta, b \rangle \equiv$$
$$\begin{cases} \alpha(0) = \beta(0) \;\wedge\; \langle \alpha', a' \rangle \xrightarrow{\boxed{1|\cdots|k}} \langle \beta', b' \rangle & \text{if } a(0) < b(0) < a(k) \\ \alpha(j) = \beta(0) \;\wedge\; \langle \alpha^j, a^j \rangle \xrightarrow{\boxed{1|\cdots|k}} \langle \beta', b' \rangle & \text{if } a(k+j-1) < b(0) < a(k+j),\; j > 0 \end{cases}$$

The `ShiftFIFO`$_k$ is the lossy version of `FIFO`$_k$, where the arrival of a data item when the channel buffer is full triggers the loss of the oldest data item in the buffer to make room for the new arrival.

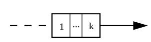
Constraint Automata The deterministic parameterized constraint automaton for the asynchronous bounded shift FIFO channel with capacity of k is



4.8 Lossy bounded FIFO channel – LossyFIFO$_k$

Description An asynchronous, buffered (with capacity k), ordered, and lossy channel with a source and a sink end [1].

Circuit `LossyFIFO`$_k$ is graphically represented by

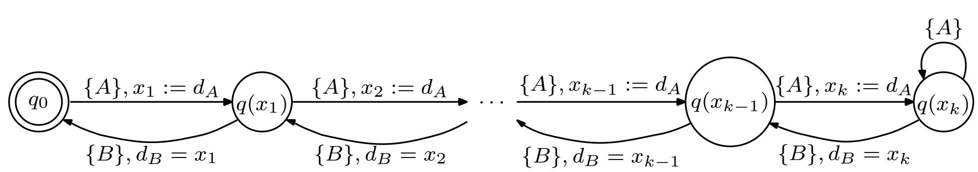


ABT The `LossyFIFO`$_k$ channel, $\dashrightarrow\!\boxed{1|\cdots|k}\!\rightarrow$, for any $k \geq 1$, is defined for all timed data streams $\langle \alpha, a \rangle$ and $\langle \beta, b \rangle$ by

$$\langle \alpha, a \rangle \dashrightarrow\!\boxed{1|\cdots|k}\!\rightarrow \langle \beta, b \rangle \equiv$$
$$\alpha(0) = \beta(0) \;\wedge\; \begin{cases} \langle \alpha', a' \rangle \dashrightarrow\!\boxed{1|\cdots|k}\!\rightarrow \langle \beta', b' \rangle \\ \qquad \text{if } a(0) < b(0) < a(k) \\ \langle \alpha(1).\cdots.\alpha(k).\alpha^{k+j},\; a(1).\cdots.a(k).a^{k+j} \rangle \dashrightarrow\!\boxed{1|\cdots|k}\!\rightarrow \langle \beta', b' \rangle \\ \qquad \text{if } a(k+j-1) < b(0) < a(k+j),\; j > 0 \end{cases}$$

The `LossyFIFO`$_k$ is another lossy version of `FIFO`$_k$, where data which arrives when the channel buffer is full are lost.

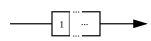
Constraint Automata The deterministic parameterized constraint automaton for the asynchronous bounded lossy FIFO channel with capacity of k is

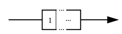


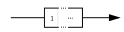
4.9 Unbounded FIFO Buffer — UnboundedFIFO

Description An asynchronous, buffered (unbounded), and ordered channel with a source and a sink end [1].

Circuit The `UnboundedFIFO` channel is graphically represented by

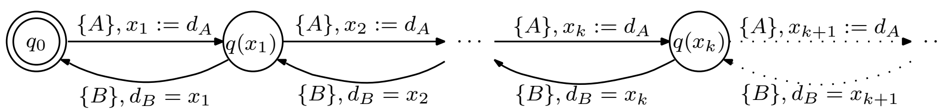


ABT The `UnboundedFIFO` channel, , is defined for all timed data streams $\langle \alpha, a \rangle$ and $\langle \beta, b \rangle$ by

$\langle \alpha, a \rangle$  $\langle \beta, b \rangle \equiv \alpha = \beta \wedge a < b$

What goes in, comes out, but with an arbitrary (non-zero) delay.

Constraint Automata The deterministic parameterized constraint automaton for the asynchronous unbounded FIFO channel is

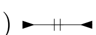


4.10 Asynchronous drain – AsyncDrain

Description An asynchronous, unbuffered, and lossy channel with two source ends [1].

Circuit The `AsyncDrain` channel is graphically represented by

ABT The `AsyncDrain` channel, , is defined for all timed data streams $\langle \alpha, a \rangle$ and $\langle \beta, b \rangle$ by

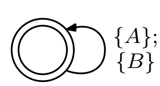
$(\langle \alpha, a \rangle, \langle \beta, b \rangle;)$  $\equiv a \bowtie b$

where

$$a \bowtie b \equiv a(0) \neq b(0) \wedge \begin{cases} a' \bowtie b & \text{if } a(0) < b(0) \\ a \bowtie b' & \text{if } b(0) < a(0) \end{cases}$$

The channel guarantees that two operations on its two ends never succeed simultaneously. The channel is *fair* by alternating between its two ends and giving each a chance to dispose of a data item. All data items written to this channel are lost.

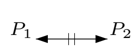
Constraint Automata The corresponding deterministic constraint automaton for the asynchronous drain channel is



4.11 Asynchronous Spout – AsyncSpout

Description An asynchronous, unbuffered channel with two sink ends [1].

Circuit The `AsyncSpout` channel with patterns P_1 and P_2 is graphically represented by

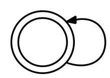


ABT The `AsyncSpout` channel, $^{P_1}\!\leftarrow\!\Vert\!\rightarrow^{P_2}$, is defined, for all timed data streams $\langle\alpha, a\rangle$ and $\langle\beta, b\rangle$, by

$$^{P_1}\!\leftarrow\!\Vert\!\rightarrow^{P_2}(;\langle\alpha, a\rangle, \langle\beta, b\rangle) \equiv \begin{cases} a(0) \neq b(0) \ \wedge \ \alpha(0) \ni P_1 \ \wedge \ \beta(0) \ni P_2 \\ ^{P_1}\!\leftarrow\!\Vert\!\rightarrow^{P_2}(;\langle\alpha', a'\rangle, \langle\beta', b'\rangle) \end{cases}$$

It is an unbounded source of data items that match with its specified patterns, P_1, P_2 and never can be taken from its opposite ends simultaneously. The channel is *fair* by alternating between its two ends and giving each a chance to obtain a data item from the channel. The data items are produced in a non-deterministic order.

