# Constraint-based analysis of business process models

Changizi, B.

# 6

# A Constraint-Based Semantics Framework for Reo

## 6.1 Introduction

In Chapter 5, we presented our approach for automatic transformation of business process models into Reo [CKA10]. This enables the use of Reo analysis methods and tools on these processes that originally were not expressed in Reo. Performing analysis on a Reo connector requires the behavior of the connector expressed in one of the formal semantics of Reo.

Each of these formal semantics comes with a set of definitions and operators, which enable calculating semantics of a Reo connector. The straight-forward algorithms of supporting tools for automating this process are developed based on these definitions. These custom algorithms are computationally expensive and not optimized. As a result, in practice the size of a connector they can support is small.

Another inherent limitation of these algorithms stem from that they model data explicitly. As a consequence, in practice the set of input data needs to be limited to a predefined small set. This holds even for connecters with no data-sensitive

components, which shows the same behavior for each data item.

Even though different formal semantics of a Reo connector describe the behavior of the same model, since each of them focuses on some behavioral aspects such as context-sensitivity or data-awareness, and ignores some other aspects, it is possible that one aspects of its semantics describes some behavior that another semantics considers invalid. A classical example of this case is when a *lossySync* channel is connected to a $FIFO_1$ channel. The constraint automata and the coloring semantics for this example describe different behavior.

In this chapter, we present a constraint-based framework to derive formal semantics of a Reo connector. We form a constraint by encoding the behavior of constructs of the connector.

Our framework eliminates the result of expressiveness gap among Reo formal semantics by incorporating more than one semantics in deriving the behavior of a Reo connector. This way, we transform problem of calculating formal semantics of a Reo connector into a constraint satisfaction problem, for which efficient and optimized methods and tools exist. We use the symbolic approach to deal with data, i.e, rather than dealing with concrete values, we split the data domain to ranges for which the connector exhibits different behavior.

This work is a necessary step for providing fully automated model checking for data-aware and context-dependent Reo connectors. It can be seen as a generalization of the constraint-based framework presented in [Pro11], that is used as a base for Reo's distributed execution engine. However, there are major differences between them. For instance, the framework for the Reo execution engine only provide support for synchrony and context-sensitivity, while our method deals with priority and data-constraints as well.

## 6.2 Reo constraint satisfaction problem (RCSP)

In this section, we extend the constraint-based framework in [Pro11] to incorporate all behavioral dimensions addressed by various semantic models for Reo. In our framework, we denote each of these elements by variables over their proper domains.

We relate these variables to each other and restrict possible values they can assume using constraints whose solutions give the underlying formal semantics of the network. In this section, we deal only with connectors whose semantics can be expressed in CASM or CC. Later, we extend our framework to also support priority.

Let $\mathcal{N} = \mathcal{N}^{src} \cup \mathcal{N}^{mix} \cup \mathcal{N}^{snk}$ be the global set of nodes, $\mathcal{M}$ the global set of state memory variables, and $\mathcal{D}$ the global set of numerical data values. The set

of primitive ends $\mathcal{P}$ consists of all primitive ends $p$ derived from $\mathcal{N}$ by marking its elements with superscripts $c$ and $k$, according to the following grammar:

$$p ::= r^c \mid s^k$$

where $r \in \mathcal{N}^{src} \cup \mathcal{N}^{mix}$ and $s \in \mathcal{N}^{snk} \cup \mathcal{N}^{mix}$. Observe that the primitive ends $n^c$ and $n^k$ connect on the common node $n$.

Let $p \in \mathcal{P}$, $n \in \mathcal{N}$ and $m \in \mathcal{M}$ be a primitive end, a node, and a state memory variable, respectively. A free variable $v$ that occurs in the constraints encoding the behavior of a Reo network has one of the following forms:

- $\tilde{n}$ ranges over $\{\top, \bot\}$ to show presence or absence of flow on the node $n$.
- $\hat{n}$ ranges over $\mathcal{D}$ to represent the data value passing through the node $n$.
- $\mathring{m}, \mathring{m}'$ range over $\{\top, \bot\}$ to denote whether or not the state memory variable $m$ is defined in, respectively, the source and the target states of the transition to which the encoded guard belongs.
- $\hat{m}, \hat{m}'$ range over $\mathcal{D}$ to represent the values of the state memory variable $m$ in, respectively, the source and the target states of the transition to which the encoded guard belongs.
- $p^{\triangleright}$ ranges over $\{\top, \bot\}$ to state that the reason for lack of data-flow through the primitive end $p$ originates from the primitive to which $p$ belongs or the context (of this primitive).

Note that not all of the introduced variables are required for encoding the behavior of every Reo network. In presence of context-dependent primitives like *lossySync* or in priority-sensitive networks, constraints include variables of the form $p^{\triangleright}$. For the stateful elements such as $FIFO_1$, variables like $\mathring{m}, \mathring{m}', \hat{m}$, and $\hat{m}'$ appear in the constraints.

Observe that the interpretation of some of the mentioned variables depends on the values of other variables. Referring to the variable $p^{\triangleright}$ makes sense only if $\tilde{n} = \bot$, where $p = n^c$ or $p = n^k$ (i.e., the primitive end $p$ belongs to the node $n$); and $\hat{n}$, $\hat{m}$ and $\hat{m}'$ make sense only if $\tilde{n} = \top$, $\mathring{m} = \top$ and $\mathring{m}' = \top$, respectively.

The grammar for a constraint $\Psi$ encoding the behavior of a Reo network is as follows:

$$
\begin{array}{lcll}
t & ::= & \hat{n} \mid \hat{m} \mid \hat{m}' \mid d \mid t \circledast\ d & \text{(terms)} \\
a & ::= & \tilde{n} \mid p^{\triangleright} \mid \mathring{m} \mid \mathring{m}' \mid t = t \mid t < t & \text{(atoms)} \\
\boldsymbol{\psi} & ::= & \top \mid a \mid \neg\boldsymbol{\psi} \mid \boldsymbol{\psi} \wedge\ \boldsymbol{\psi} & \text{(formulae)}
\end{array}
$$

where $d \in \mathcal{D}$ is a constant, $\circledast \in \{+, -, *, /, \%, \hat{}\}$, and $p$ is either of the form $n^c$ or $n^k$.

A solution to a formula $\boldsymbol{\psi}$ is defined over the variable sets $V \times V_d$, where the variables in $V$ are mapped to a value in $\{\bot, \top\}$ and values in $V_d$ are mapped to subsets of $D$. The satisfaction rules for a solution $\langle \delta, \delta_d \rangle$ are defined as follows:

| | |
|---|---|
| $\langle \delta, \delta_d \rangle \vDash \top$ | always |
| $\langle \delta, \delta_d \rangle \vDash \tilde{n}$ | iff $\delta(\tilde{n}) = \top$ |
| $\langle \delta, \delta_d \rangle \vDash p^{\triangleright}$ | iff $\delta(p^{\triangleright}) = \top$ |
| $\langle \delta, \delta_d \rangle \vDash \mathring{m}$ | iff $\delta(\mathring{m}) = \top$ |
| $\langle \delta, \delta_d \rangle \vDash \mathring{m}'$ | iff $\delta(\mathring{m}') = \top$ |
| $\langle \delta, \delta_d \rangle \vDash P(t_1, t_2, ..., t_n)$ | iff $(\delta_d(t_1), \delta_d(t_2), ..., \delta_d(t_n)) \subseteq I(P(t_1, t_2, ..., t_n))$ |
| $\langle \delta, \delta_d \rangle \vDash \boldsymbol{\psi}_1 \wedge \boldsymbol{\psi}_2$ | iff $\langle \delta, \delta_d \rangle \vDash \boldsymbol{\psi}_1 \wedge \langle \delta, \delta_d \rangle \vDash \boldsymbol{\psi}_2$ |
| $\langle \delta, \delta_d \rangle \vDash \neg\boldsymbol{\psi}$ | iff $\langle \delta, \delta_d \rangle \nvDash \boldsymbol{\psi}$ |

There exists an associated interpretation, $I(P) \subseteq 2^{D^n}$, for each $n$-ary predicate $P$.

