# Chapter 3

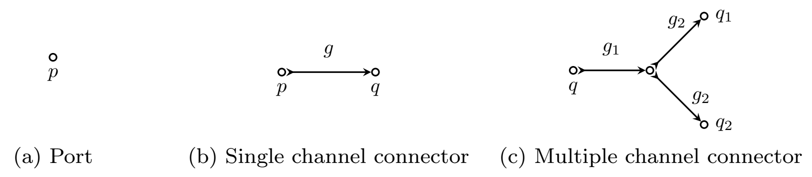
# Reo and constraint automata

After having introduced SDNs in Chapter 2, we now turn to Reo, a coordination language that we will use in the next chapter to model SDNs. The advantage of using Reo comes from its intuitive graphical syntax that comes equipped with precise automata-based semantics. The emphasis in Reo is on data and synchronization constraints, expressed by connectors, determining their behavior and importance during composition. In fact, Reo has already been used for modeling and analyzing a wide range of systems, including communication protocols [59], workflow systems [85], and control systems [9].

## 3.1 A short introduction to Reo

Reo is a coordination language for the compositional construction of component connectors [5]. Connectors in Reo are modeled as directed graphs describing the way data flows through a system. The nodes of the graph are called ports and can be used to connect a connector to other connectors. The behavior of a Reo system is determined by the way these ports are connected together, as well as the constraints that are imposed on the data flowing through the system. A port that is used exclusively as the source of edges is called an input port and represents an interface through which the connector receives data. Dually, a port that is only the target of the edges of a connector is called an output port (or sink), and represents an interface through which the connector offers data to the environment. Ports that are both source and target of some edges of the connector are "hidden" to the environment and have a ‘merge-replicate’ behavior: it accepts data from one (chosen non-deterministically) of the target edges of the connector and immediately sends it to all edges with the hidden port as a source. Isolated ports do not have any behavior by themselves. Note that, being an input or an output port is a property that depends on the connector, and in fact, when composing connectors, the same port can be input for one and output for the other.

Figure 3.1 shows the graphical representation of three simple Reo connectors: (a) is an isolated Reo port $p$, (b) is a connector representing a channel where the data flows from the input port $p$ to the output port $q$ under the constraints $g$, and (c) is a connector with an unnamed internal port that enables the flow of data from $q$ to both $q_1$ and $q_2$ under some data constraints. Besides data constraints, connectors specify also synchronization constraints that are visualized by using two different types of edges: synchronous and asynchronous. Specifically, all the connectors in this example use synchronous edges implying that flow between the input and output ports of the connector is logically happening at the same time, i.e. they synchronize.



(a) Port (b) Single channel connector (c) Multiple channel connector

Figure 3.1: Port, channel, and connector

Synchronization constraints in Reo are strict, meaning that they impose synchronization between ports connected by synchronous edges, and nothing else. No flow among other input and output ports belonging to a connector is allowed. This is of importance when two connectors are connected, as data is only allowed to flow between the common input and output ports, as well as on input or output ports that belong to one but not to the other connector.

## 3.2 Constraint automata

Constraint automata are a formalism to describe the behavior of Reo connectors and their composition [13]. Constraint automata can be thought of as conceptual generalizations of finite state automata where data and synchronization constraints influence which transitions are enabled on a given state.

We assume given a finite set $\mathbb{D}$ (ranged over by $d$) denoting all possible data that can be sent and received by Reo connectors, and a finite set $\mathbb{P}$ of ports names ranged over by $p, q$. Here $p \in \mathbb{P}$ is a port of a connector (a different concept than a port of a switch in an SDN). While later we will distinguish between input and output ports, for simplicity, and following the original presentation [13] we do not consider this difference here. Given a non-empty subset $N \subseteq \mathbb{P}$ of ports, we define the set $DC(N)$ by the following grammar::

$$g ::= true \mid p = d \mid g_1 \wedge g_2 \mid \neg g\,.$$

Here $p = d$ is the basic constraint imposing the data $d$ to be available at port $p \in N$. Basic constraints can be composed using the usual Boolean operation. Since $\mathbb{D}$ is a finite set, we sometimes use $p = q$ to denote the constraint $\bigwedge_{d \in Data}((p = d) \wedge (q = d))$. As usual, we write $p \neq d$ for $\neg(p = d)$.

**Definition 3.1** ([13]). *A constraint automaton is a tuple* $(S, N, \longrightarrow, S_0)$ *where*

- $S$ *is a finite set of states,*
- $N \subseteq \mathbb{P}$ *is a finite set of ports,*
- $\longrightarrow \subseteq S \times (N \times DC(N)) \times S$ *is a transition relation such that* $s \xrightarrow{P,g} s'$ *implies* $P \neq \emptyset$ *and* $g \in DC(P)$*, and*
- $S_0 \subseteq S$ *is the set of initial states.*

A transition $s_1 \xrightarrow{P,g} s_2$ is enabled in state $s_1$ when all ports in $P$ have data available (in the case of inputs) or no data (in the case of outputs). In this case, the automaton moves to state $s_2$ if all data constraints imposed by $g$ are satisfied.