Constraint Automata The deterministic constraint automaton for the asynchronous spout channel is

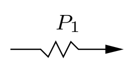


$\{A\}, d_A \ni P_1;$
$\{B\}, d_B \ni P_2$

4.12 Filter

Description A synchronous, unbuffered, ordered, lossy channel which has both a source and a sink end [1].

Circuit The `Filter` channel with pattern P_1 is graphically represented by

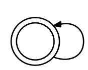


ABT The $\texttt{Filter}_{P_1}$ channel, $\overset{P_1}{\rightsquigarrow}$, is defined, for all timed data streams $\langle\alpha, a\rangle$ and $\langle\beta, b\rangle$, by

$$\langle\alpha, a\rangle \overset{P_1}{\rightsquigarrow} \langle\beta, b\rangle \equiv \begin{cases} a(0) = b(0) \ \wedge \ \alpha(0) = \beta(0) \ \wedge \\ \langle\alpha', a'\rangle \overset{P_1}{\rightsquigarrow} \langle\beta', b'\rangle & \text{if } \alpha(0) \ni P_1 \\ \langle\alpha', a'\rangle \overset{P_1}{\rightsquigarrow} \langle\beta, b\rangle & \textit{otherwise} \end{cases}$$

It is a special lossy synchronous channel. It transfers only those data items that match with its specified pattern, P_1 and loses the rest.

Constraint Automata The deterministic constraint automaton for the filter channel is



$\{A, B\}, d_A \ni P_1 \wedge d_A = d_B$

4.13 Ordered

Description An asynchronous, buffered, and ordered channel with a source and a sink end [3].

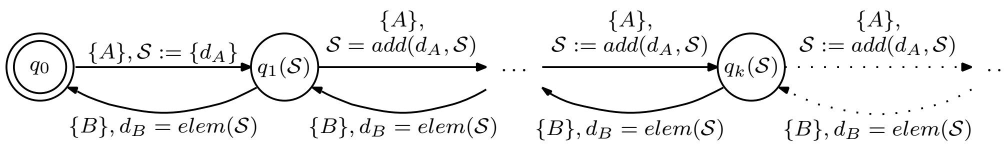
Circuit The `Ordered` channel is graphically represented by

ABT ABT formalism abstracts away from the stream of input and output requests. The information about the data pattern that an output value need to satisfy is therefore not captured. In the `Ordered` channel the choice of the output data element from the buffer is dependent on the data pattern which is specified together with the output request. Without this information the Ordered channel's ABT specification does not describes in which manner the `Ordered` channel is ordered.

The `Ordered` channel, $\xrightarrow{\texttt{Ord}}$, is defined for all timed data streams $\langle \alpha, a \rangle$, $\langle \beta, b \rangle$ and $i, j, m, n, o, p \in \mathbb{R}_0^+$ by

$$\langle \alpha, a \rangle \xrightarrow{\texttt{Ord}} \langle \beta, b \rangle \equiv \begin{array}{l} \forall i, \exists j : \alpha(i) = \beta(j) \ \wedge \ a(i) < b(j) \ \wedge \\ (\forall m, o, \exists n, p : (\alpha(m) = \beta(n) \ \wedge \ \alpha(o) = \beta(p) \ \wedge \ m = o) \Rightarrow n = p) \end{array}$$

Constraint Automata The deterministic parametrized constraint automaton for the ordered channel is



Variations Ordered_k

4.14 Ordered_k

Description An asynchronous, buffered, and ordered channel with a source and a sink end [3].

Circuit The $\texttt{Ordered}_\texttt{k}$ channel is graphically represented by

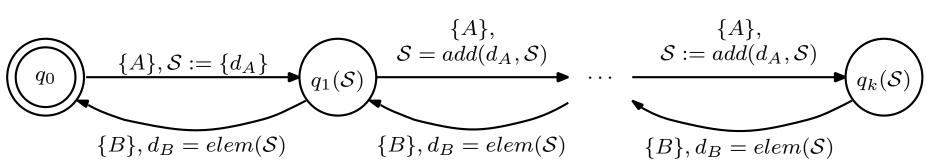
$$\xrightarrow{\texttt{Ord}_\texttt{k}}$$

ABT The $\texttt{Ordered}_\texttt{k}$ channel, $\xrightarrow{\texttt{Ord}_\texttt{k}}$, is defined for all timed data streams $\langle \alpha, a \rangle$, $\langle \beta, b \rangle$ and $i, j, m, n, o, p \in \mathbb{R}_0^+$ by

$$\langle \alpha, a \rangle \xrightarrow{\texttt{Ord}_\texttt{k}} \langle \beta, b \rangle \equiv \begin{array}{l} \forall i, \exists j : \alpha(i) = \beta(j) \ \wedge \ a(i) < b(j) < a(k) \ \wedge \\ (\forall m, o, \exists n, p : (\alpha(m) = \beta(n) \ \wedge \ \alpha(o) = \beta(p) \ \wedge \ m = o) \Rightarrow n = p) \end{array}$$

(*Comment: Explanation TODO*)

Constraint Automata The deterministic parametrized constraint automaton for the ordered bounded buffer channel with capacity of k is



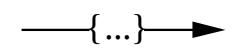
(*Comment: TODO*)

Variations Ordered

4.15 Set

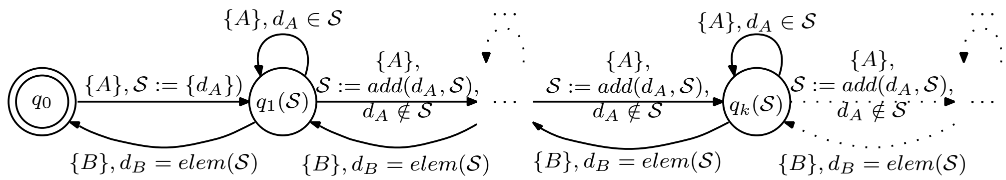
Description An asynchronous, buffered (unbounded), and unordered channel with a source and a sink end [3].

Circuit The `Set` channel is graphically represented by



ABT (*Comment: Explanation TODO*)

Constraint Automata The deterministic parametrized constraint automaton for the set channel is



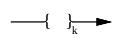
(*Comment: TODO*)

Variations Set$_k$

4.16 Set$_k$

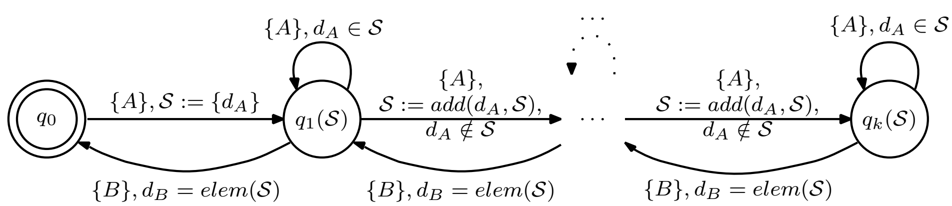
Description An asynchronous, buffered, and unordered channel with a source and a sink end [3].