**Definition 6.2.1 (Reo constraint satisfaction problem)** *A Reo constraint satisfaction problem (RCSP) is a tuple $\langle \mathcal{P}, \mathcal{M}, M_0, \mathcal{V}, C \rangle$, where:*

- *$\mathcal{P}$ is a finite set of primitive ends.*
- *$\mathcal{M}$ is a finite set of state memory variables.*
- *$M_0 \subseteq \mathcal{M}$ is a set of state memory variables that define the initial configuration of a Reo network.*
- *$\mathcal{V}$ is a set of variables $v$ defined by the grammar*

$$v ::= \tilde{n} \mid p^{\triangleright} \mid \mathring{m} \mid \mathring{m}' \mid \hat{n} \mid \hat{m} \mid \hat{m}'$$

*for $n \in \mathcal{N}, p \in \mathcal{P}$, and $m \in \mathcal{M}$. The values that the variables of the forms $\hat{n}, \hat{m}$, and $\hat{m}'$ can assume are subsets of $\mathcal{D}$, and the other variables are Boolean, with values in $\{\top, \bot\}$.*

- $C = \{C_1, C_2, ..., C_m\}$ *is a finite set of constraints, where each* $C_i$ *is a constraint given by the grammar* $\Psi$ *involving a subset of variables* $V_i \subseteq \mathcal{V}$.

**Example 6.2.1** *The RCSP of a* sync *channel with the source end* $a$ *and the sink end* $b$ *is* $\langle \{a, b\}, \emptyset, \emptyset, \{\tilde{a}, \tilde{b}, \hat{a}, \hat{b}\}, \tilde{a} \Leftrightarrow \tilde{b} \wedge \tilde{a} \Rightarrow (\hat{a} = \hat{b}) \rangle$. *The solutions for this constraint problem give the behavior of the* sync *channel as the channel allows data-flow on its source end iff its sink end can dispense it simultaneously (which agrees with the semantics of this channel as defined in other formal models of Reo). In case of data-flow, the values of the data items passing through the ends of this channel are equal.*

We obtain the constraints corresponding to a Reo network by composing the RCSPs of its constituents as defined below.

**Definition 6.2.2 (Composition)** *The composition of two RCSPs* $\rho_1 = \langle \mathcal{P}_1, \mathcal{M}_1, M_{0,1}, \mathcal{V}_1, C_1 \rangle$ *and* $\rho_2 = \langle \mathcal{P}_2, \mathcal{M}_2, M_{0,2}, \mathcal{V}_2, C_2 \rangle$ *is defined as follows:*

$$\rho_1 \odot \rho_2 = \langle \mathcal{P}_1 \cup \mathcal{P}_2, \mathcal{M}_1 \cup \mathcal{M}_2, M_{0,1} \cup M_{0,2}, \mathcal{V}_1 \cup \mathcal{V}_2, C_1 \wedge C_1 \rangle$$

However, connecting two Reo networks must not introduce incorrect data-flow possibilities. This is done by enforcing a restriction on the possible solutions through the following axiom:

**Axiom 6.2.1 (Mixed node axiom)** *When two Reo networks connect on the common node* $x$, *where* $x^c$ *is a source end in one network and* $x^k$ *is a sink end in the other, the following constraint must hold:*

$$\neg \tilde{x} \Leftrightarrow (x^{c\triangleright} \vee x^{k\triangleright})$$

The *mixed node axiom*, which applies to all mixed nodes in a network, states that a node $x$ cannot produce the reason for no-flow all by itself.

### 6.2.1 Encoding Reo elements in RCSPs

Table 6.2.2 summarizes the constraint encodings associated with commonly used Reo elements. If a Reo network does not contain any context-dependent channel, the variables encoding the context-dependency can be ignored in its RCSP. Table 6.2.1 shows the encoding of Reo elements from Table 6.2.2 where the context variables are removed. Note that in these tables, $a$ and $b$ denote the source and the sink ends of a primitive, respectively, and that *dom* refers to the domain of the given function

**Table 6.2.1**: Context-independent encoding of Reo primitives

| Channel | Constraints |
|---|---|
|  | $\psi_{Sync}(a,b) : \tilde{a} \Leftrightarrow \tilde{b} \wedge \tilde{a} \Rightarrow (\hat{a} = \hat{b})$ |
|  | $\psi_{SyncDrain}(a_1,a_2) : \tilde{a}_1 \Leftrightarrow \tilde{a}_2$ |
|  | $\psi_{AsyncDrain}(a_1,a_2) : \neg(\tilde{a}_1 \wedge \tilde{a}_2)$ |
|  | $\psi_{LossySync} : \tilde{b} \Rightarrow \tilde{a} \wedge \tilde{b} \Rightarrow (\hat{a} = \hat{b})$ |
|  | $\psi_{Merger}(a_{0..i},b) : \tilde{b} \Leftrightarrow (\bigvee_i \tilde{a}_i) \bigwedge_{j,j\neq i} \neg(\tilde{a}_i \wedge \tilde{a}_j) \wedge \tilde{a}_i \Rightarrow (\hat{a}_i = \hat{b})$ |
|  | $\psi_{Replicator}(a,b_{0..i}) : \tilde{a} \Leftrightarrow (\bigwedge_i \tilde{b}_i) \wedge \tilde{a} \Rightarrow (\bigwedge_i (\hat{b}_i = \hat{a}))$ |
|  | $\psi_{Router}(a,b_{0..i}) : \tilde{a} \Leftrightarrow (\bigvee_i \tilde{b}_i) \bigwedge_{j,j\neq i} \neg(\tilde{b}_i \wedge \tilde{b}_j) \wedge \tilde{b}_i \Rightarrow (\hat{b}_i = \hat{a})$ |
|  | $\psi_{FIFO_1}(a,b,m) : \tilde{a} \Rightarrow (\neg\mathring{m} \wedge \mathring{m}' \wedge (\hat{m}' = \tilde{a})) \wedge \tilde{b} \Rightarrow (\mathring{m} \wedge \neg\mathring{m}' \wedge (\hat{m} = \hat{b})) \wedge (\neg\tilde{a} \wedge \neg\tilde{b}) \Rightarrow (\mathring{m} \Leftrightarrow \mathring{m}' \wedge \mathring{m} \Rightarrow (\hat{m} = \hat{m}))$ |
|  | $\psi_{Filter}(a,b,P) = \tilde{b} \Rightarrow (\tilde{a} \wedge \hat{b} \in dom(P) \wedge P(\hat{a}) \wedge (\hat{a} = \hat{b}))$ |
|  | $\psi_{Transformer}(a,b,f) = \tilde{b} \Rightarrow (\tilde{a} \wedge \hat{b} \in dom(f)) \wedge \tilde{b} \Rightarrow (\hat{b} = f(\hat{a}))$ |

