



Universiteit
Leiden
The Netherlands

Constraint-based analysis of business process models

Changizi, B.

Citation

Changizi, B. (2020, February 21). *Constraint-based analysis of business process models*. IPA Dissertation Series. Retrieved from <https://hdl.handle.net/1887/85677>

Version: Publisher's Version

License: [Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden](#)

Downloaded from: <https://hdl.handle.net/1887/85677>

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

Cover Page



Universiteit Leiden



The handle <http://hdl.handle.net/1887/85677> holds various files of this Leiden University dissertation.

Author: Changizi, B.

Title: Constraint-based analysis of business process models

Issue Date: 2020-02-25

4

Formal Semantics for Reo

4.1 Introduction

A benefit of employing coordination languages in general and Reo in particular is that they express the coordination patterns explicitly and separate them from the computational part of the code. This opens up possibilities for performing various types of analysis and automation such as model checking, code generation, automated test generation, etc.

To be able to perform such tasks, it is insufficient to describe the behavior of Reo models in a verbal manner. We need a more rigorous way to unambiguously specify semantics of Reo models.

Several formal semantics have been proposed in the recent years that express the behavior of Reo connectors. Jongmans et al. [JA12] present a comprehensive overview of thirty models. They grouped these models into the following categories:

- *Coalgebraic models*: Two coalgebraic semantics of Reo, Timed Data Streams [Arb02] [AR02] [RKNP04] and Record Streams [IB08] [IBC11] rely on the coalgebraic concept of *stream*, which refers to an infinite sequence of elements of a given set. This class of semantics are difficult to use for analysis purpose, for

instance, as an underlying model to apply model checking techniques [JA12].

- *Automata-based semantics:* A big number of Reo operational semantics are based on automata. States in these automata correspond to the states of a Reo network, while the transitions denote I/O operations.

A list of automata based semantics for Reo are: port automata (PA) [KC09], Constraint Automata [BSAR06], Labeled Constraint Automata (LCA) [KB09], Timed Constraint Automata (TCA) [ABBR04], Probabilistic Constraint Automata [Bai05], Quantitative Constraint Automata (QCA) [ACMM07] [MA09], Continuous Time Constraint Automata (CTCA) [BW06], Resource Sensitive Timed Constraint Automata (RSTCA) [MA07a], Transactional Constraint Automata (TNCA) [MA10], Behavioral Automata (BA) [Pro11], Buchi Automata [IB08] [IBC11] [IBC08] [IBC11], Guarded Automata [BCS12] [Mar09], Stochastic Guarded Automata [MSKA10] [MSKA14], Intentional Automata [Cos10], Quantitative Intentional Automata [ACvdM⁺09], and Action Constraint Automata [KCA10].

- *Structural operational semantics:* Some of the semantics proposed for Reo are expressed in terms of structural operational semantics. Sun Meng et al. [MAA⁺12] model Reo networks in terms of the Unifying Theories of Programming (UTP) [Hoa13]. A UTP design consists of predicates that express assumptions on inputs and commitments on outputs.

Another work in this field is done by Mousavi et al. [MSA06]. They present a Structural Operational Semantics (SOS) for some of Reo primitives in Gordon Plotkin's style [Plo04]. In the proposed semantics, data-flow of a Reo connector is represented by a set of rules, which pair the structure of the connector with functions that map the nodes to potentially infinite sequences of data items.

Tile Model [ABC⁺09] is a more recent SOS-based formal semantics for Reo that extends Gordon Plotkin's SOS inference rules. In this model, transitions are described as movements from an initial state to a final state upon firing related triggers.

Tile Model defines composition in three ways:

- horizontal composition that models synchronization, where the effect of one tile is a trigger for another tile,
- vertical composition, which is a composition occurring in time. This is when the final state of one tile matches the initial state of another tile,

– parallel composition that captures concurrency.

- *Semantics based on graph-coloring.* Connector coloring (CC) [CCA07] is a formal semantics for Reo that describes the behavior of a connector by assigning different colors to its ports.