For example, the three automata corresponding to the connectors of Figure 3.1 are defined as follows

(a) $A_1 = (\{s_1\}, \{p\}, \emptyset, \{s_1\})$;

(b) $A_2 = (\{s_2\}, \{p, q\}, \longrightarrow_2, \{s_2\})$ with $s_2 \xrightarrow{\{p,q\},\, g}_2 s_2$; and

(c) $A_3 = (\{s_3\}, \{q, q_1, q_2\}, \longrightarrow_3, \{s_3\})$ with $s_3 \xrightarrow{\{q,q_1,q_2\},\, g_1 \wedge g_2}_3 s_3$.

The semantics of a constraint automaton describe how it behaves and evolves over time. It is defined in terms of Timed Data Streams (TDS) [7] which associate with each port an infinite sequence of events representing the data flowing through the port and the time when this occurs. The transitions of the automaton then define the constraints that the data must satisfy, whereas synchronization relates to the time of the data flow at the synchronizing ports.

The composition of two constraint automata is a fundamental operation in the formal modeling and analysis of systems using Reo. It describes how data flow events of the two automata should align with each other.

**Definition 3.2.** *The product of the two constraint automata $A_1 = (S_1, N_1, \longrightarrow_1, S_{1,0})$ and $A_2 = (S_2, N_2, \longrightarrow_2, S_{2,0})$ with disjoint sets of states $S_1$ and $S_2$, is defined as the automaton*

$$A_1 \bowtie A_2 = (S, N, \longrightarrow, S_0)$$

*where $S = S_1 \times S_2$, $N = N_1 \cup N_2$, $S_0 = S_{1,0} \times S_{2,0}$ and $\longrightarrow$ is defined by the following three rules:*

$$\frac{s_1 \xrightarrow{P_1, g_1}_1 t_1 \quad s_2 \xrightarrow{P_2, g_2}_2 t_2 \ \text{ and } \ P_1 \cap N_2 = P_2 \cap N_1}{\langle s_1, s_2 \rangle \xrightarrow{P_1 \cup P_2,\, g_1 \wedge g_2} \langle t_1, t_2 \rangle}$$

$$\frac{s_1 \xrightarrow{P_1,\, g_1}_1 t_1 \ \text{ and } \ P_1 \cap N_2 = \emptyset}{\langle s_1, s_2 \rangle \xrightarrow{P_1,\, g_1} \langle t_1, s_2 \rangle} \quad \text{and} \quad \frac{s_2 \xrightarrow{P_2,\, g_2}_2 t_2 \ \text{ and } \ P_2 \cap N_1 = \emptyset}{\langle s_1, s_2 \rangle \xrightarrow{P_2,\, g_2} \langle s_1, t_2 \rangle}$$

Similar to the product automaton the above composition combines the states of the two automata into pairs to represent the possible configurations of the composed system. Common ports at each transition are matched based on their names. This step ensures that the time of flow events of one automaton coincides with the time of flow events of the other automaton. Also, the data coincides, as data constraints of both automata must be valid. Data flow at disjoint ports remains unchanged and happens in an interleaving fashion.

For example, the automata $A_2$ and $A_3$ above share only the port $q$. When composing them, the behaviors at the other ports are unchanged, but the flow of data at the common port $q$ must be the same. The resulting automaton is $A_1 \bowtie A_2 = (\{\langle s_2, s_3 \rangle\}, \{p, q, q_1, q_2\}, \longrightarrow, \{\langle s_2, s_3 \rangle\})$ with only one transition, namely:

$$\langle s_2, s_3 \rangle \xrightarrow{\{p,q,q_1,q_2\},\, g \wedge g_1 \wedge g_2} \langle s_2, s_3 \rangle \,.$$

Note that all four ports synchronize and that data in $p$ must satisfy the same constraints $g \wedge g_1 \wedge g_2$ as data at all other ports because it flows from $p$ to $q_1$ and $q_2$ passing through $q$. Formally there is no direction in this flow, and all ports are still open to the environment, while it may be desirable for port $q$ to be restricted to further communication with the environment after a composition. We will consider an extension addressing both points in the next section.

### 3.2.1 Constraint automata with memory

In many systems, the current behavior may depend on past events. Constraint automata lack the ability to retain information about past inputs or data, which can influence the current behavior of components. While this capability could be recovered by adding more states to the automaton in the case of finite data, the presentation of the automaton's behavior would enormously suffer and the number of states would increase exponentially. Moreover, in the presence of infinite data adding states would not suffice. By incorporating memory, an automaton gains the ability to become stateful by storing data in local internal variables. And these variables can be utilized in constraints just as input or output ports in ordinary constraints automata.