**Table 6.2.2**: Context-dependent encoding of Reo primitives

| Channel | Constraints |
|---|---|
|  | $\psi_{Sync}(a,b) : \tilde{a} \Leftrightarrow \tilde{b} \wedge \tilde{a} \Rightarrow (\hat{a} = \hat{b}) \wedge \neg(a^{c\triangleright} \wedge b^{k\triangleright})$ |
|  | $\psi_{SyncDrain}(a_1,a_2) : \tilde{a}_1 \Leftrightarrow \tilde{a}_2 \wedge \neg(a_1^{c\triangleright} \wedge a_2^{c\triangleright})$ |
|  | $\psi_{AsyncDrain}(a_1,a_2) : \tilde{a}_1 \Rightarrow (\neg\tilde{a}_2 \wedge a_2^{c\triangleright}) \wedge \tilde{a}_2 \Rightarrow (\neg\tilde{a}_1 \wedge a_1^{c\triangleright})$ |
|  | $\psi_{LossySync}(a,b) : \tilde{b} \Rightarrow \tilde{a} \wedge \tilde{b} \Rightarrow (\hat{a} = \hat{b}) \wedge \neg a^{c\triangleright} \wedge \neg\tilde{a} \Rightarrow b^{k\triangleright}$ |
|  | $\psi_{Merger}(a_{0..i},b) : \tilde{a}_i \Leftrightarrow \tilde{b} \wedge \tilde{a}_i \Rightarrow (\hat{a}_i = \hat{b}) \wedge \neg\tilde{b} \Rightarrow ((\neg b^{k\triangleright} \bigwedge_i a_i^{c\triangleright}) \vee (b^{k\triangleright} \wedge \neg a_i^{c\triangleright} \bigwedge_{j,j!=i} a_j^{k\triangleright}))$ |
|  | $\psi_{Replicator}(a,b_{0..i}) : \tilde{a} \Leftrightarrow \bigwedge_i \tilde{b}_i \wedge (\tilde{a} \Rightarrow \bigwedge_i (\hat{b}_i = \hat{a})) \wedge \neg\tilde{a} \Rightarrow ((\neg a^{c\triangleright} \bigwedge_i b_i^{k\triangleright}) \vee (\neg b_i^{k\triangleright} \bigwedge_{j,j\neq i} b_j^{k\triangleright} \wedge a^{c\triangleright}))$ |
|  | $\psi_{Router}(a,b_{0..i}) : \tilde{a} \Leftrightarrow (\bigvee_i \tilde{b}_i) \bigwedge_{j,j\neq i} \neg(\tilde{b}_i \wedge \tilde{b}_j) \wedge \tilde{b}_i \Rightarrow (\hat{b}_i = \hat{a}) \wedge \tilde{a} \Leftrightarrow (\neg a^{c\triangleright} \vee \neg(\bigvee_i b_i^{k\triangleright}))$ |
|  | $\psi_{FIFO_1}(a,b,m) : \tilde{a} \Rightarrow (\neg\mathring{m} \wedge \mathring{m}' \wedge (\hat{m}' = \hat{a})) \wedge \tilde{b} \Rightarrow (\mathring{m} \wedge \neg\mathring{m}' \wedge (\hat{m} = \hat{b})) \wedge (\neg\tilde{a} \wedge \neg\tilde{b}) \Rightarrow (\mathring{m} \Leftrightarrow \mathring{m}' \wedge \mathring{m} \Rightarrow (\hat{m} = \hat{m}')) \wedge \neg\mathring{m} \Rightarrow b^{k\triangleright} \wedge \mathring{m} \Rightarrow a^{c\triangleright}$ |
|  | $\psi_{Filter}(a,b,P) = \tilde{b} \Rightarrow (\tilde{a} \wedge \hat{a} \in dom(P) \wedge P(\hat{a})) \wedge \tilde{b} \Rightarrow (\hat{a} = \hat{b}) \wedge (\neg\tilde{a} \Rightarrow (\neg a^{c\triangleright} \Leftrightarrow b^{k\triangleright})) \wedge (\tilde{a} \wedge \neg\tilde{b} \Rightarrow b^{k\triangleright})$ |
|  | $\psi_{Transformer}(a,b,f) = \tilde{b} \Rightarrow (\tilde{a} \wedge \hat{a} \in dom(f)) \wedge \tilde{b} \Rightarrow (\hat{b} = f(\hat{a})) \wedge \neg(a^{c\triangleright} \wedge b^{k\triangleright})$ |

or predicate. In the case of elements with more than one source or sink ends, we use indices.

The intuition behind these constraints is that their solutions reflect the semantic model of each element as given by CASM and CC.

**Example 6.2.2** *Figure 6.2.1 shows a Reo network that consists of a* transformer *channel with the function* $3 * \hat{a}$*, whose domain is the set of numbers* $Number$ *and a* filter *channel with the condition* $\hat{b}\%2 = 0$ *and domain* $Number$*.*

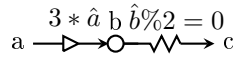



**Figure 6.2.1**: A data-aware Reo connector

Since none of the Reo primitives in Figure 6.2.1 is context-dependent, we use the constraints corresponding to the primitives in this network as defined in Table 6.2.1.

$$\psi_{Transformer}(a, b, 3 * \hat{a}) = \tilde{a} \Leftrightarrow \tilde{b} \wedge \tilde{a} \Rightarrow (\hat{a} \in Number \wedge \hat{b} = 3 * \hat{a})) \tag{6.1}$$

$$\psi_{Filter}(b, c, \hat{b}\%2 = 0) = \tilde{c} \Rightarrow (\tilde{b} \wedge \hat{b} \in Number \wedge (\hat{b}\%2 = 0)) \tag{6.2}$$

Equation 6.1 states that flow occurs on the source end of the *transformer* channel iff it occurs on its sink end. In addition, flow can exist only if the data item that enters the source end of the channel is a number. In this case, the data item written on the sink end is three times the value of the source data item.

Equation 6.2 expresses that flow on the source end of the *filter* channel leads to flow on its sink end, iff the data item belongs to the channel's accepting pattern (which is $\hat{b}\%2 = 0$).

In this case, the value of data items passing through the ends are equal. No flow through the sink end $c$ is either due to no flow on $b$ or that the incoming data item does not satisfy the accepting pattern. As mentioned, the conjunction of these constraints (subject to Axiom 7.2.1, which trivially holds in this case) encodes the behavior of the given Reo network.

### 6.2.2 Solving RCSPs

In this section, we show how to obtain the solutions of RCSPs. Since Reo Constraint Satisfaction Problems (RCSPs) have predicates with free variables of types Boolean

($\{\top, \bot\}$) and data ($\mathcal{D}$), a SAT-solver or a numeric constraint solver cannot solve them alone. Satisfiability Modulo Theories (SMT) [BSST09] solvers find solutions for propositional satisfiability problems where propositions are either Boolean or constraints in a specific theory.

However, SMT-solvers are not applicable in our case either, because unlike SAT-solvers they find only an instance of a solution as opposed to the complete set of answers. Another drawback of most SAT- and SMT-solvers is that they work only on quantifier-free formulae, while we use existential quantifies to implement the *hiding* operator of constraint automata (see Section 6.3).

To generate the CASM corresponding to a given Reo network, we need all solutions and thus resort to a hybrid approach that uses both SAT-solvers and Computer Algebra Systems (CASs), namely, REDUCE [Ray87], which is a system for general algebraic computations.

First, we form a pure Boolean constraint system by substituting data dependent constraints with new Boolean variables and find all solutions for the new constraints using a SAT-solver. Then, by substituting each such solution into the original constraints, we obtain a data dependent constraint satisfaction problem that a CAS can solve symbolically. From these solutions, we extract a CASM corresponding to the Reo network encoded by the original set of constraints. Our approach avoids state explosion by treating data constraints symbolically. In the following, we elaborate on our approach.

In an RCSP $\langle \mathcal{P}, \mathcal{M}, M_0, \mathcal{V}, C \rangle$, let $\mathcal{V}_B$ and $\mathcal{V}_D$ be the sets of free Boolean and free data variables of $C$, respectively, where $\mathcal{V} = \mathcal{V}_B \cup \mathcal{V}_D$, and let $A_D$ be the set of atomic predicates of $C$ containing data variables. The following is our procedure for solving $C$.

1. We obtain $C_B$ from $C$ by replacing every occurrence of $x \in A_D$ with a unique new Boolean variable $y \notin \mathcal{V}$. For example, for $C = (\tilde{c} \Rightarrow \tilde{b}) \wedge (\tilde{c} \Rightarrow (\hat{b} \in Number \Rightarrow \hat{b}\%2 = 0))$ in Figure 6.2.1, we obtain $C_B$ as $(\tilde{c} \Rightarrow \tilde{b}) \wedge (\tilde{c} \Rightarrow (y_1 \Rightarrow y_2))$ where $y_1$ and $y_2$ replace $\hat{b} \in Number$ and $\hat{b}\%2 = 0$, respectively.

2. An off-the-shelf SAT-solver can find the set of solutions $S_B$ for $C_B$. We define the finite set of constraints $C[S_B] = \{C[v_1, v_2, \ldots, v_n \setminus z_1, z_2, \ldots z_n] \mid \text{for all distinct } v_i \in \mathcal{V}_B, 1 \leq i \leq n = |\mathcal{V}_B|, z_i \in S(v_i), S \in S_B\}$.

3. Every $C_D \in C[S_B]$ is a numerical constraint satisfaction problem, which we (symbolically) solve using a Computer Algebra System. Every solution to each $C_D$ along with the SAT solution $S \in \mathcal{S}_B$ that produced $C_D \in C[S_B]$ in the previous step, constitute a solution to the RCSP.

Using the presented technique, we obtain the solutions for the RCSP corresponding to Examples 6.2.2 as follows:

1. $\langle\{\tilde{a} = \bot, \tilde{b} = \bot, \tilde{c} = \bot\}, \top\rangle$,
2. $\langle\{\tilde{a} = \top, \tilde{b} = \bot, \tilde{c} = \bot\}, \hat{a} \notin Number\rangle$,
3. $\langle\{\tilde{a} = \top, \tilde{b} = \top, \tilde{c} = \bot\}, \hat{a} \in Number \wedge \hat{b} = 3 * \hat{a} \ \wedge \ \hat{b}\%2 \neq 0\rangle$,
4. $\langle\{\tilde{a} = \top, \tilde{b} = \top, \tilde{c} = \top\}, \hat{a} \in Number \wedge \ \hat{b} = 3 * \hat{a} \ \wedge \ \hat{b}\%2 = 0 \ \wedge \ \hat{b} = \hat{c}\rangle$.