Circuit The `Set`$_{\texttt{k}}$ channel is graphically represented by



ABT (*Comment: Explanation TODO*)

Constraint Automata The deterministic parametrized constraint automaton for the set channel with limited capacity k

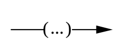


Variations Set

4.17 Bag

Description An asynchronous, buffered (unbounded), and unordered channel with a source and a sink end [3].

Circuit The `Bag` channel is graphically represented by



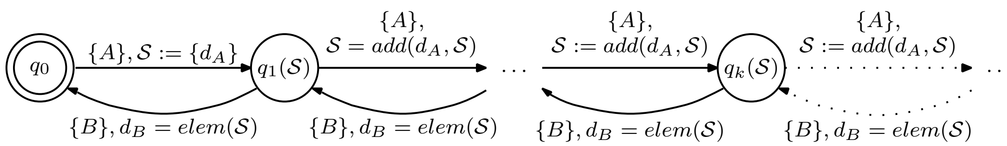
ABT The `Bag` channel, , is defined for all timed data streams $\langle \alpha, a \rangle$, $\langle \beta, b \rangle$ and $i, j, m, n, o, p \in \mathbb{R}_0^+$ by

$\langle \alpha, a \rangle$ $\langle \beta, b \rangle \equiv$

$$\forall i, \exists j : \alpha(i) = \beta(j) \ \wedge \ a(i) < b(j) \ \wedge$$
$$(\forall m, o, \exists n, p : (\alpha(m) = \beta(n) \ \wedge \ \alpha(o) = \beta(p) \ \wedge \ m = o) \Rightarrow n = p)$$

(*Comment: Explanation TODO*)

Constraint Automata The deterministic parametrized constraint automaton for the bag channel is



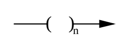
(*Comment: TODO*)

Variations Bag$_n$

4.18 Bag$_n$

Description An asynchronous, buffered (with capacity n), and unordered channel with a source and a sink end [3].

Circuit The `Bag_n` channel is graphically represented by



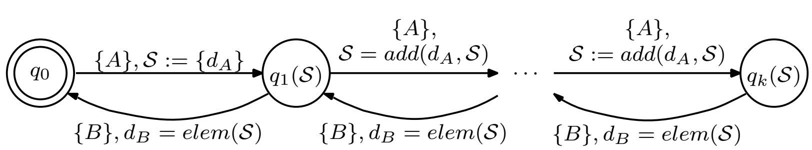
ABT The `Bag_k` channel, , is defined for all timed data streams $\langle \alpha, a \rangle$, $\langle \beta, b \rangle$ and $i, j, m, n, o, p \in \mathbb{R}_0^+$ by

$\langle \alpha, a \rangle$ $\langle \beta, b \rangle \equiv$

$$\forall i, \exists j : \alpha(i) = \beta(j) \ \wedge \ a(i) < b(j) < a(k) \ \wedge$$
$$(\forall m, o, \exists n, p : (\alpha(m) = \beta(n) \ \wedge \ \alpha(o) = \beta(p) \ \wedge \ m = o) \Rightarrow n = p)$$

(*Comment: Explanation TODO*)

Constraint Automata The deterministic parametrized constraint automaton for the bag channel with limited capacity is



(*Comment: TODO*)

Variations Bag

4.19 DelaySet

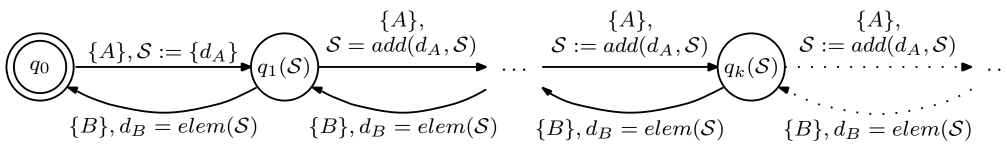
Description An asynchronous, buffered, unbounded and unordered channel with a source and a sink end [3].

Circuit The `DelaySet` channel is graphically represented by

ABT The `DelaySet` channel, $\xrightarrow{\texttt{DSet}}$, is defined for all timed data streams $\langle \alpha, a \rangle$, $\langle \beta, b \rangle$ and $i, j, m, n, o, p \in \mathbb{R}_0^+$ by

$$\langle \alpha, a \rangle \xrightarrow{\texttt{DSet}} \langle \beta, b \rangle \equiv \begin{array}{l} \forall i, \exists j : \alpha(i) = \beta(j) \ \wedge \ a(i) < b(j) \ \wedge \\ (\forall m, o, \exists n, p : (\alpha(m) = \beta(n) \ \wedge \ \alpha(o) = \beta(p) \ \wedge \ m = o) \Rightarrow n = p) \end{array}$$

Constraint Automata The deterministic parametrized constraint automaton for the DelaySet channel is



(*Comment: TODO*)

Variations DelaySet$_k$

4.20 DelaySet$_k$

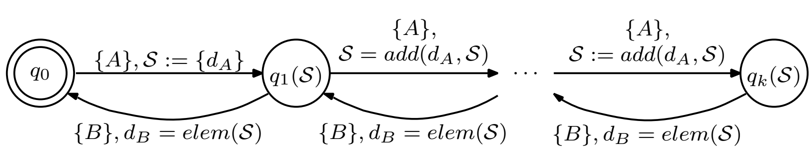
Description An asynchronous, buffered, bounded and unordered channel with a source and a sink end [3].

Circuit The `DelaySet`$_\texttt{k}$ channel is graphically represented by

ABT The `DelaySet`$_\texttt{k}$ channel, $\xrightarrow{\texttt{DSet}_k}$, is defined for all timed data streams $\langle \alpha, a \rangle$, $\langle \beta, b \rangle$ and $i, j, m, n, o, p \in \mathbb{R}_0^+$ by

$$\langle \alpha, a \rangle \xrightarrow{\texttt{DSet}_k} \langle \beta, b \rangle \equiv \begin{array}{l} \forall i, \exists j : \alpha(i) = \beta(j) \ \wedge \ a(i) < b(j) < a(k) \ \wedge \\ (\forall m, o, \exists n, p : (\alpha(m) = \beta(n) \ \wedge \ \alpha(o) = \beta(p) \ \wedge \ m = o) \Rightarrow n = p) \end{array}$$

Constraint Automata The deterministic parametrized constraint automaton for the delay set channel with limited capacity k is



(*Comment: TODO*)

Variations `DelaySet`

5 Connectors

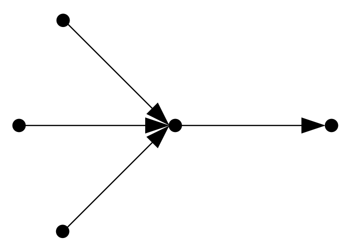
Connectors are generally considered to be non-primitive, whereas channels are often primitive, and usually have an "arity" which is not equal to 2. Channels always have arity 2. In general, however, channels are just special cases of connectors.