The colors designate presence or absence of data-flow. This model accounts for synchronization and context dependency. It captures context dependency by propagating negative information about the absence of data-flow inside a Reo network.

The most important types of semantics that have influenced and provided basis for the other classes of semantics are constraint automata and coloring semantics. These models are the underlying models of several tools for Reo ranging from animation to testing and model checking.

In this chapter, we present the definition and examples for Reo semantics that are relevant to this thesis. In addition, we briefly discuss the time complexity of obtaining formal semantics of a Reo network using the computation rules defined by the formal semantics.

4.2 Constraint automata

Definition 4.2.1 (Constraint automaton [BSAR06]) *A constraint automaton is a tuple $\mathcal{A} = (Q, \mathcal{N}, \rightarrow, q_0)$, where*

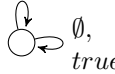
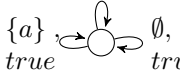
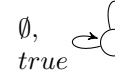
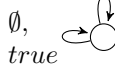
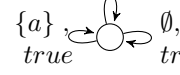


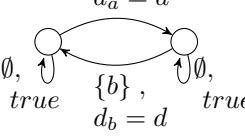
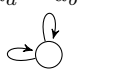

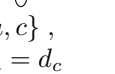
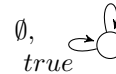
- Q is a set of states,
- \mathcal{N} is a set of port names,
- $\rightarrow \subseteq Q \times 2^{\mathcal{N}} \times DC \times Q$ is a transition relation, where DC is the set of data constraints over a finite data domain $Data$,
- $q_0 \in Q$ is an initial state.

We write $q \xrightarrow{N,g} p$ instead of $(q, N, g, p) \in \rightarrow$. Table 4.2.1 depicts the CA corresponding to the most common Reo elements.

Constraint automata have a compositional nature. Therefore, the semantics of a whole model can be obtained through the *composition* of the given semantics of its participant elements.

Following is the definition of the *product* operator, which performs the composition.

Table 4.2.1: Constraint automata for basic Reo primitives

$\{a, b\},$ $d_a = d_b$  CA corresponding to $a \bullet \longrightarrow b$	$\{a, b\},$ $d_a = d_b$ $\{a\},$ $true$  CA corresponding to $a \bullet \cdots \longrightarrow b$	$\{a, b\},$ $true$  CA corresponding to $a \bullet \longleftrightarrow b$
$\{a, b\},$ $true$  CA corresponding to $a \longleftrightarrow b$	$\{b\}, true$ $\{a\}, true$  CA corresponding to $a \bullet \dashrightarrow b$	$\emptyset, true$ $\{a, b\},$ $expr(d_a)$ $\wedge d_a = d_b$ $\{a\},$ $\neg expr(d_a)$  CA corresponding to $a \bullet \text{p} \longrightarrow b$
$\{a, b\},$ $d_b = f(d_a)$  CA corresponding to $a \bullet \xrightarrow{f} b$	$\{a\},$ $d_a = d$  CA corresponding to $a \bullet \boxed{} \longrightarrow b$	$\{a, b, c\},$ $d_a = d_b = d_c$  CA corresponding to $a \bullet \begin{matrix} \nearrow b \\ \searrow c \end{matrix}$
$\{a, b\},$ $d_a = d_b$  $\{a, c\},$ $d_a = d_c$  CA corresponding to $a \bullet \otimes \begin{matrix} b \\ c \end{matrix}$	$\{a, b, c\},$ $d_c = < d_a, d_b >$  CA corresponding to $a \bullet \oplus \longrightarrow c$	