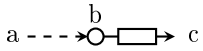



**Figure 6.2.2**: A context-dependent Reo connector

**Example 6.2.3** *Figure 6.2.2 depicts a Reo network that consists of a* lossySync *channel and a* $\text{FIFO}_1$ *channel connecting on the node b.*

Since the Reo network in Figure 6.2.1 contains a *lossySync* that is a context dependent channel, we use the context-aware RCSP encoding from Table 6.2.2:

$$\psi_{LossySync}(a, b) = \tilde{b} \Rightarrow (\tilde{a} \wedge (\hat{a} = \hat{b})) \wedge \neg a^{c\triangleright} \wedge \neg\tilde{a} \Rightarrow b^{k\triangleright}. \tag{6.3}$$

$$\psi_{FIFO_1}(b, c, m) = \tilde{b} \Rightarrow (\neg\mathring{m} \wedge \mathring{m}' \wedge (\hat{m}' = \hat{b})) \wedge \tilde{c} \Rightarrow (\mathring{m} \wedge \neg\mathring{m}' \wedge (\hat{m} = \hat{c})) \wedge (\neg\tilde{b} \wedge \neg\tilde{c}) \Rightarrow ((\mathring{m} \Leftrightarrow \mathring{m}') \wedge \mathring{m} \Rightarrow (\hat{m} = \hat{m}')) \wedge \neg\mathring{m} \Rightarrow c^{c\triangleright} \wedge \mathring{m} \Rightarrow b^{k\triangleright}. \tag{6.4}$$

Equation 6.3 states that flow on the sink end of the *lossySync* is due to flow on its source end. If there is flow on the sink end of the *lossySync*, the data items exchanged at the source and the sink ends are the same. However, it is possible that the source end has flow, but the sink end does not. In this case, the reason for no flow comes from the environment with which the sink end communicates. The third possible behavior of the channel is that there is no flow on the source end due to the environment, in which case the channel provides a reason for no flow on its sink end.

Equation 6.4 expresses the behavior of the $FIFO_1$ channel as follows: The flow on the source end of the channel states that the value of the variable representing the state memory (of the current state) is undefined. The flow on the source end defines the state memory variable for the next state to contain the value of the

incoming data item. On the other hand, flow on the sink end means that the value of the state memory variable is defined. The data item leaving the sink end is equivalent to the buffer's data item. In addition, the value of the state memory variable becomes undefined in the next state. If there is no flow on the ends, the variables related to the states stay the same. Being empty, the $FIFO_1$ channel provides a reason for no flow on its sink end, while being full does so on the source end of the channel.

The solutions for the RCSP 6.4, (where for brevity, we omit the values of the variables representing the context, such as $b^{c\triangleright}$) are as follows:

1. $\langle\{\tilde{a} = \bot, \tilde{b} = \bot, \tilde{c} = \bot, \mathring{m} = \bot, \mathring{m}' = \bot\}, \top\rangle$,
2. $\langle\{\tilde{a} = \top, \tilde{b} = \top, \tilde{c} = \bot, \mathring{m} = \bot, \mathring{m}' = \top\}, \hat{a} = \hat{b} \wedge \hat{m}' = \hat{b}\rangle$,
3. $\langle\{\tilde{a} = \top, \tilde{b} = \bot, \tilde{c} = \bot, \mathring{m} = \top, \mathring{m}' = \top\}, \hat{m} = \hat{m}'\rangle$,
4. $\langle\{\tilde{a} = \bot, \tilde{b} = \bot, \tilde{c} = \bot, \mathring{m} = \top, \mathring{m}' = \top\}, \hat{m} = \hat{m}'\rangle$,
5. $\langle\{\tilde{a} = \top, \tilde{b} = \bot, \tilde{c} = \bot, \mathring{m} = \top, \mathring{m}' = \bot\}, \hat{m} = \hat{c}\rangle$,
6. $\langle\{\tilde{a} = \bot, \tilde{b} = \bot, \tilde{c} = \top, \mathring{m} = \top, \mathring{m}' = \bot\}, \hat{m} = \hat{c}\rangle$.

### 6.2.3 Constructing CASM

In order to construct the CASM from the set of solutions $S$ for an RCSP $\langle\mathcal{P}, \mathcal{M}, M_0, \mathcal{V}, C\rangle$, we first define

- $\mathcal{N} = \{n \mid n^c \in \mathcal{P} \vee n^k \in \mathcal{P}\}$

and then map each solution $\langle s, s_d\rangle \in S$ into a transition $q \xrightarrow{N,g} p$ as follows:

- $q = \langle\{m \mid m \in \mathcal{M}, s\,(\mathring{m}) = \top\}\rangle$,
- $p = \langle\{m \mid m \in \mathcal{M}, s\,(\mathring{m}') = \top\}\rangle$,
- $N = \{n \mid n \in \mathcal{N}, s\,(\tilde{n}) = \top\}$,
- The data constraint $g$ is (a syntactic variant of) $s_d$.

We obtain the CASM $A = (Q, \mathcal{N}, \rightarrow, q_0, \mathcal{M})$ from the set $\longrightarrow$ of all transitions generated above, where:

- $Q = \{q \mid q \xrightarrow{N,g} p \vee p \xrightarrow{N,g} q\}$,
- $q_0 = \langle\{m \mid m \in M_0, s(\mathring{m}) = \top\}\rangle$,

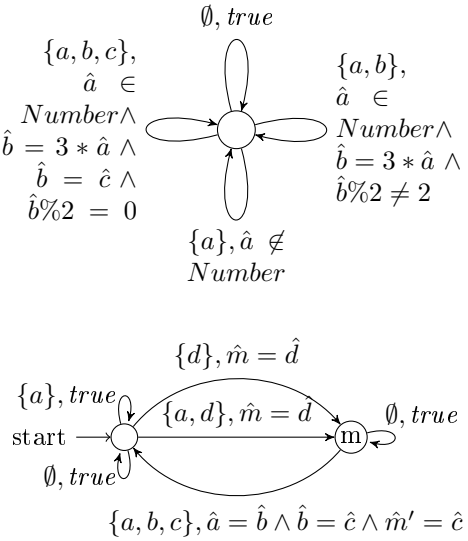



**Figure 6.2.3:** CASMs generated for Figures 6.2.1 and 6.2.2

- $\mathcal{M}$ is the same $\mathcal{M}$ as in the RCSP.

Applying the above procedure to the solutions of RCSPs constraints generates their corresponding CASMs. For instance, the first solution for the constraints in Example 6.2.2 generates the transition $q \xrightarrow{\emptyset, true} q$, where $q$ is the only state of the CASM, which has no state memory variable. This is so because the set of variables of the form $\mathring{m}$ is empty. Also, the transition has no synchronizing port, because the value of every one of the variables $\tilde{a}, \tilde{b}$ and $\tilde{c}$ is $\perp$. Figures 6.2.3a and 6.2.3b show the CASMs derived from the RCSPs in Examples 6.2.1 and 6.2.2.

Our approach deals with data in a symbolic fashion, where we partition the global set of data values to equivalence classes toward which a Reo network behaves differently. This is in contrast with the traditional way of dealing with data in the formal semantics of Reo (and other models), where they consider a different state for each possible value that can be stored in buffers and a distinct transition for each data value passing through the ports.

Our symbolic approach allows working with an infinite data domain. In addition, rather than implementing the highly time- and memory-demanding custom-made algorithms to generate Reo formal semantics, we use the efficient SAT-solvers and computer algebra systems to solve constraints whose solutions are equivalent to these models.