Next, we extend constraint automata by adding local variables to store data as recently studied in [57]. Except for the existential quantifier in constraints that could be easily added, the main difference with the approach below is a neater treatment of the values in the memory before and after the execution of a transition.

In addition to the set $\mathbb{D}$ of data and $\mathbb{P}$ of ports as used in constraint automata, we now assume another disjoint finite set $\mathbb{M}$ of memory cells ranged over by $m$. Further, let $\mathcal{F}$ be a set of function symbols and $\mathcal{P}$ a set of predicate symbols. Each predicate symbol and each function symbol comes with an arity, the number of arguments it expects. A term is defined as follows:

$$t ::= d \mid p \mid m \mid m^\bullet \mid f(t, ..., t)$$

The idea is that $p$ denotes the value at the port $p$, $m$ evaluates the value stored in the memory cell before the execution of a transition, and $m^\bullet$ to the value immediately after the execution of the transition. Evaluation of functions is as usual. Terms are used in constraints that are defined by the following predicate formulas:

$$\phi ::= \top \mid p = t \mid m = t \mid m^\bullet = t \mid P(t, ..., t) \mid \phi \wedge \phi \mid \neg\phi$$

The constraint $p = t$ denotes the equality between the value passing through the port $p$, and the value obtained by evaluating the term $t$; $m = t$ is the equality between the value stored in the memory $m$ before evaluating the constraint and the value denoted by $t$; and $m^\bullet = t$ is the equality between the value stored in the memory $m$ immediately after the evaluation of the constraint and the value denoted by $t$. The others are just the usual constraints.

In order to define the satisfaction of constraints, we assume the existence of a function $\hat{f}{:}\mathbb{D}^n \to \mathbb{D}$ for each $f \in \mathcal{F}$ of arity $n$, and a subset $\hat{P} \subseteq \mathbb{D}^m$ for each predicate symbol $P \in \mathcal{P}$ of arity $m$. For fixed sets of input ports $I \subseteq \mathbb{P}$, output ports $O \subseteq \mathbb{P}$ and hidden ports $H \subseteq \mathbb{P}$, the evaluation of constraint is defined by using the function $\alpha{:}I \cup O \cup \mathbb{M} \to \mathbb{D}_\perp$, and an environment $\eta{:}H \to \mathbb{D}_\perp$ assigning values to hidden ports. $\alpha$ is used for the visible components of a Reo connector. Here $\alpha(p)$ represents the value passing through port $p$ unless $\alpha(p) = \perp$ denotes the absence of data flow through port $p$. Similarly $\alpha(m)$ denotes the value stored in the memory cell $m$.

We denote by $At$ the set of all atoms $\alpha$. Note that $m^\bullet$ is not a part of an atom, because it refers to the value of $m$ after the evaluation of a transition. Therefore we need pairs of atoms, one for the current values stored in memory cells, and another for storing the side effect of an evaluation, i.e., the value of a memory cell after the evaluation. Evaluation of guards is defined inductively as follows:

$$\begin{array}{llll}
\alpha_1\alpha_2 \models_\eta & \top & & \\
\alpha_1\alpha_2 \models_\eta & p = t & \text{iff} & \alpha_1(p) = [\![t]\!]^\eta_{\alpha_1\alpha_2} \\
\alpha_1\alpha_2 \models_\eta & m = t & \text{iff} & \alpha_1(m) = [\![t]\!]^\eta_{\alpha_1\alpha_2} \\
\alpha_1\alpha_2 \models_\eta & m^\bullet = t & \text{iff} & \alpha_2(m) = [\![t]\!]^\eta_{\alpha_1\alpha_2} \\
\alpha_1\alpha_2 \models_\eta & P(t_1, ..., t_n) & \text{iff} & \langle [\![t_1]\!]^\eta_{\alpha_1\alpha_2}, ..., [\![t_n]\!]^\eta_{\alpha_1\alpha_2} \rangle \in \hat{P} \\
\alpha_1\alpha_2 \models_\eta & \phi_1 \wedge \phi_2 & \text{iff} & \alpha_1\alpha_2 \models_\eta \phi_1 \text{ and } \alpha_1\alpha_2 \models_\eta \phi_2 \\
\alpha_1\alpha_2 \models_\eta & \neg\phi & \text{iff} & \alpha_1\alpha_2 \not\models_\eta \phi
\end{array}$$

Finally, we define the evaluation of a guard without hidden ports as follows:

$$\alpha_1\alpha_2 \models \phi \text{ if and only if there is } \eta \text{ such that } \alpha_1\alpha_2 \models_\eta \phi.$$