5.1 Merger

Description This connector takes an arbitrary number of source nodes. Data input to these nodes is merged, non-deterministically, and available at a sink node. Data can only be transferred if a take is being requested at the sink node simultanously with a write at one of the source nodes. Ties are broken non-deterministically [6, 1, 5, 4].

The functionality of a merger is derived directly from the functionality of Reo nodes. We include it as a connector because of its general usefulness.

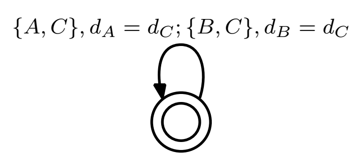
Circuit The Reo circuit for the `Merger` connector with three sources is



ABT The ABT for a merge of two sources and one sink is a ternary relation M defined for timed data streams $\langle \alpha, a\rangle$, $\langle \beta, b\rangle$, $\langle \gamma, c\rangle$ by

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

Constraint Automata The constraint automaton for a Merger with two inputs (A and B) and one output (C) is



Context Can be used in any context, generally in the guise of a mixed node.

Variations This connector may have any number of input nodes.

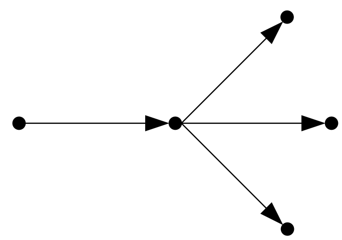
Also Known As A mixed node.

5.2 Replicator

Description This connector has a single source node and multiple sink nodes. Data input is replicated to all of the sink nodes. Data flows only when all sink nodes are ready to take and the source node is ready to write.

The functionality of a replicator is derived directly from the functionality of Reo nodes. We include it as a connector because of its general usefulness [6, 8, 1, 5, 4].

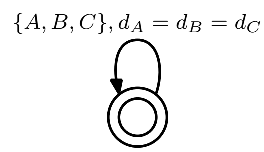
Circuit The Reo circuit for the `Replicator` connector with three outputs is



ABT The ABT for the `Replicator` with an input end two output ends is a ternary relation R defined for timed data streams $\langle\alpha, a\rangle$, $\langle\beta, b\rangle$, $\langle\gamma, c\rangle$ by

$$R(\langle\alpha, a\rangle; \langle\beta, b\rangle, \langle\gamma, c\rangle) \quad \equiv \quad \alpha = \beta = \gamma \ \wedge a = b = c$$

Constraint Automata The constraint automaton for a `Replicator` with one input (A) and two outputs (B and C) is



Context Can be used in any context, generally in the guise of a mixed node.

Variations This connector may have any number of output nodes.

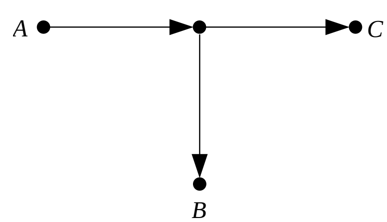
Composite Primitives This circuit is constructed using a node and a number of synchronous channels.

Also Known As A mixed node.

5.3 Take-Cue Regulator

Description In this circuit, the data from one node (A) to another (B) is regulated by the taking of data at a third node (C). That is, data can flow from A to B only if both A and B are ready and, further, that C is also ready. This mean that the usual connection between A and B is regulated by the behaviour at C. Because this is a take-cue regulator, C regulates using take and receives the data written at A [6, 1, 3].

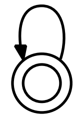
Circuit The Reo circuit for a Take-Cue Regulator is



ABT The ABT for a take-cue regulator of an input end with an output end and an output end used to regulate the flow is a ternary relation TQR defined for timed data streams $\langle\alpha, a\rangle$, $\langle\beta, b\rangle$, $\langle\gamma, c\rangle$ by

$$TQR(\langle\alpha, a\rangle; \langle\beta, b\rangle, \langle\gamma, c\rangle) \quad \equiv \quad \alpha = \beta = \gamma \ \wedge a = b = c$$

Constraint Automata A deterministic constraint automata for the Take-cue regulator is



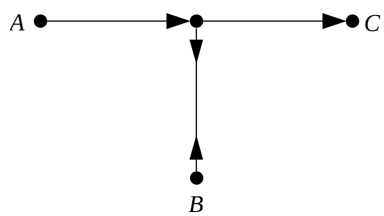
Variations A write-cue regulator.

Also Known As Interestingly, this is exactly the same circuit as a replicator. The difference is how it is perceived.

5.4 Write-Cue Regulator

Description In this circuit, the data from one node (A) to another (B) is regulated by the writing of data to a third node (C). That is data can flow from A to B only if both A and B are ready and, further, that C is also ready. This mean that the usual connection between A and B is regulated by the behaviour at C. Because this is a write-cue regulator, C regulates using write, though the data it writes is lost [6, 1, 5, 4].

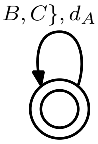
Circuit The Reo circuit for a Write-Cue Regulator is



ABT The ABT for a take-cue regulator of an input end with an output end and an output end used to regulate the flow is a ternary relation WQR defined for timed data streams $\langle \alpha, a \rangle$, $\langle \beta, b \rangle$, $\langle \gamma, c \rangle$ by

$$WQR(\langle \alpha, a \rangle, \langle \gamma, c; \langle \beta, b \rangle \rangle) \quad \equiv \quad \alpha = \gamma \ \wedge a = b = c$$

Constraint Automata A deterministic constraint automata for the Write-cue regulator is

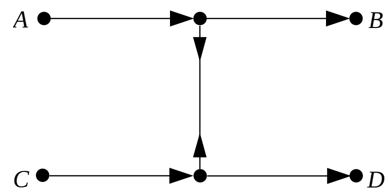


Variations Take-cue regulator.

5.5 Barrier Synchronizer

Description A barrier synchronizer connector enables data items to pass from A to B and from C to D, but only at the same time, that is, data can only flow when there is either a write or take pending on all of A, B, C, and D [6, 1, 5, 4].

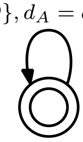
Circuit The Reo circuit for the Barrier Synchronizer is



ABT The ABT for a barrier synchronizer is a quaternary relation BS defined for timed data streams $\langle \alpha, a\rangle$, $\langle \beta, b\rangle$, $\langle \gamma, c\rangle$ and $\langle \delta, d\rangle$ by

$$BS(\langle \alpha, a\rangle, \langle \gamma, c\rangle; \langle \beta, b\rangle, \langle \delta, d\rangle) \quad \equiv \quad \alpha = \beta \ \wedge \ \gamma = \delta \ \wedge a = b = c = d$$

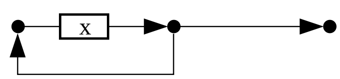
Constraint Automata A deterministic constraint automata for the Barrier Synchronizer is



5.6 Feedback Loop

Description Using feedback, it is possible to have a circuit which produces a continuous, constant stream of data on demand [6].