An experimental study done on the efficiency of using SAT-solvers to generate

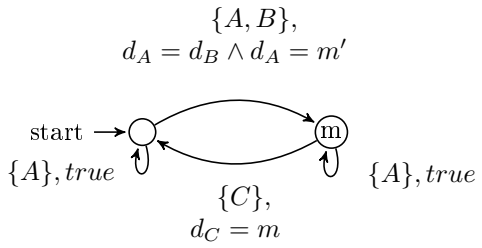



**Figure 6.2.4**: CASM for Figure 6.2.2

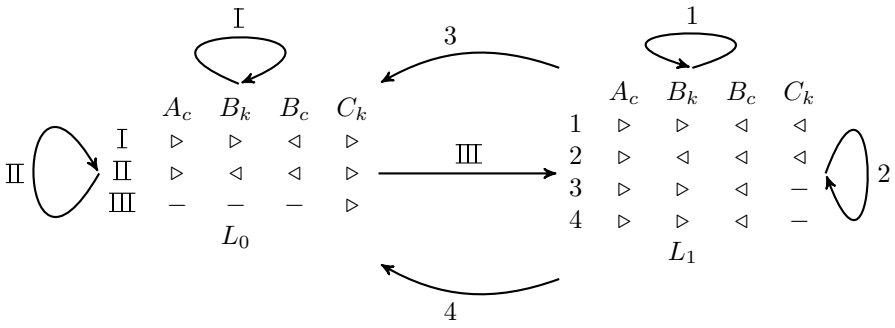



**Figure 6.2.5**: CC for Figure 6.2.2

Reo formal semantics [Pro11] compares two prototypes based on constraint satisfaction techniques and connector coloring semantics, without taking data constraints in consideration. The results illustrate that the approach based on constraint solving scales better and is more efficient. In chapter 7 we present an evaluation through a case study, which affirms this conclusion.

## 6.3 Hiding

We use *hiding* to abstract from internal transitions. This is a mechanism to support hierarchy and is used to create components.

The author in [Pro11] proposes applying the existential quantifier to the constraints encoding of the behavior of a network to abstract from internal ports and their corresponding data variables. Similarly, we use existential quantifiers such as $\exists \tilde{e}, \hat{e}, e^{\triangleright} : C$, where $C$ is the RCSP of a Reo network and $e$ is an internal node to hide.

Although several algorithms exist for the problem of quantifier elimination in Boolean algebra and first order logic [BZ07] [Abd02] [Dav88], we are not aware of

any working tool that does quantifier elimination on Boolean algebraic formulae. Therefore, our tool implements the *hiding* operator as defined for CASM.

Hiding the internal nodes on some transitions can make the set of their synchronized nodes empty. Here, we refer to such a transition as an *empty* transition, if the free variables of its guard are merely state memory variables. Under some circumstances, we can merge the source and the target states of empty transitions. Let $q$ and $p$ be two states in a CASM such that $q \xrightarrow{\emptyset,g} p$. The following are the conditions under which the state $p$ can merge into the sate $q$:

1. The states $q$ and $p$ have the same number of state memory variables.

2. The guard $g$ consists of the conjunction of the predicates of the form of $x = y'$, for $x, y \in \mathcal{M}$. This way, $g$ defines a correspondence relation between the state memory variables of the state $q$ and those of the state $p$.

3. For each transition $q \xrightarrow{N,g'} r$ where $r \notin \{p, q\}$, there is a transition $p \xrightarrow{N,g''} r$ such that $g' \Leftrightarrow g''_g$, where $g''_g$ is obtained from $g$ by replacing all occurrences of the next state memory variable $y'$ with the next state memory variable $x'$, if $g$ contains $x = y'$ for state memory variables $x, y \in \mathcal{M}$.

4. For each transition $r \xrightarrow{N,g'} p$ where $r \notin \{p, q\}$, there is a transition $r \xrightarrow{N,g''} q$ such that $g'' \Leftrightarrow g'_g$, where $g'_g$ is derived from $g$ by substituting all occurrences of the state memory variable $x$ in $g$ with the state memory variable $x$, if $g$ contains $x = y'$ for state memory variables $x, y \in \mathcal{M}$.

Provided that the above conditions hold, the state $p$ merges into the state $q$ as follows:

1. We eliminate the transition $q \xrightarrow{\emptyset,g} p$.

2. We remove the state $p$ after substituting $y$, $y'$, and $p$ with $x$, $x'$, and $q$ in all transitions. Observe that such substitutions convert the non-eliminated transitions between the states $q$ and $p$ into loops over the state $q$.

**Example 6.3.1** *Figure 6.3.1 shows a* $\text{FIFO}_2$ *derived from composing two* $\text{FIFO}_1$*s. The CASM corresponding to the* $\text{FIFO}_2$ *is in Figure 6.3.2a. Figure 6.3.2b depicts the CASM resulting from hiding the mixed node b. Figure 6.3.2c presents the result of eliminating the empty transitions.*

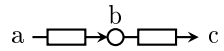
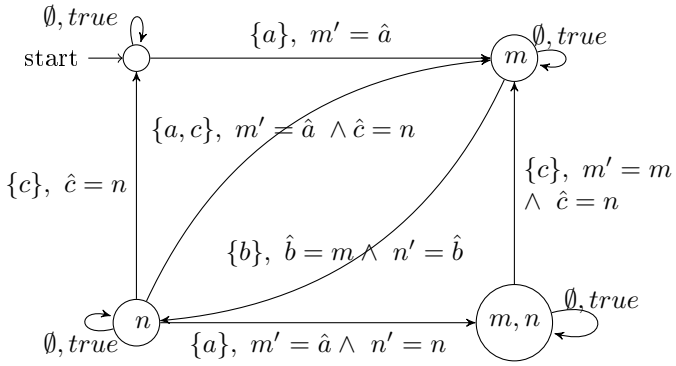
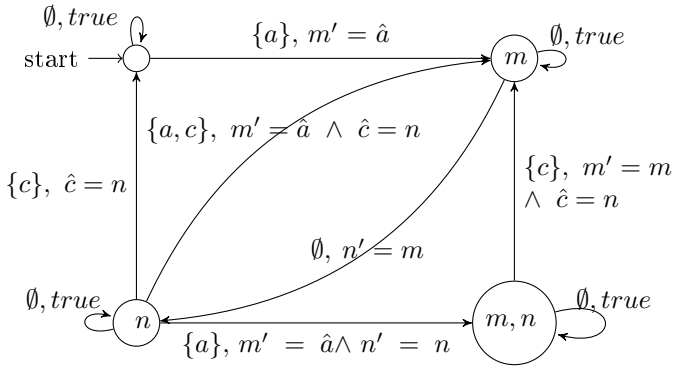


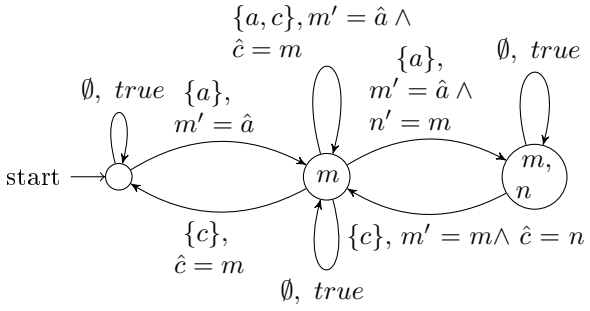

**Figure 6.3.1**: Two $FIFO_1$s forming $FIFO_2$




**(a)** CASM of Figure 6.3.1




**(b)** Hiding internal ports




**(c)** Merging the states

**Figure 6.3.2**: Hiding the empty transition and merging its source and target states for the CASM of $FIFO_2$ in Figure 6.3.1

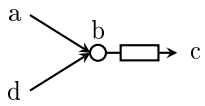



**Figure 6.4.1**: A sample Reo network

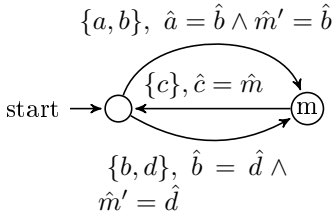



**Figure 6.4.2**: CASM corresponding to Figure 6.4.1

## 6.4 Correctness and compositionality

CASM and CC model the presence and the absence of data flow on a Reo network at different levels of granularity. For instance, Figure 6.4.2 and Figure 6.4.3 are the CASM and CC semantics for the Reo network in Figure 6.4.1. As the figures show, the node $b$ in CASM is mapped to three primitive ends in CC, which do not necessarily have the same coloring.

In this section, we formally investigate the relation between the solutions of the RCSP for a given Reo network and its CC and CASM semantics. However, we first need to present some definitions.