Definition 4.2.2 (Product on constraint automata) *The product of constraint automata $\mathcal{A}_1 = (Q_1, \mathcal{N}_1, \rightarrow_1, q_{0,1})$ and $\mathcal{A}_2 = (Q_2, \mathcal{N}_2, \rightarrow_2, q_{0,2})$ is defined as:*

$$\mathcal{A}_1 \bowtie \mathcal{A}_2 = (Q_1 \times Q_2, \mathcal{N}_1 \cup \mathcal{N}_2, \rightarrow, q_{0,1} \times q_{0,2})$$

where the following rules define the transition relation \rightarrow :

$$\frac{q_1 \xrightarrow{N_1, g_1} p_1, q_2 \xrightarrow{N_2, g_2} p_2, N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1}{\langle q_1, q_2 \rangle \xrightarrow{N_1 \cup N_2, g_1 \wedge g_2} \langle p_1, p_2 \rangle}$$

$$\frac{q_1 \xrightarrow{N_1, g_1} p_1, N_1 \cap \mathcal{N}_2 = \emptyset}{\langle q_1, q_2 \rangle \xrightarrow{N_1, g_1} \langle p_1, q_2 \rangle}$$

$$\frac{q_2 \xrightarrow{N_2, g_2} p_2, \mathcal{N}_1 \cap N_2 = \emptyset}{\langle q_1, q_1 \rangle \xrightarrow{N_2, g_2} \langle q_1, q_2 \rangle}$$

We can abstract from the data-flow on certain Reo nodes using the *hiding* operator defined as follows:

Definition 4.2.3 (Hiding on constraint automata) *Let $\mathcal{A} = (Q, \mathcal{N}, \rightarrow, q_0)$ be a CA and $C \in \mathcal{N}$.*

The constraint automaton that results from hiding the node C in automaton \mathcal{A} is $\exists C[\mathcal{A}] = (Q, \mathcal{N} \setminus \{C\}, \rightarrow_C, q_0)$ and the transition relation \rightarrow_C is defined as follows:

$$\frac{p \xrightarrow{N, g} q, N' = N \setminus \{C\}, g' = \exists C[g]}{p \xrightarrow{N', g'}_C q}, \text{ where}$$

$$\exists C[g] = \bigvee_{d \in \mathcal{D}} g[d(C)/d].$$

Example 4.2.1 *Figure 4.2.2 depicts the CA semantics of the Reo network of Figure 4.2.1. According to CA, it is possible that the lossySync channel loses the incoming data in the state q , where the FIFO_1 channel is empty. This is an example of undesired behavior that is the result of the fact that CA is not a context-dependent semantics.*

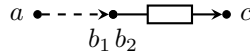


Figure 4.2.1: A context-dependent Reo connector

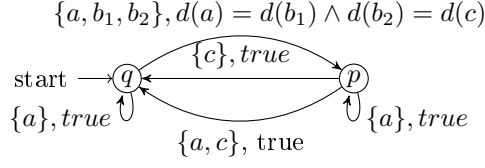


Figure 4.2.2: Constraint automaton of the Reo network of Figure 4.2.1

Example 4.2.2 Figure 4.2.4 illustrates the CA of the Reo network of Figure 4.2.3. Since, CA is data-aware it can describes the correct behavior of this data-aware network.

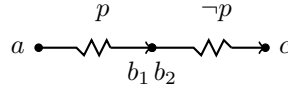


Figure 4.2.3: A data-aware Reo connector

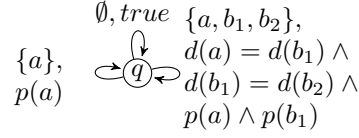


Figure 4.2.4: Constraint automaton of the Reo network of Figure 4.2.3

4.3 Constraint automata with state memory

Constraint automata with state memory (CASM) [PSHA12] extends CA with variables that represent local memory cells of automata states. Because CASM elaborates on state information, we choose to use CASM instead of CA, in our work.

Definition 4.3.1 (Constraint automaton with state memory) A constraint automaton with state memory (CASM) is a tuple $A = (Q, \mathcal{N}, \rightarrow, q_0, \mathcal{M})$ where

- Q is a finite set of states.
- \mathcal{N} is a finite set of names.
- \rightarrow , a finite subset of $Q \times 2^{\mathcal{N}} \times DC(\mathcal{N}, \mathcal{M}, \mathcal{D}) \times Q$, is the transition relation of A , where $DC(\mathcal{N}, \mathcal{M}, \mathcal{D})$ is the set of data constraints, defined below.