Here $[\![t]\!]^{\eta}_{\alpha_1\alpha_2}$ denotes the value of the term $t$ and is defined inductively by:

$$\begin{array}{rcl}
[\![d]\!]^{\eta}_{\alpha_1\alpha_2} & = & d \\
[\![p]\!]^{\eta}_{\alpha_1\alpha_2} & = & \begin{cases} \alpha_1(p) & \text{if } p \in I \cup O \\ \eta(p) & \text{if } p \in H \end{cases} \\
[\![m]\!]^{\eta}_{\alpha_1\alpha_2} & = & \alpha_1(m) \\
[\![m^\bullet]\!]^{\eta}_{\alpha_1\alpha_2} & = & \alpha_2(m) \\
[\![f(t_1, ..., t_n)]\!]^{\eta}_{\alpha_1\alpha_2} & = & \hat{P}([\![t_1]\!]^{\eta}_{\alpha_1\alpha_2}, ..., [\![t_n]\!]^{\eta}_{\alpha_1\alpha_2})
\end{array}$$

We are now ready for the definition of constraint automata with memory cells describing operationally the behavior of a Reo connector.

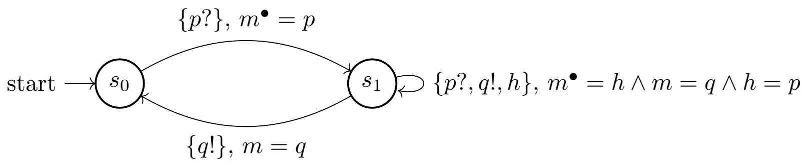
**Definition 3.3.** *A constraint automaton with memory is a tuple $(S, I, O, H, M, \longrightarrow, S_0)$ where $S$ is a finite set of states with $S_0 \subseteq S$ the set of initial states, $I, O, H \subseteq \mathbb{P}$ are sets of ports known by the automaton, $M \subseteq \mathbb{M}$ is the set of memory cells, and $\longrightarrow$ is a transition relation with $s \xrightarrow{N,\phi} s'$ denoting a transition from a state $s$ to $s'$ synchronizing a set of ports $N \subseteq I \cup O \cup H$ under the data constraint $\phi$. We assume that the ports appearing in $\phi$ are a subset of $N$ and the memory cells occurring in $\phi$ are a subset of $M$.*

An *execution* of a constraint automaton is described by means of infinite strings [55] in $At^{\omega}$. An infinite string $\alpha \cdot w$ is an execution starting from the state $s$, denoted by $\alpha \cdot w \in E(s)$ if and only if there is a transition $s \xrightarrow{N,\phi} s'$ such that the following three conditions hold:

1. $\forall p \in I \cup O$, $p \notin N$ iff $\alpha(p) = \perp$;
2. $w = \alpha' \cdot w'$ and $\alpha\alpha' \models \phi$;
3. $w \in E(s')$

By the above definition, a constraint of a transition $s \xrightarrow{N,\phi} s'$ is evaluated in an execution $\alpha \cdot w$ starting from a state $s$ with respect to its first two atoms. Furthermore, only the input and output ports in $N$ fire, meaning that a value passes through them as recorded by $\alpha$, and the rest of the string $w$ is an execution of the target state $s'$.

Consider the following constraint automaton:

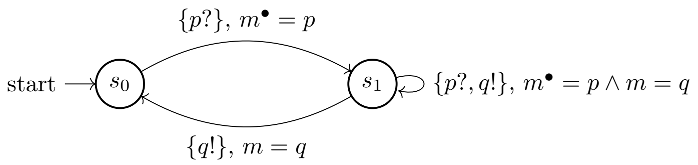

$\{p?\}$, $m^\bullet = p$
start
$s_0$
$s_1$
$\{p?, q!, h\}$, $m^\bullet = h \wedge m = q \wedge h = p$
$\{q!\}$, $m = q$

Here the suffixes "?" and "!" on a port are just syntactic means for indicating which ports belong to $I$ and $O$ respectively. The unmarked ports belong to $H$. An example of an execution of the above automaton starting from $s_0$ is the infinite string:

$$[p = 1, q = \perp, m = 22] \cdot [p = 3, q = 1, m = 1] \cdot [p = 5, q = 3, m = 3] \cdot$$
$$[p = \perp, q = 5, m = 5] \cdot [p = 7, q = \perp, m = 33] \cdot \ldots$$

Note that the value of the memory of the second element of the string is equal to the value at port $p$ of the first element, and the value of port $q$ of the second element. Similarly for the value of $p$ in the second element and the value of $q$ and the memory $m$ in the third element.

The above automaton has the same executions from the initial state as the following automaton without hidden ports.