For a given network $R$ with $A = (Q, \mathcal{N}, \rightarrow, q_0, \mathcal{M})$, its CASM and $C = \langle \mathcal{P}, \mathcal{L}, L_0, \eta \rangle$, its CC, we define the function $O_R : \mathcal{P} \rightarrow \mathcal{N}$ as it maps each CC port to its corresponding node in CASM.

**Definition 6.4.1 (Correlation $\sim$)** *Let $A = (Q, \mathcal{N}, \rightarrow, q_0, \mathcal{M})$ be a CASM and $C = \langle \mathcal{P}, \mathcal{L}, L_0, \eta \rangle$ be a CC. We define the relation $\sim$: $Q \times \mathcal{L}$, as follows:*

- *$q_0 \sim L_0$, if $\mathcal{N} = \bigcup_{p \in \mathcal{P}} O_R(p)$.*
- *For each $p \in Q$ and $L' \in \mathcal{L}$, $p \sim L'$ if the following conditions hold:*
  1. *There exists $q \in Q$ and $L \in \mathcal{L}$ such that $q \xrightarrow{N,g} p$ and $L' = \eta(L, l)$, where $l \subset L$,*
  2. *For all $n \in N, n = O_R(p) \Leftrightarrow l(e) = -$,*
  3. *$q \sim \mathcal{L}$.*

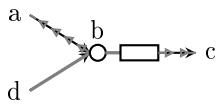



**Figure 6.4.3**: A coloring annotated state of the CC corresponding to Figure 6.4.1

*If a relation $\sim$ exists between $Q$ and $\mathcal{L}$, then we say that $A$ correlates to $C$, written as $A \sim C$.*

It is easy to see that if $A$ and $C$ belong to the same Reo network, then $q_0 \sim L_0$. Therefore, $A \sim C$.

**Definition 6.4.2 ($id$ mapping)** *For the CASM $A = (Q, \mathcal{N}, \rightarrow, q_0, \mathcal{M})$ and the coloring semantics $C = \langle \mathcal{P}, \mathcal{L}, L_0, \eta \rangle$ such that $A \sim C$, the function $id : \mathcal{L} \rightarrow 2^Q$ correlates coloring tables with subsets of $Q$ such that $id(L)$ returns the set of all $q \in Q$ wherein the data-flow possibilities resulting from the outgoing transitions of $q$ correspond to the data-flow possibilities prescribed by the coloring table $L$.*

The following example illustrates Definition 6.4.2.

**Example 6.4.1** *Figure 6.2.4 and Figure 6.2.5 are, respectively, the CASM and the CC of the Reo network of Figure 6.2.2.*

*Note that we have modified the presentation of the CC to resemble the CASM structure. For instance, the transition $L_1 \xrightarrow{i} L_2$ represents $L_2 = \eta$ ($L_1$, $cols_{L_1}$ $[i]$), where the $cols_{L_1}$ is the possible colorings for each coloring table as shown in the example.*
*Let $q$ designate the state without a state memory variable in the CASM of Figure 6.2.5, and let $p$ designate the state with the state memory variable $m$. Then, according to Definition 6.4.1, $q \sim L_0$ and $p \sim L_1$.*

**Definition 6.4.3 (Memory cells of a state)** *We use $\mathcal{M}_q$ to denote the set of all $m \in \mathcal{M}$ that syntactically appear as $m$ in a data constraint $g$ on an outgoing transition $q \xrightarrow{N,g} p$ of the state $q$. Analogously, we use $\mathcal{M}'_q$ to denote the set of all $m \in \mathcal{M}$ that syntactically appear as $m'$ in a data constraint $g$ on an incoming transition $p \xrightarrow{N,g} q$ of the state $q$. We call $\mathcal{M}_q$ and $\mathcal{M}'_q$, respectively, the* accessed *and the* updated *memory cells of the state $q$.*

**Definition 6.4.4 (Encoding a Reo network)** *For the semantics for a Reo network $R$ as $A = (Q, \mathcal{N}, \rightarrow, q_0, \mathcal{M})$ and $C = \langle \mathcal{P}, \mathcal{L}, L_0, \eta \rangle$, the RSCP $\Psi = \langle \mathcal{P}, \mathcal{M}, M_0, \mathcal{V}, \mathcal{C} \rangle$ encodes $R$ in terms of its CASM and CC semantics iff the following conditions hold:*

1. *For all solution pairs* $\langle s, s_d \rangle \vDash \Psi$, *there exist a transition* $q \xrightarrow{N,g} p$ *and a colorings* $l \in L \in \mathcal{L}$ *such that*

   (a) *for all* $m \in \mathcal{M}$, $m \in \mathcal{M}_q$ *iff* $s(\mathring{m}) = \top$

   (b) *for all* $m \in \mathcal{M}$, $m \in \mathcal{M}_p$ *iff* $s(\mathring{m}') = \top$

   (c) *for all* $n \in \mathcal{N}$, $n \in N$ *iff* $s(\tilde{n}) = \top$

   (d) *for all* $\hat{v} \in \mathcal{V}$, $[g]\,\hat{v} \backslash s_d(\hat{v})$

   (e) *for all* $p \in \mathcal{P}$, $s(\tilde{p}) = \top$ *iff* $l(n) = -$ *coloring*

   (f) *for all* $p \in \mathcal{P}$, $s(e^{\triangleright}) = \top$ *where* $e$ *is either* $p^c$ *or* $p^k$ *iff* $l(n) = \triangleleft$

   (g) *for all* $p \in \mathcal{P}$, $s(e^{\triangleright}) = \bot$ *where* $e$ *is either* $p^c$ *or* $p^k$ *iff* $l(n) = \triangleright$

   (h) *for all* $p \in \mathcal{P}$ *such that* $p^c \cup p^k \subset \mathcal{P}$, *if* $sol(\tilde{p}) = \bot$, *then* $p^{c\triangleright} \vee p^{k\triangleright}$.

2. *For all transitions* $q \xrightarrow{N,g} p$, *and colorings* $l \in L \in \mathcal{L}$ *such that* $q \sim L$ *and* $p \sim \eta(L, l)$, *there exists a solution* $\langle s, s_d \rangle$ *such that*

   (a) *for all* $\mathring{m} \in \mathcal{V}$, $s(\mathring{m}) = \top$ *iff* $q \in id(L)$ *and* $m \in \mathcal{M}_q$

   (b) *for all* $\mathring{m}' \in \mathcal{V}$, $s(\mathring{m}') = \top$ *iff* $p \in id(\eta(L, l))$ *and* $m \in \mathcal{M}'_p$

   (c) *for all* $\tilde{n} \in \mathcal{V}$ *iff* $n \in N$ *and* $l(n) = -$

   (d) *for all* $\hat{v} \in \mathcal{V}$, $g\,[v] \setminus s_d\,(\hat{v})$

   (e) *for all* $e^{\triangleright} \in \mathcal{V}$, *where* $e$ *is either* $n^c$ *or* $n^k$, $s(e^{\triangleright}) = \top$ *iff* $n \notin N$ *and* $l(n) = \triangleleft$

   (f) *for all* $e^{\triangleright} \in \mathcal{V}$, *where* $e$ *is either* $n^c$ *or* $n^k$, $s(e^{\triangleright}) = \bot$ *iff* $n \notin N$ *and* $l(n) = \triangleright$

The purpose of this encoding is to obtain the behavior of the Reo network as specified in both its CASM and CC semantics by solving the RCSP $\psi$.

**Theorem 6.4.1 (Correctness)** *For the CASM* $A = (Q, \mathcal{N}, \rightarrow, q_0, \mathcal{M})$ *and the CC* $C = \langle \mathcal{P}, \mathcal{L}, L_0, \eta \rangle$ *such that* $A \sim C$, *let* $\Psi$ *be the RCSP encoding* $A$ *and* $C$. *The CASM* $A' = (Q', \mathcal{N}', \rightarrow', {q'}_0, \mathcal{M}')$ *and the CC* $C' = \langle \mathcal{P}', \mathcal{L}', {L'}_0, \eta' \rangle$ *extracted from the solutions of* $\Psi$ *are refinements of* $A$ *and* $C$ *and* $A' \sim C'$.

**Proof** For all solution $s \vDash \Psi$, there is a coloring $l'$ and a transition $q' \xrightarrow{N',g'} p'$ such that the first part of Definition 6.4.4 holds:

- $l' \in L$,
- $q' \xrightarrow{N',g'} p'$,

- $q' \in Q$,
- $p' \in Q$.