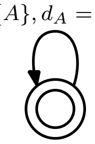
Circuit A simple feedback loop is given by the circuit:



ABT The ABT for a Feedback Loop is an unary relation FL_X defined for the timed data stream $\langle \alpha, a\rangle$,

$$FL_X(\langle \alpha, a\rangle) \equiv \alpha(0) = X \ \wedge \ FL_X(\langle \alpha', a'\rangle)$$

Constraint Automata A deterministic constraint automata for the Feedback Loop is

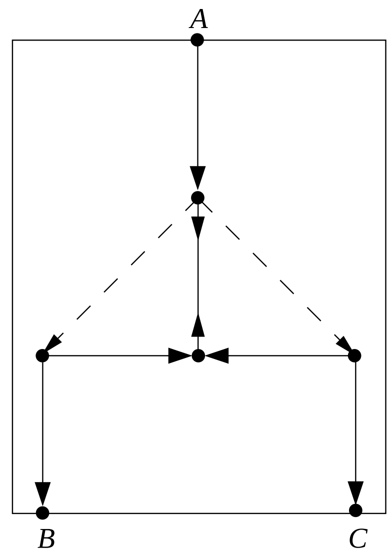


5.7 Asynchronous drain using the merge connector and a synchronous drain [6]

5.8 Exclusive Router

Description Each data item entering via node A will be synchronously passed to either node B or node C, but not both, depending upon which of B and C first makes a request for data. Ties are broken non-deterministically [8, 2, 5, 4, 7].

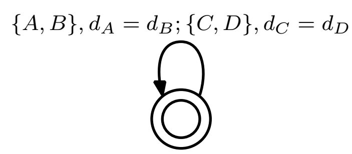
Circuit The Reo circuit for the Exclusive Router is



ABT The ABT for an exclusive router is a ternary relation *ExR* defined for timed data streams $\langle\alpha, a\rangle$, $\langle\beta, b\rangle$, $\langle\gamma, c\rangle$ by

$$
\begin{aligned}
&\mathit{ExR}(\langle\alpha, a\rangle; \langle\beta, b\rangle, \langle\gamma, c\rangle) \equiv \\
&\quad b(0) \neq c(0) \;\wedge \\
&\quad \begin{cases} \alpha(0) = \beta(0) \;\wedge\; a(0) = b(0) \;\wedge\; \mathit{ExR}(\langle\alpha', a'\rangle, \langle\beta', b'\rangle, \langle\gamma, c\rangle) & \text{if } b(0) < c(0) \\ \alpha(0) = \gamma(0) \;\wedge\; a(0) = c(0) \;\wedge\; \mathit{ExR}(\langle\alpha', a'\rangle, \langle\beta, b\rangle, \langle\gamma', c'\rangle) & \text{if } c(0) < b(0) \end{cases}
\end{aligned}
$$

Constraint Automata A deterministic constraint automaton for the Exclusive Router is



Also Known As ExRouter [8].

Notice the similarity of the semantics with those of Merger. Hardly surprising, given that they have the same constraint automaton, just a different direction of data flow.

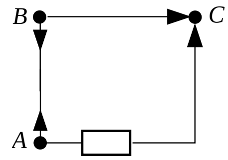
5.9 Replicator Connector [8]

As opposed to the previously defined replicator connector, this connector duplicates the elements sent along a channel.

5.10 Ordering

Description The behaviour of this connector imposes an order on the flow of data items written to A and B and passed to C. The first item comes from A, then from B, then back to A. Data can only flow if data is present at both A and B simultaneously [1], [5], [4], [3].

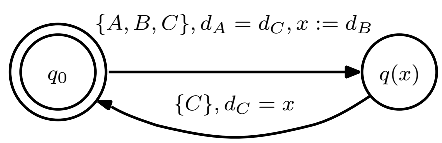
Circuit The Reo circuit for the Ordering is



ABT The ABT for the ordering connector is a ternary relation OC defined for timed data streams $\langle \alpha, a \rangle$, $\langle \beta, b \rangle$, $\langle \gamma, c \rangle$ by

$$\begin{array}{l} OC(\langle \alpha, a \rangle, \langle \beta, b \rangle; \langle \gamma, c \rangle) \equiv \\ \quad \alpha(0) = \gamma(0) \ \wedge \ \beta(0) = \gamma(1) \ \wedge \ a(0) = b(0) = c(0) \ \wedge \ a(1) = b(1) > c(1) \ \wedge \\ \quad OC(\langle \alpha', a' \rangle, \langle \beta', b' \rangle; \langle \gamma'', c'' \rangle) \end{array}$$

Constraint Automata The deterministic parameterized constraint automaton for the Ordering connector is



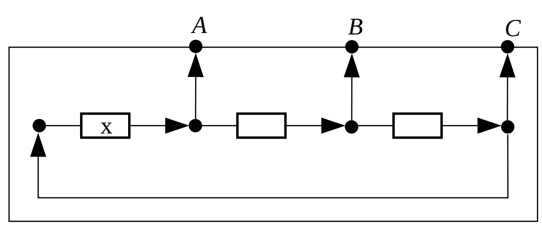
Also Known As Interleaving connector [8]. Note that using a sequencer is less constraint manner for achieving the same effect.

Related Connectors If this simultaneity between the sink nodes is too strong, use a Sequencer.

5.11 Sequencer

Description A sequencer consists of some number of nodes (3 in our example), say A, B, C. The sequencer begins by outputing a token to one of the nodes, say A. This enables the circuit connected to A to take. The sequencer then moves into a state in which B can take. After B is taken, then the sequencer moves into a state where C can take. After C is taken, the sequencer returns to the initial state and the cycle can repeat itself. This circuit thus imposes an order on the flow of data at the nodes A, B, and C [1, 5, 6, 4]

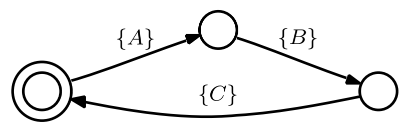
Circuit The Reo circuit for the Sequencer is



ABT The ABT for an sequencer is a ternary relation SQ defined for timed data streams $\langle \alpha, a \rangle$, $\langle \beta, b \rangle$, $\langle \gamma, c \rangle$ by