$\{p?\}$, $m^\bullet = p$
start
$s_0$
$s_1$
$\{p?, q!\}$, $m^\bullet = p \wedge m = q$
$\{q!\}$, $m = q$

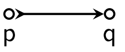
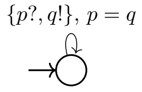
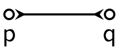
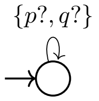
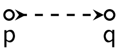
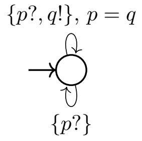
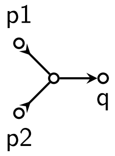
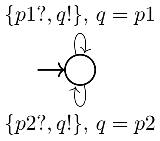
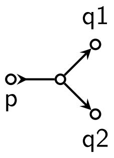
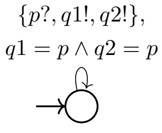
While in general it is not always possible to remove all hidden ports without modifying the set of executions, for simplicity and when there is no problem, in the sequel we will simplify a constraint automaton by removing hidden ports and obtaining an automaton with the same structure (states and transitions) and the same executions from its initial state.

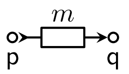
The language of a constraint automaton consists of the projection with respect to the input and output ports of all executions starting from the initial state. It represents the behavior of the automaton as visible from the environment. As such, only input and output ports are visible, but not hidden ports or memory cells. For example, the language accepted by the above two constraint automata is the same (i.e., they are language equivalent) and it includes the following infinite string

$$[p = 1, q = \perp] \cdot [p = 3, q = 1] \cdot [p = 5, q = 3] \cdot [p = \perp, q = 5] \cdots .$$

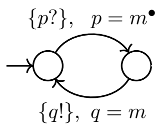
### 3.2.2 Basic and more complex connectors

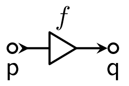
In the following table, we associate constraint automata with memory to a few basic Reo connectors here described by means of their usual graphical representation [5, 65].

| | | |
|---|---|---|
|  | The *synchronous connector* accepts data from its input port $p$, and it passes synchronously to its output port $q$. |  |
|  | The *synchronous drain* has two input ports $p$ and $q$, from which it accepts any data, but only when the two ports can be synchronized. The data received as input is not important, only ports' synchronization matters. |  |
|  | The *lossy synchronous connector* accepts data from its input port $p$, it either passes them synchronously to its output port $q$, or loses them in the channel without any reason. |  |
|  | The *non-deterministic merger* receives data from either its input ports $p1$ or $p2$ and sends it to the output port $q$ synchronously. If data is available at both input ports, only one of them is chosen non-deterministically. |  |
|  | The *replicator connector* receives data from its input port $p$ and replicates it to both output ports $q1$ and $q2$. |  |

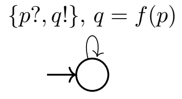


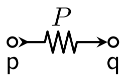
The *FIFO1 connector* receives data from the input port $p$ if the internal buffer $m$ is empty. The data is stored in the buffer, which can only contain at most one data item. When $m$ is full its content flows to the output port $q$ and it becomes empty. The behavior of a similar connector with dots inside the box is represented by the automaton with the other state as the starting state.



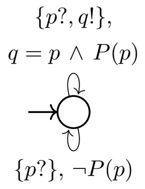


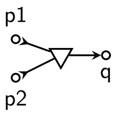
The *transformer connector* applies a function $f$ to a data item received through its input port $p$, and synchronously offers the data resulting from evaluating $f(p)$ to it output port $q$.



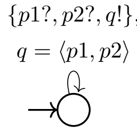


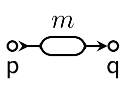
The pattern of a *filter connector* $P \subseteq Data$ specifies the type of data items that can be transmitted through the channel. Any value $d \in P$ is accepted through its input port $p$ if the output port $q$ can simultaneously dispense $d$; all data items that do not satisfy $P$ are always accepted at the input $p$ but they are immediately lost.



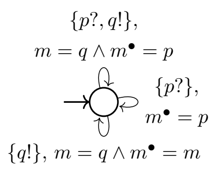


The *pairing connector* accepts two data items through its input ports $p1$ and $p2$, and synchronously output their pairing thorugh the port $q$.





The *variable connector* output the content of $m$ at the port $q$ and update $m$ with the data at the input $p$. If only the input port is enabled then update $m$ with the data received, while if only the output port $q$ is enabled, then it sends the value of $m$ to $q$ and does not change $m$.


$\{p?, q!\},$
$m = q \wedge m^\bullet = p$
$\{p?\},$
$m^\bullet = p$
$\{q!\},\ m = q \wedge m^\bullet = m$

Note that the pairing connector is similar to a transformer but with two inputs and using as a function the pairing $\langle -, - \rangle$. In all automata in the table, we assume that the ports known by each automaton are those used in the connectors.

We are now ready to extend the product of constraint automata given in the previous section so as to consider different types of ports, internal local memory, and restrictions of the ports used during synchronization. As before, the join operation guarantees synchronization and the same data flow at common ports but leaves the behavior at ports unknown to one of the automata unchanged.