We construct $A' = (\bigcup (q' \cup p') , \mathcal{N}', \rightarrow', q'_0, \mathcal{M}')$ and $C' = \langle \mathcal{P}', \mathcal{L}', L'_0, \eta' \rangle$ from the solutions, where $A' \sqsubseteq A$ and $C' \sqsubseteq C$.

**Lemma 6.4.1** *Assume the condition (1) of Definition 6.4.4 holds for two RC-SPs* $\Psi_1 = \langle \mathcal{P}_1, \mathcal{M}_1, \mathcal{M}_{0,1}, \mathcal{V}_1, \mathcal{C}_1 \rangle$ *and* $\Psi_2 = \langle \mathcal{P}_2, \mathcal{M}_2, \mathcal{M}_{0,2}, \mathcal{V}_2, \mathcal{C}_2 \rangle$ *for automata* $A_1 = (Q_1, \mathcal{N}_1, \rightarrow_1, q_{0_1}, \mathcal{M}_1)$ *and* $A_2 = (Q_2, \mathcal{N}_2, \rightarrow_2, q_{0_2}, \mathcal{M}_2)$ *and colorings* $C_1 = \langle \mathcal{P}_1, \mathcal{L}_1, L_{0_1}, \eta_1 \rangle$ *and* $C_2 = \langle \mathcal{P}_2, \mathcal{L}_2, L_{0_2}, \eta_2 \rangle$. *Then the condition (1) of Definition 6.4.4 holds for* $Psi_1 \odot Psi_2$, $A_1 \bowtie A_2$ *and* $C_1 \bullet C_2$.

**Proof** Assume $\langle s, s_d \rangle \vDash \Psi_1$ and $\langle s, s_d \rangle \vDash \Psi_2$. Let $\langle s_1, s_{d_1} \rangle$ and $\langle s_2, s_{d_2} \rangle$ be the images of $\langle s, s_d \rangle$ over $\mathcal{V}_1$ and $\mathcal{V}_2$, respectively. Then $\langle s_1, s_{d_1} \rangle \vDash \Psi_1$ and $\langle s_2, s_{d_2} \rangle \vDash \Psi_2$ and for each $v \in \mathcal{V}_1 \cap \mathcal{V}_2$, $s_1(v) = s_2(v)$ and $s_{d_1}(v) = s_{d_2}(v)$.
Therefore, there exist transitions $q_1 \xrightarrow{N_1, g_1}_1 p_1$ and $q_1 \xrightarrow{N_2, g_2}_2 p_2$ and colorings $l_1 \in L_1 \in \mathcal{L}_1$ and $l_2 \in L_2 \in \mathcal{L}_2$ such that the condition *(1)* of Definition 6.4.4 holds.
For each $\tilde{v} \in \mathcal{V}_1 \cap \mathcal{V}_2$, $s_1(\tilde{v}) = \top$ iff $v \in N_1$ and $v \in N_2$. Therefore, $N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1$, which means $\langle q_1, q_2 \rangle \xrightarrow{N_1 \cup N_2, g_1 \wedge q_2} \langle p_1, p_2 \rangle$.
For each $\tilde{v} \in \mathcal{V}_1 \cap \mathcal{V}_2$, $s_1(\tilde{v}) = \top$ iff $l_1(O_R(n)) = -$ and $l_2(O_R(n)) = -$, $s_1(\tilde{v}) = \bot \wedge s_1(v^{\triangleright}) = \top$ iff $l_1(O_R(n)) = \triangleright$ and $l_2(O_R(n)) = \triangleright$, and $s_1(\tilde{v}) = \bot \wedge s_1(v^{\triangleright}) = \bot$ iff $l_1(O_R(n)) = \triangleright$ and $l_2(O_R(n)) = \triangleleft$.
On the other hand,

- for all $m \in \mathcal{M}$, $m \in \mathcal{M}_q$ iff $s(\mathring{m}) = \top$
- for all $m \in \mathcal{M}$, $m \in \mathcal{M}_p$ iff $s(\mathring{m}') = \top$
- for all $n \in \mathcal{N}$, $n \in N$ iff $s(\tilde{n}) = \top$
- for all $\hat{v} \in \mathcal{V}$, $[g]\, \hat{v} \backslash s_d(\hat{v})$
- for all $p \in \mathcal{P}$, $s(\tilde{p}) = \top$ iff $l(n) = -$
- for all $p \in \mathcal{P}$, $s(e^{\triangleright}) = \top$ where $e$ is either $p^c$ or $p^k$ iff $l(n) = \triangleleft$
- for all $p \in \mathcal{P}$, $s(e^{\triangleright}) = \bot$ where $e$ is either $p^c$ or $p^k$ iff $l(n) = \triangleright$

Therefore, the condition *(1)* of Definition 6.4.4 holds for $\Psi_1 \odot \Psi_2$, $A_1 \bowtie A_2$ and $C_1 \bullet C_2$.

**Lemma 6.4.2** *Assume the condition (2) of Definition 6.4.4 holds for two RCSPs* $\Psi_1 = \langle \mathcal{P}_1, \mathcal{M}_1, \mathcal{M}_{0,1}, \mathcal{V}_1, \mathcal{C}_1 \rangle$ *and* $\Psi_2 = \langle \mathcal{P}_2, \mathcal{M}_2, \mathcal{M}_{0,2}, \mathcal{V}_2, \mathcal{C}_2 \rangle$ *and for CASMs* $A_1$ *and* $A_2$ *and CCs* $C_1$ *and* $C_2$*. Then the condition (2) of Definition 6.4.4 holds for* $\Psi_1 \odot \Psi_2$*,* $A_1 \bowtie A_2$ *and* $C_1 \bullet C_2$*.*

**Proof** Consider the solutions $\langle s_1, s_{d_1} \rangle \vDash \Psi_1$ and $\langle s_2, s_{d_2} \rangle \vDash \Psi_2$ such that $\langle s_1, s_{d_1} \rangle$ encodes $q_1 \xrightarrow{N_1, g_1}_1 p_1$ and $l_1 \in C_1$ and $\langle s_2, s_{d_2} \rangle$ encodes $q_2 \xrightarrow{N_2, g_2}_2 p_2$ and $l_2 \in C_2$. Then, $\langle s, s_d \rangle \vDash \Psi_1 \odot \Psi_2$, where $\langle s, s_d \rangle = \langle s_1 \cup s_2, s_{d_1} \cup s_{d_2} \rangle$. Here, we distinguish between two cases:

- For all $v \in dom(s_1) \cap dom(s_2)$ and for all $\hat{v} \in dom(s_{d_1}) \cap dom(s_{d_2})$, $s_1(v) = s_2(v)$ and $s_{d_1}(\hat{v}) = s_{d_2}(\hat{v})$.

- Otherwise.

The former case describes valid solutions. For two transitions $q_1 \xrightarrow{N_1, g_1}_1 p_1$ and $q_2 \xrightarrow{N_2, g_2}_2 p_2$, we have $\langle q_1, q_2 \rangle \xrightarrow{N_1 \cup N_2, g_1 \wedge g_2} \langle p_1, p_2 \rangle$ iff $N_1 \cup \mathcal{N}_2 = N_2 \cup \mathcal{N}_1$.
For two colorings $l_1$ and $l_2$, the coloring $l = l_1 \odot l_2$ is valid iff either $e^c \in dom(l_1)$ and $e^k \in dom(l_2)$ and $\neg(l_1(e^c) = \lhd \wedge l_2(e^k) = \lhd)$ or $e^k \in dom(l_1)$ and $e^c \in dom(l_2)$, $\neg(l_1(e^k) = \lhd \wedge l_2(e^c) = \lhd)$.
For all $n \in N_1$ and $n \in N_2$, $s_1(n) = \top$, $s_2(n) = \top$, $n \in N_1 \cap N_2$ and $N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1$ means that $\{n | \tilde{n} \in \mathcal{P}_1 \wedge s_1(\tilde{n}) = \top \wedge \tilde{n} \in \mathcal{P}_2\} = \{n | \tilde{n} \in \mathcal{P}_2 \wedge s_2(\tilde{n}) = \top \wedge \tilde{n} \in \mathcal{P}_1\}$. So, $\{n | \tilde{n} \in \mathcal{P}_1 \cup \mathcal{P}_2 \wedge s_1(\tilde{n}) = \top\} = \{n | \tilde{n} \in \mathcal{P}_1 \cap \mathcal{P}_2 \wedge s_2(\tilde{n}) = \top\}$. This means that for all $\tilde{n} \in \mathcal{P}_1 \cap \mathcal{P}_2$, $s_1(\tilde{n}) = s_2(\tilde{n})$.
For all $q_1 \in Q_1$, $m \in M(q_1)$ iff $s_1(m) = \top$ and $q_2 \in Q_2$, $m \in M(q_2)$ iff $s_2(m) = \top$. Since $\mathcal{M}_1 \cap \mathcal{M}_2 = \emptyset$, $M(\langle q_1, q_2 \rangle) = M(q_1) \cup M(q_2)$, $m \in M(\langle q_1, q_2 \rangle)$ iff $s_1(m) = \top \vee s_2(m) = \top$.
If $s_{d_1} \Rightarrow g_1$ and $s_{d_2} \Rightarrow g_2$, then $s_{d_1} \cup s_{d_2} \Rightarrow g_1 \wedge g_2$.
The latter gives invalid solutions, which are impossible. Therefore, the condition (*2*) of Definition 6.4.4 holds for $\Psi_1 \odot \Psi_2$, $A_1 \bowtie A_2$ and $C_1 \bullet C_2$.