- $q_0 \in Q$ is an initial state.
- \mathcal{M} is a set of memory cell names, where $\mathcal{N} \cap \mathcal{M} = \emptyset$.

Every $n \in \mathcal{N}$ represents a node in a Reo connector. The set \mathcal{N} is partitioned into three mutually disjoint sets of source nodes \mathcal{N}^{src} , mixed nodes \mathcal{N}^{mix} , and sink nodes \mathcal{N}^{snk} .

Because we make the replication and merge inherent in Reo nodes explicit as *replicator* and *merger* primitives, at most two primitive ends coincide on every node $n \in \mathcal{N}$. Thus, it follows that a source or a sink node contains only a single (source or sink) primitive end, and a mixed node contains exactly one source and one sink primitive ends.

We write $q \xrightarrow{N,g} p$ instead of $(q, N, g, p) \in \rightarrow$. For every transition $q \xrightarrow{N,g} p$, we require that $g \in DC(N, \mathcal{M}, \mathcal{D})$, where \mathcal{D} is the global set of numerical data values and $DC(N, \mathcal{M}, \mathcal{D})$ is the language defined by the following grammar:

$$\begin{aligned}
g &::= \text{true} \mid \neg g \mid g \wedge g \mid u = u \mid u < u, \\
u &::= d(n) \mid m' \mid m \mid v.
\end{aligned}$$

In this grammar,

- $=$ is the symmetric equality relation,
- $<$ is a total order relation,
- $n \in N \subseteq \mathcal{N}$ denotes a node name,
- $d(n)$ represents the data item exchanged through the node n ,
- $m \in \mathcal{M}$ correspond to a memory cell in the current state, which is the source state of the transition,
- m' stands for the memory cell $m \in \mathcal{M}$ in the next state, which is the target state of the transition,
- $v \in \mathcal{D}$.

As usual, *false* stands for $\neg \text{true}$, $x > y$ stands for $y < x$, and other logical operators, such as \vee and \Rightarrow (the implication symbol) can be built from the given operators.

Transitions with data constraints that can be reduced to *false* using the Boolean laws are impossible and we omit them. A data constraint g that is always *true* can be left out.

We use \mathcal{M}_g to represent the set of all $m \in \mathcal{M}$ that syntactically appear as m in a data constraint g ; and \mathcal{M}'_g to refer to the set of all $m \in \mathcal{M}$ that syntactically appear as m' in g .

The valuation function $\mathcal{V}_q : \mathcal{M} \rightarrow 2^{\mathcal{D}}$ designates the set of values $\mathcal{V}_q(m)$ of a memory cell $m \in \mathcal{M}$ in a state $q \in Q$, where $\mathcal{V}_{q_0}(m) = \emptyset$ for all $m \in \mathcal{M}$.

A transition $q \xrightarrow{N,g} p$ in a given constraint automaton with state memory is possible only if there exists a substitution for every syntactic element $d(n)$, m , and m' that appears in g to satisfy g .

A substitution simultaneously replaces in g :

- every occurrence of $d(n)$ with the data value exchanged through the node $n \in \mathcal{N}$;
- every occurrence of m' of every $m \in \mathcal{M}$ with a value $v \in \mathcal{D}$;
- every occurrence $m \in \mathcal{M}$ with:
 - the special symbol ' \circ ' if $\mathcal{V}_q(m) = \emptyset$,
 - a value $v \in \mathcal{V}_q(m)$, otherwise.

The guard g is satisfied if proper replacement values can be found to make g *true*. Making this transition, the automaton defines the valuation function \mathcal{V}_p for the target state p , as follows:

- For every $m \in \mathcal{M}'_g$, $\mathcal{V}_p(m)$ is the set of all $v \in \mathcal{D}$ whose replacements for m' satisfy g .
- For every other $m \in \mathcal{M}$, $\mathcal{V}_p(m) = \emptyset$.