**Definition 3.4.** *Assume $S_1 \cap S_2 = \emptyset$, $M_1 \cap M_2 = \emptyset$, $H_1 \cap (I_2 \cup O_2 \cup H_2) = \emptyset$ and $H_2 \cap (I_1 \cup O_1 \cup H_1) = \emptyset$. The product of the two constraint automata $A_1 =$ $(S_1, I_1, O_1,$
$H_1, M_1, \longrightarrow_1, S_{1,0})$ and $A_2 = (S_2, I_2, O_2, H_2, M_2, \longrightarrow_2, S_{2,0})$ is defined as the following automaton:*

$$A_1 \bowtie A_2 = (S, I, O, H, M, \longrightarrow, S_0)$$

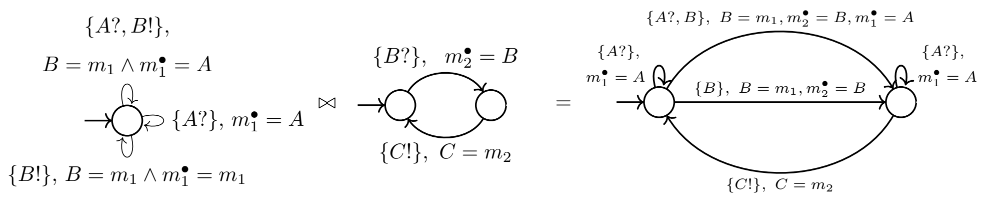
*where $S = S_1 \times S_2$, $S_0 = S_{1,0} \times S_{2,0}$, $M = M_1 \cup M_2$ $I = (I_1 - O_2) \cup (I_2 - O_1)$, $O = (O_1 - I_2) \cup (O_2 - I_1)$, $H = (I_1 \cap O_2) \cup (I_2 \cap O_1) \cup H_1 \cup H_2$, and $\longrightarrow$ is defined by the following rules:*

$$\frac{s_1 \xrightarrow{N_1, \phi_1}_1 t_1 \quad s_2 \xrightarrow{N_2, \phi_2}_2 t_2 \ \text{and} \ Prt_1 \cap N_2 = Prt_2 \cap N_1}{\langle s_1, s_2 \rangle \xrightarrow{N_1 \cup N_2,\, \phi_1 \wedge \phi_2} \langle t_1, t_2 \rangle}$$

$$\frac{s_1 \xrightarrow{N_1, \phi_1}_1 t_1 \ \text{and} \ Prt_2 \cap N_1 = \emptyset}{\langle s_1, s_2 \rangle \xrightarrow{N_1,\, \phi_1} \langle t_1, s_2 \rangle} \quad \text{and} \quad \frac{s_2 \xrightarrow{N_2, \phi_2}_2 t_2 \ \text{and} \ Prt_1 \cap N_2 = \emptyset}{\langle s_1, s_2 \rangle \xrightarrow{N_1,\, \phi_1} \langle s_1, t_2 \rangle}$$

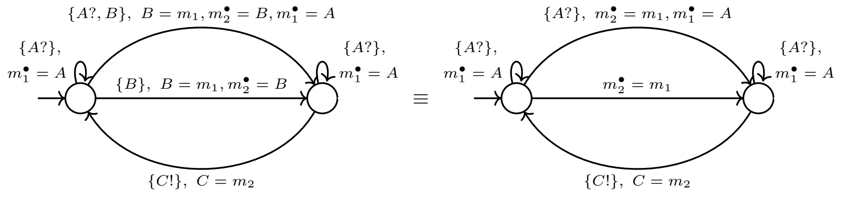
*Here $Prt_1 = I_1 \cup O_1$, and $Prt_2 = I_2 \cup O_2$.*

**Example 1.** Figure 3.2 shows an example of a composition of a *variable* (on the left) on ports $\{A?, B!\}$ with a *FIFO1* connector (second automata from the left) acting on port $\{B?, C!\}$. The result is a new automaton with $B$ as the hidden port (the last automaton), which however is the language equivalent to the automaton of a non-deterministic merger (in Figure 3.3 on the right) on ports $\{A?, C!\}$. Initially, *variable* stores a value of $m_1$, and $FIFO1$ starts with $m_2 = \perp$.