**Theorem 6.4.2 (Compositionality)** *If* $\Psi_1$ *encodes the automaton* $A_1$ *and the CC* $C_1$ *and* $\Psi_2$ *encodes the automaton* $A_2$ *and the CC* $C_2$*, then* $\Psi_1 \odot \Psi_2$ *encodes the automaton* $A_1 \bowtie A_2$ *and the CC* $C_1 \bullet C_2$*.*

**Proof** It follows directly from Lemmas 6.4.1 and 6.4.2.

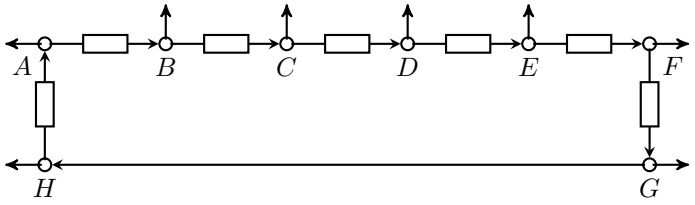



**Figure 6.4.4**: 7-Sequencer

### 6.4.1 Performance evaluation

In the remainder of this section, we perform an evaluation on the performance of the presented constraint-based approach along with a brief comparison with the existing approaches, namely, connector coloring and constraint automata.

The execution time of the algorithm depends on the number of states of the given RLTS and the time required to solve the constraints encoding of the network. Thus, to study the performance of our framework and to compare it with the existing approaches in computing operational semantics of Reo networks, we choose the case of *N-Sequencer*, which consists of $N$ *FIFO* channels that are circularly connected.

In this example, adding each $FIFO_1$ channel doubles the number of the states in the corresponding semantics model and increases the complexity of the constraints encoding the behavior of the network by adding new variables and new assertions on them.

This makes the network a good choice for our benchmarking, where we would like to compare the solutions on state explosion.

Since we are interested in comparing our approach with the existing tools, we do not include priority in our case study. This is justified by the fact that incorporating priority does not affect the number of states in the model and only will influence the size of the constraint. In addition, adding more $FIFO_1$ channels to the network increases both the number of the states and the size of the constraint capturing the semantics of the network. Since we are using optimized third-library tools to solve the constraints, we do not distinguish between the various form of constraints obtained from different channels and instead we are just interested in the approximate growth of the constraints.

Figure 6.4.4 shows a *7-sequencer*. Though the size of the operational semantics model of this network grows in a linear fashion in relation with $N$, the number of intermediate states to compute the final results grows exponentially.

The benchmarks have been performed on Mac Book Pro OS X El Capitan with

2.8 GHz Intel Core i7 and 16 GB MHz DDR3 memory. The implementation of our approach is in Java 8. We have used Reduce Algebra System[Ray87] to compute the conjunctive normal form of the constraints and to solve them. We have also experimented with an optimization on the number of the variables used in the constraints by substituting equal variables with a single variable. The result of the original and the optimized approaches are presented with red and blue square markers, respectively.

Figure 6.4.5a presents the average time required for computing a single solution of the RCSP of a *N-Sequencer*. Figure 6.4.5b demonstrates the relation between $N$ and the size of the RCSP's constraints of a *N-Sequencer*. This is an indication of complexity of the constraint that needs to be solved. Note that the number of solutions for RCSP of a *N-Sequencer* is *2N*, which equals to the number of transitions in the corresponding RLTS. Finally, Figure 6.4.5c illustrates the total time required to compute all solutions of a RCSP's constraint of a *N-Sequencer*. Figure 6.4.5d shows the time consumed to calculate the coloring semantics and the constraint automata semantics of *N-Sequencer*s using the ECT tool-set. For $N = 16$, the computation of coloring semantics fails with the stack overflow error. The same happens while computing the constraint automata semantics for $N = 21$.

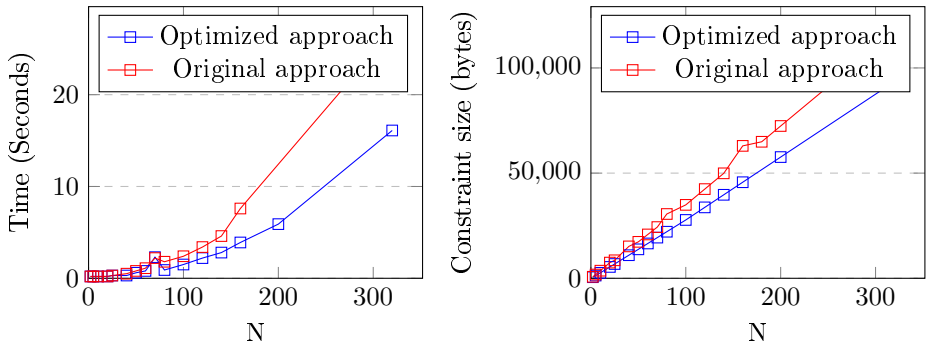
As the results show our approach can handle bigger models compared to the existing ECT tools. It is interesting to observe that the difference between the original and optimized approaches becomes more significant for bigger values of $N$. Another possible optimization point is the call to Reduce program that is currently implemented by invoking the program externally. We expect a better performance due to reduction of external invocation overhead by including the source code of the Reduce Algebra System in our tool.

## 6.5 Conclusions

In this chapter, we have presented a constraint-based framework that encodes the semantics of Reo networks as constraint satisfaction problems whose predicates are either Boolean propositions or numerical constraints. We presented a hybrid approach to find the solutions for these problems.

An advantage of our approach is that it treats data constraints symbolically to mitigate the state explosion problem. From this solution, we construct the semantic model corresponding to a Reo network in the form of constraint automata with state memory.

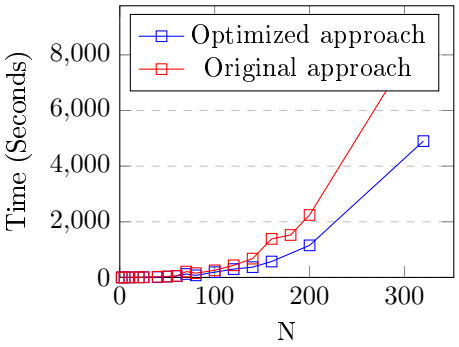
Our framework supports *product* and *hiding* operations on constraint automata.



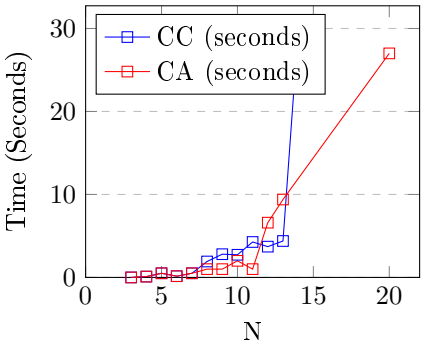

**(a)** Time (s), single solution




**(b)** Size of the RCSP




**(c)** Time (s), calculating the RLTS




**(d)** Time (s), building CA and connector coloring table

**Figure 6.4.5**: Performance evaluation based on N-Sequencer network

We have implemented and integrated our approach as a tool in the ECT. In the next section, we use this framework to encode priority. It makes our work the most expressive framework that exists to analyze Reo networks.