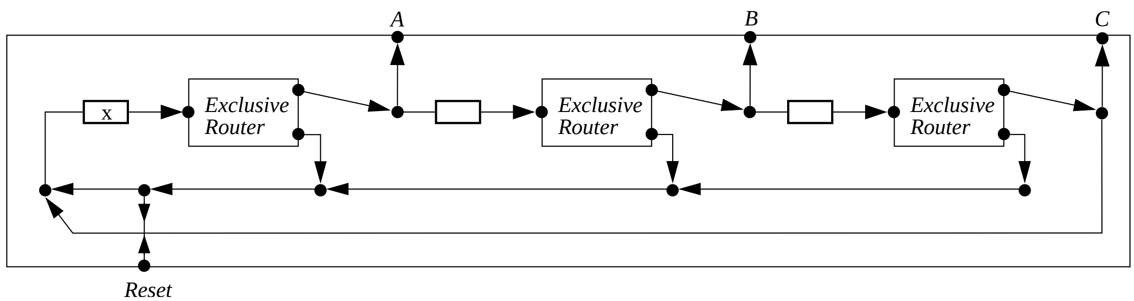
$$SQ(;\langle \alpha, a \rangle, \langle \beta, b \rangle, \langle \gamma, c \rangle) \quad \equiv \quad a < b < c < a'$$

Constraint Automata A deterministic constraint automata for the Sequencer is

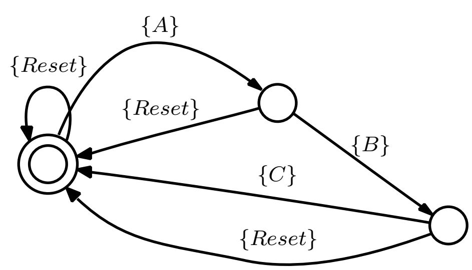


Variations This circuit is a more asychrounous version of Ordering.

Sequencer with Reset [4]. This circuit is similar in behaviour to a Sequencer. It consists of an additional channel which, when data is written to it, returns the sequencer to its initial state.



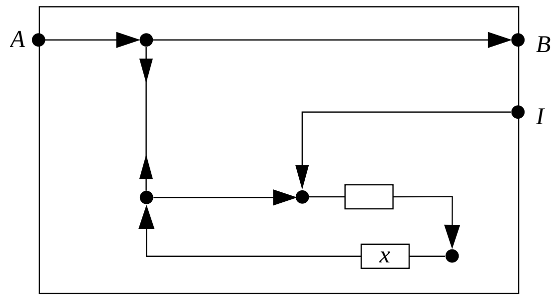
A deterministic constraint automaton for the Sequencer with Reset connectors is:



5.12 Inhibitor

Description Data written at A flows freely to B until some data value is written at I, after which data flow stops for good [1]. *Interestingly, this circuit deadlocks by design.*

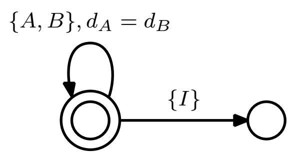
Circuit The Reo circuit for the Inhibitor is



ABT The ABT for an inhibitor is a ternary relation *Ih* defined for timed data streams $\langle \alpha, a \rangle$, $\langle \beta, b \rangle$, $\langle \iota, i \rangle$ by

$$Ih(\langle \alpha, a \rangle, \langle \iota, i \rangle; \langle \beta, b \rangle) \equiv$$
$$\begin{cases} a(0) = b(0) \ \wedge \ \alpha(0) = \beta(0) \ \wedge \ Ih(\langle \alpha', a' \rangle, \langle \iota, i \rangle; \langle \beta', b' \rangle) & \text{if } a(0) < i(0) \\ \alpha = a = \beta = b = \iota' = i' = \langle \rangle & \text{if } i(0) < a(0) \end{cases}$$

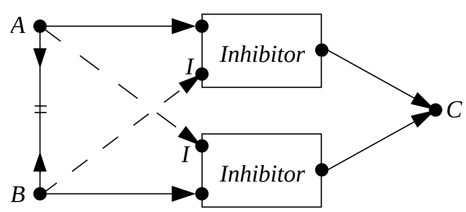
Constraint Automata A deterministic constraint automata for the Sequencer is



5.13 Or-Selector

Description Data is non-deterministically chosen from one of its two inputs and sent synchronously to the output C. Once either A or B is chosen, no data can flow through to C from the other. Data from the end not chosen is simply lost [1].

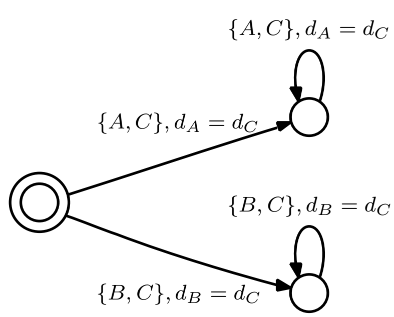
Circuit The Reo circuit for the Or-Selector is



ABT The ABT for an inhibitor is a ternary relation *OS* defined for timed data streams $\langle \alpha, a \rangle$, $\langle \beta, b \rangle$, $\langle \gamma, c \rangle$ by

$$OS(\langle \alpha, a \rangle, \langle \beta, b \rangle; \langle \gamma, c \rangle) \quad \equiv \quad (a = c \ \wedge \ \alpha = \gamma) \ \vee \ (b = c \ \wedge \ \beta = \gamma)$$

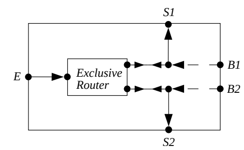
Constraint Automata A deterministic constraint automata for the Or-Selector is



5.14 Selector

Description Availability of a value at its input node E "enables" this connector to select a value available on one of the input nodes $B1$ or $B2$ for transfer respectively through $S1$ or $S2$ [4]. The prerequisite for transfer is that both $B1$ and $S1$ (or $B2$ and $S2$) are ready to write/take. Note that if either $S1$ or $S2$ are not ready when their respective B is, the data is lost. Similarly, if all nodes are ready, then the tie is broken nondeterministically, and data in the losing B node is lost.