$\{A?, B!\}$,
$B = m_1 \wedge m_1^\bullet = A$
$\{A?\}$, $m_1^\bullet = A$
$\{B!\}$, $B = m_1 \wedge m_1^\bullet = m_1$
$\bowtie$
$\{B?\}$, $m_2^\bullet = B$
$\{C!\}$, $C = m_2$
$=$
$\{A?, B\}$, $B = m_1, m_2^\bullet = B, m_1^\bullet = A$
$\{A?\}$, $m_1^\bullet = A$
$\{B\}$, $B = m_1, m_2^\bullet = B$
$\{A?\}$, $m_1^\bullet = A$
$\{C!\}$, $C = m_2$

Figure 3.2: The composition of two automata



$\{A?, B\}$, $B = m_1, m_2^\bullet = B, m_1^\bullet = A$
$\{A?\}$, $m_1^\bullet = A$
$\{B\}$, $B = m_1, m_2^\bullet = B$
$\{A?\}$, $m_1^\bullet = A$
$\{C!\}$, $C = m_2$
$\equiv$
$\{A?\}$, $m_2^\bullet = m_1, m_1^\bullet = A$
$\{A?\}$, $m_1^\bullet = A$
$m_2^\bullet = m_1$
$\{A?\}$, $m_1^\bullet = A$
$\{C!\}$, $C = m_2$

Figure 3.3: Removing hidden ports

Note that the port $B$ is a hidden port in the resulting automaton because it is an output port of one connector and an input port of the other. It is not hard to see that the join operation is associative and commutative. The full conjunction process is as follows.

Assume the first automaton is named $A_1$, where the state in $A_1$ is 1, the transition above is $\mathfrak{t}_1$, the right transition is $\mathfrak{t}_2$, and the transition below is $\mathfrak{t}_3$. The second automaton is called $A_2$, the first state from left is named 2, the second state from left is 3, the transition from state 2 to state 3 is $\mathfrak{t}_4$, the transition from state 3 to state 2 is $\mathfrak{t}_5$, then we have the following information:

For $A_1$:
$I_1 = \{A\}, O_1 = \{B\}, Q_1 = \{1\}, q_1 = 1, Prt_1 = \{A, B\}, H = \emptyset, M_1\colon m_1 = v$
the transition $\mathfrak{t}_1$: $1 \xrightarrow{N_1,\phi_1} 1$, $N_1 = \{A, B\}$, $\phi_1$: $B = m_1 \wedge m_1^\bullet = A$
the transition $\mathfrak{t}_2$: $1 \xrightarrow{N_2,\phi_2} 1$, $N_2 = \{A\}$, $\phi_2$:$m_1^\bullet = A$
the transition $\mathfrak{t}_3$: $1 \xrightarrow{N_3,\phi_3} 1$, $N_3 = \{B, \}$, $\phi_3$: $B = m_1 \wedge m_1^\bullet = m_1$

For $A_2$:
$I_2 = \{B\}, O_2 = \{C\}, Q_2 = \{2, 3\}, q_2 = 2, Prt_2 = \{B, C\}, H = \emptyset, M_2\colon m_2 = \perp$

the transition $\mathfrak{t}_4$: $2 \xrightarrow{N_4,\phi_4} 3$, $N_4 = \{B\}$, $\phi_4$: $m_2^\bullet = B$
the transition $\mathfrak{t}_5$: $3 \xrightarrow{N_5,\phi_5} 2$, $N_5 = \{C\}$, $\phi_5$: $C = m_2$

Applying to Definition 3.4, the composition of $A_1$ and $A_2$ is $(Q, I, O, H, M, \longrightarrow, q_0)$, where

$Q = Q_1 \times Q_2 = \{\langle 1,2\rangle, \langle 1,3\rangle\}$,
$I = (I_1 - O_2) \cup (I_2 - O_1) = (\{A\} - \{C\}) \cup (\{B\} - \{B\}) = \{A\} \cup \emptyset = \{A\}$,
$O = (O_1 - I_2) \cup (O_2 - I_1) = (\{B\} - \{B\}) \cup (\{C\} - \{A\}) = \emptyset \cup \{C\} = \{C\}$,
$H = (I_1 \cap O_2) \cup (I_2 \cap O_1) \cup H_1 \cup H_2 = (\{A\} \cap \{C\}) \cup (\{B\} \cap \{B\}) \cup \emptyset \cup \emptyset = \{B\}$,
$M = [m_1 = v, m_2 = \perp]$.

The transitions are defined as follows:

► $\mathfrak{t}_1$ join with $\mathfrak{t}_4$:
$\because N_1 \cap Prt_2 = N_4 \cap Prt_1 = \{B\}$,
$\therefore \mathfrak{t}_1$ join with $\mathfrak{t}_4$: $\langle 1,2\rangle \xrightarrow{N_6,\,\phi_6} \langle 1,3\rangle$,
$N_6$ (*i.e.*, $N_1 \cup N_4) = \{A, B\}$, $\phi_6 = \phi_1 \wedge \phi_4$: $B = m_1 \wedge m_1^\bullet = A \wedge m_2^\bullet = B$
► $\mathfrak{t}_1$ join with $\mathfrak{t}_5$:
$\because N_1 \cap Prt_2 = \{B\}$, $N_5 \cap Prt_1 = \emptyset$,
$\therefore \mathfrak{t}_1$ join with $\mathfrak{t}_5$: $\langle 1,3\rangle \xrightarrow{N_5,\,\phi_5} \langle 1,2\rangle$, $N_5 = \{C\}$, $\phi_5$: $C = m_2$
► $\mathfrak{t}_2$ join with $\mathfrak{t}_4$:
$\because N_2 \cap Prt_2 = \emptyset$, $N_4 \cap Prt_1 = \{B\}$,
$\therefore \mathfrak{t}_2$ join with $\mathfrak{t}_4$: $\langle 1,2\rangle \xrightarrow{N_2,\phi_2} \langle 1,2\rangle$, $N_2 = \{A\}$, $\phi_2$: $m_1^\bullet = A$
► $\mathfrak{t}_2$ join with $\mathfrak{t}_5$:
$\because N_2 \cap Prt_2 = \emptyset$, $N_5 \cap Prt_1 = \emptyset$,
$\therefore \mathfrak{t}_2$ join with $\mathfrak{t}_5$: $\langle 1,3\rangle \xrightarrow{N_2,\phi_2} \langle 1,3\rangle$, $N_2 = \{A\}, \phi_2$: $m_1^\bullet = A$,
$\langle 1,3\rangle \xrightarrow{N_5,\phi_5} \langle 1,2\rangle$, $N_5 = \{C\}, \phi_5$: $C = m_2$
► $\mathfrak{t}_3$ join with $\mathfrak{t}_4$:
$\because N_3 \cap Prt_2 = N_4 \cap Prt_1 = \{B\}$,
$\therefore \mathfrak{t}_3$ join with $\mathfrak{t}_4$: $\langle 1,2\rangle \xrightarrow{N_7,\phi_7} \langle 1,3\rangle$,
$N_7$(*i.e.*, $N_3 \cup N_4) = \{B\}$, $\phi_7$:$m_2^\bullet = B \wedge B = m_1 \wedge m_1^\bullet = m_1$
► $\mathfrak{t}_3$ join with $\mathfrak{t}_5$:

$\because N_3 \cap Prt_2 = \{B\},\ N_5 \cap Prt_1 = \emptyset,$
$\therefore \mathfrak{t}_3$ join with $\mathfrak{t}_5$: $\langle 1,3\rangle \xrightarrow{N_5,\phi_5} \langle 1,2\rangle,\ N_5 = \{C\},\ \phi_5\colon C = m_2$

The list above has seven transitions after the conjunction, with regards to some transitions that are overlapping, we keep one of them and remove the others. For instance, the second transition of the list $A_1 \bowtie A_2$ appears in the fourth and last one, we only keep one transition as the result. Port $B$ is removed in the final automaton since it becomes the hidden port.

**Example 2.** As another more complex example, in Figure 3.4 we introduce a three-input sequencer that regulates the flow of data from the input ports $p1, p2$, and $p3$, in sequential order, one after the other. Similar sequencers can be defined for any number of ports [41]. The connector is obtained by properly composing three synchronous drain connectors (connecting $p1$ with $i_2$, $p2$ with $j_2$, and $p3$ with $k_2$), one non-deterministic merger (connecting $j_1$ and $j_3$ with $j2$) two replicators (connecting $i_1$ with $i_2$ and $i_3$, and $k_1$ with $k_2$ and $k_3$), two FIFO1 connectors (one from $i_3$ to $j_1$, and another from $j_3$ to $k_1$), and finally a FIFO1 connector from $k_3$ to $i_1$ with a buffer initially storing a token data.

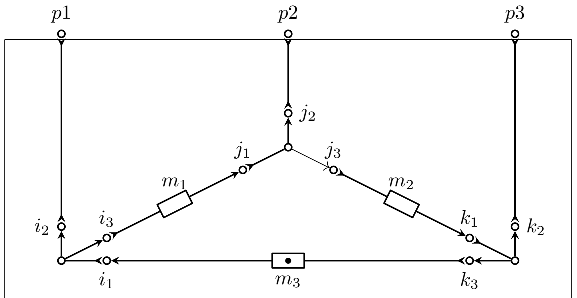
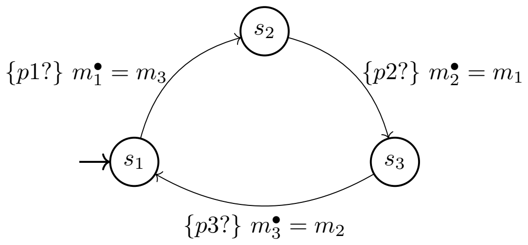


Figure 3.4: A three-input sequencer

Starting from the constraint automata of the basic connectors we have given above, and composing them according to Figure 3.4, we obtain the following three-state constraint automaton, where we have removed all hidden ports.


$s_1$
$s_2$
$s_3$
$\{p1?\}\ m_1^\bullet = m_3$
$\{p2?\}\ m_2^\bullet = m_1$
$\{p3?\}\ m_3^\bullet = m_2$