Circuit The Reo circuit for the Selector is

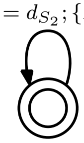


ABT The ABT for the selector connector is a quinary relation SC defined for timed data streams $\langle \epsilon, e\rangle$, $\langle \beta_1, b_1\rangle$, $\langle \beta_2, b_2\rangle$, $\langle \sigma, s_1\rangle$, $\langle \sigma, s_2\rangle$ by

$$SC(\langle \epsilon, e\rangle, \langle \beta_1, b_1\rangle, \langle \beta_2, b_2\rangle; \langle \sigma_1, s_1\rangle, \langle \sigma_2, s_2\rangle) \equiv$$
$$\begin{cases} SC(\langle \epsilon, e\rangle, \langle \beta_1', b_1'\rangle, \langle \beta_2, b_2\rangle; \langle \sigma_1, s_1\rangle, \langle \sigma_2, s_2\rangle) & \text{if } b_1(0) < e(0) \\ SC(\langle \epsilon, e\rangle, \langle \beta_1, b_1\rangle, \langle \beta_2', b_2'\rangle; \langle \sigma_1, s_1\rangle, \langle \sigma_2, s_2\rangle) & \text{if } b_2(0) < e(0) \\ SC(\langle \epsilon, e\rangle, \langle \beta_1', b_1'\rangle, \langle \beta_2', b_2'\rangle; \langle \sigma_1, s_1\rangle, \langle \sigma_2, s_2\rangle) & \text{if } b_1(0) = b_2(0) < e(0) \\ \beta_1(0) = \sigma_1(0) \ \wedge \ SC(\langle \epsilon', e'\rangle, \langle \beta_1', b_1'\rangle, \langle \beta_2, b_2\rangle; \langle \sigma_1', s_1'\rangle, \langle \sigma_2, s_2\rangle) & \text{if } b_1(0) = s_1(0) = e(0) \\ \beta_2(0) = \sigma_2(0) \ \wedge \ SC(\langle \epsilon', e'\rangle, \langle \beta_1, b_1\rangle, \langle \beta_2', b_2'\rangle; \langle \sigma_1, s_1\rangle, \langle \sigma_2', s_2'\rangle) & \text{if } b_2(0) = s_2(0) = e(0) \\ \beta_1(0) = \sigma_1(0) \ \wedge \ SC(\langle \epsilon', e'\rangle, \langle \beta_1', b_1'\rangle, \langle \beta_2', b_2'\rangle; \langle \sigma_1', s_1'\rangle, \langle \sigma_2, s_2\rangle) & \text{if } b_1(0) = s_1(0) = s_2(0) = e(0) \\ \beta_1(0) = \sigma_1(0) \ \wedge \ SC(\langle \epsilon', e'\rangle, \langle \beta_1', b_1'\rangle, \langle \beta_2', b_2'\rangle; \langle \sigma_1, s_1\rangle, \langle \sigma_2', s_2'\rangle) & \text{if } b_1(0) = s_1(0) = s_2(0) = e(0) \end{cases}$$

Constraint Automata The deterministic contraint automaton for the selectonr connector is

$$\{B_1\}; \{B_2\}; \{E, B_1, S_1\}, d_{B_1} = d_{S_1}; \{E, B_2, S_2\}, d_{B_2} = d_{S_2}; \{E, B_1, S_1, B_2\}, d_{B_1} = d_{S_1}; \{E, B_2, S_2, B_1\}, d_{B_2} = d_{S_2};$$

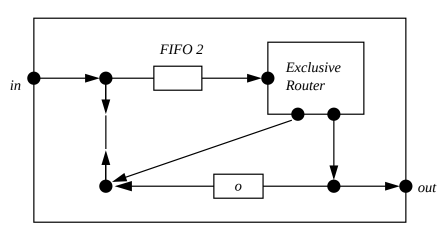


Variations Surely there's the variation which doesn't lose data.

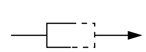
5.15 Shift-Lossy FIFO1 channel

Description This is a connector which is often used as a simple channel in the construction of other connectors. It behaves similarly to a FIFO1 channel, except that it loses its current value if its buffer is full to accept a new input value instead [2], [5], [4], [7].

Circuit The circuit for the overflow-lossy FIFO1 is



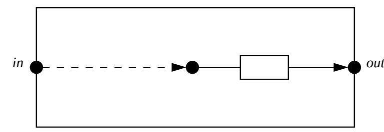
It is represented by the icon



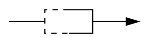
5.16 Overflow-Lossy FIFO1

Description This channel is the counterpart of the shift-lossy $FIFO_1$ channel [4]. The data-loss policy favours retaining older buffer values over newer arrivals.

Circuit The circuit for the overflow-lossy FIFO1 is



It is represented by the icon



5.17 Initializer [2]

5.18 Terminator [2]

5.19 Flow Regulator [5], [7]

5.20 Sum

Description This connector has two input nodes and one output node. When two integer inputs are available, their sum is subsequently made available on the output node [4].

Circuit There is no particular circuit for Sum. Arithmetic operations such as this can be supplied to Reo as "components".

ABT The ABT for sum is a ternary relation *Sum* defined for timed data streams $\langle\alpha, a\rangle$, $\langle\beta, b\rangle$, $\langle\gamma, c\rangle$ by

$$\begin{aligned} \mathit{Sum}(\langle\alpha, a\rangle, \langle\beta, b\rangle; \langle\gamma, c\rangle) \quad \equiv \quad & \gamma(0) = \alpha(0) + \beta(0) \\ & \wedge \ \max(a(0), b(0)) < c(0) < \min(a(1), b(1)) \\ & \wedge \ \mathit{Sum}(\langle\alpha', a'\rangle, \langle\beta', b'\rangle; \langle\gamma', c'\rangle) \end{aligned}$$

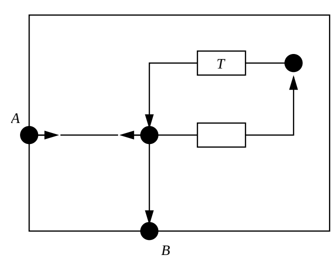
Constraint Automata (*Comment: TODO: Requires parameterized automata.*)

Variations Many variations are possible, including multiple input arguments or using a different arithmetic operation or a relational operation. Variations in the amount of synchronicity present are also possible.

5.21 Constant Replacer

Description This channel takes data from its input A and simultaneously replaces it with some constant T on output B [4]

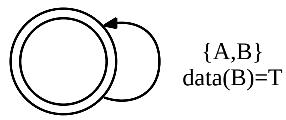
Circuit The Reo circuit for the Constant Replacer is



ABT The ABT for Constant Replacer is a binary relation $CR(T)$, parameterized by the constant T, defined for timed data streams $\langle\alpha, a\rangle$, $\langle\beta, b\rangle$ by

$$CR(T)(\langle\alpha, a\rangle; \langle\beta, b\rangle) \quad \equiv \quad a = b \ \wedge \ \beta(0) = T \ \wedge \ \beta' = \beta$$

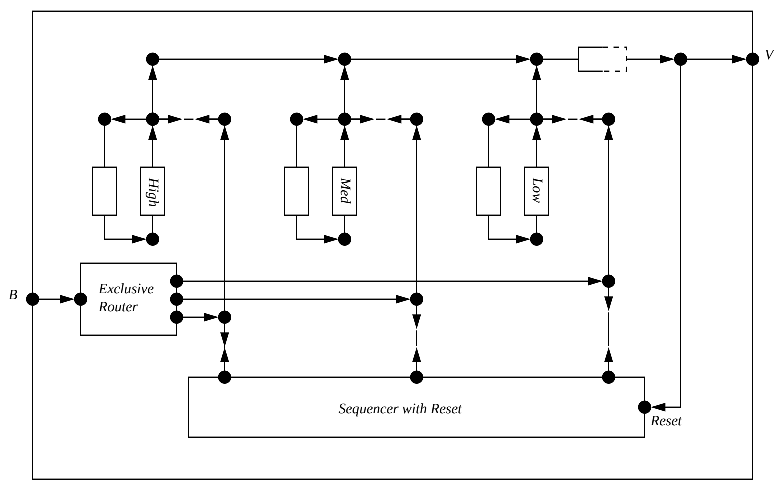
Constraint Automata The deterministic constraint automata for a constant replacer is:



5.22 Cycler

Description A cycler connector behaves as follows. The first input value through its node B places the value *High* in its shift-lossy FIFO_1 channel, ready for output through the V node. Successive input values through B "cycle" through the remaining values in the sequence *Med*, *Low*, restarting the cycle again from *High*, and make each value available, in turn, for output through V, by overriding the previous contents of the shift-lossy FIFO_1 channel [4]. Whenever a value is consumed through V, the sequencer resets the connector to restart the cycle from its leftmost value, *High*.

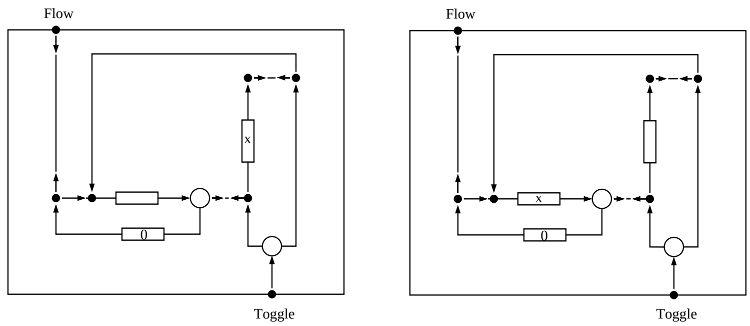
Circuit The Reo circuit for the Cycler is



5.23 Valves

Description This connector behaves as a valve. It has three nodes A, B, and I. When in the open state, data can flow synchronously from A to B. When in the closed state, no data can flow between A and B. Data on the I node acts as a toggle, changing the state between being open and closed. The circuit comes in two forms: initially open and initially closed [4].

Circuit The Reo circuits for an initially-open valve and an intially-closed value are:



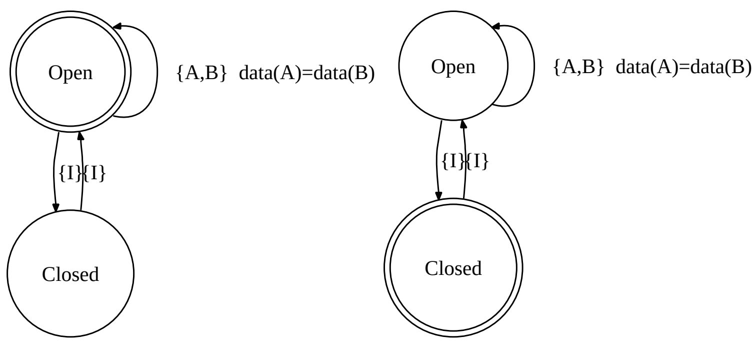
Note that the circle in these figures corresponds to an exclusive router.

ABT The ABT for initially-open and initially-closed valves are binary relations V_o and V_c mutually defined for timed data streams $\langle\alpha, a\rangle$ (flow), $\langle\beta, b\rangle$ (toggle) by

$$V_o(\langle\alpha, a\rangle, \langle\tau, t\rangle; \langle\beta, b\rangle) \equiv \begin{cases} a(0) = b(0) \wedge \alpha(0) = \beta(0) \wedge & \\ \quad V_o(\langle\alpha', a'\rangle, \langle\tau, t\rangle; \langle\beta', b'\rangle) & \text{if } a(0) < t(0) \\ V_c(\langle\alpha, a\rangle, \langle\tau', t'\rangle; \langle\beta, b\rangle) & \text{if } t(0) < a(0) \end{cases}$$

$$V_c(\langle\alpha, a\rangle, \langle\tau, t\rangle; \langle\beta, b\rangle) \equiv \begin{cases} V_o(\langle\alpha, a\rangle, \langle\tau, t\rangle; \langle\beta, b\rangle) & \text{if } a(0) < t(0) \\ V_c(\langle\alpha, a\rangle, \langle\tau', t'\rangle; \langle\beta, b\rangle) & \text{if } t(0) < a(0) \end{cases}$$

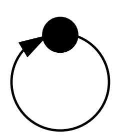
Constraint Automata The constraint automata for an initially-open valve and an intially-closed value are:



5.24 Short-circuit

Description This is not so much a circuit as a pitfall to avoid [3].

Circuit The circuit consists of a loop containing one or more synchronous chanels



ABT The behaviour of this circuit is the empty ABT (of the appropriate dimension).

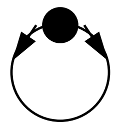
Constraint Automata The behaviour of this circuit is the empty constraint automaton (one node, no edges) over the appropriate collection of nodes.

Variations Any number of synchronous channels can be involved in the loop, giving a Transitive short-circuit.

5.25 Drain

Description This connector has one input/sink node. All data sent to this node can always immediately be accepted [3].

Circuit The Reo circuit for a drain is



ABT The ABT for this circuit is the complete set of timed data streams.

Constraint Automata This is the **1** constraint automata — it contains one node and one edge with the node name (and no data constraint).

6 Examples

References

[1] F. Arbab. A channel-based coordination model for component composition, 2001.

[2] Farbab Arbab, Christel Baier, Frank de Boer, Jan Rutten, and Sirjani Marjan. On the synthesis of reo component connectors. Submitted.

[3] Farhad Arbab. Rewrite rules for reo. Notes.

[4] Farhad Arbab. Abstract behavior types: a foundation model for components and their composition. SEN-R0305 ISSN 1386-369X, CWI, 2003.

[5] Farhad Arbab. *Interactive Computation: The New Paradigm*, chapter Coordination of Interacting Concurrent Computations. To appear, 2004.

[6] Farhad Arbab and Jan J.M.M. Rutten. A coinductive calculus of component connectors. In *Proceedings of 16th International Workshop on Algebraic Development Techniques (WADT'02)*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, 2002.

[7] Christel Baier, Marjan Sirjani, Farhad Arbab, and Jan Rutten. Modeling component connectors in Reo by constraint automata. *Submitted*, 2004.

[8] MohammadReza Mousavi, Marjan Sirjani, and Farhad Arbab. Operational semantics and model checking of component connectors in reo.

Index