A relational operator evaluates to *true* only if the values of its operands are in its respective relation. Thus, any operator with one or more \circ as an operand always evaluates to *false*.

We call a CASM, *normalized* iff

- It does not have two states with the same set of state memory variables.
- Every two transitions differ at least in their start states, their target states, or their sets of synchronizing ports.

For any arbitrary CASM that is not normalized, we can normalize it by

- introducing auxiliary variables, to make the set of state memory variables unique for each state,

- by merging the transitions that have the same start and target states and synchronize the same ports.

In the sequel, we consider only *normalized* CASMs.

Following are the definitions for *product* and *hiding* operations on CASM. Both definitions are adapted from [BSAR06].

Definition 4.3.2 (Product automaton on CASM) *The product of CASMs $\mathcal{A}_1 = (Q_1, \mathcal{N}_1, \rightarrow_1, q_{0,1}, \mathcal{M}_1)$ and $\mathcal{A}_2 = (Q_2, \mathcal{N}_2, \rightarrow_2, q_{0,2}, \mathcal{M}_2)$ is defined as:*

$$\mathcal{A}_1 \bowtie \mathcal{A}_2 = (Q_1 \times Q_2, \mathcal{N}_1 \cup \mathcal{N}_2, \rightarrow, q_{0,1} \times q_{0,2}, \mathcal{M}_1 \cup \mathcal{M}_2)$$

where the following rules define the transition relation \rightarrow :

$$\frac{q_1 \xrightarrow{N_1, g_1} p_1, \quad q_2 \xrightarrow{N_2, g_2} p_2, \quad N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1}{\langle q_1, q_2 \rangle \xrightarrow{N_1 \cup N_2, g_1 \wedge g_2} \langle p_1, p_2 \rangle}$$

$$\frac{q_1 \xrightarrow{N_1, g_1} p_1, \quad N_1 \cap \mathcal{N}_2 = \emptyset}{\langle q_1, q_2 \rangle \xrightarrow{N_1, g_1} \langle p_1, q_2 \rangle} \quad \frac{q_2 \xrightarrow{N_2, g_2} p_2, \quad \mathcal{N}_1 \cap N_2 = \emptyset}{\langle q_1, q_2 \rangle \xrightarrow{N_2, g_2} \langle q_1, p_2 \rangle}$$

Similar to CA, we can abstract from the data-flow on certain Reo nodes using the *hiding* operator defined as follows:

Definition 4.3.3 (Hiding on CASM) *Let $\mathcal{A} = (Q, \mathcal{N}, \rightarrow, q_0, \mathcal{M})$ be a CASM and $C \in \mathcal{N}$.*

The constraint automaton that results from hiding the node C in automaton \mathcal{A} is $\exists C[\mathcal{A}] = (Q, \mathcal{N} \setminus \{C\}, \rightarrow_C, q_0, \mathcal{M})$ and the transition relation \rightarrow_C is defined as follows:

$$\frac{p \xrightarrow{N, g} q, \quad N' = N \setminus \{C\}, \quad g' = \exists C[g]}{p \xrightarrow{N', g'}_C q}, \quad \text{where}$$

$$\exists C[g] = \bigvee_{d \in \mathcal{D}} g[d(C)/d].$$

To facilitate our further reasoning with CASM, we provide the following definition that gives the set of state memories used in each state.

Definition 4.3.4 (State variables) *Given the CASM $\mathcal{A} = (Q, \mathcal{N}, \rightarrow, q_0, \mathcal{M})$, we define the function $S : Q \rightarrow 2^{\mathcal{M}}$ as for each $q \xrightarrow{N, g} p$, $m \in V_g \Rightarrow m \in S(q)$ and $m' \in V_g \Rightarrow m \in S(p)$.*

Example 4.3.1 Figure 4.3.2 depicts the CASM for the Reo shown network in Figure 4.3.1. CASM provides an explicit representation for the stored values using its state variables.

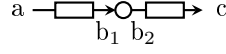


Figure 4.3.1: FIFO₂

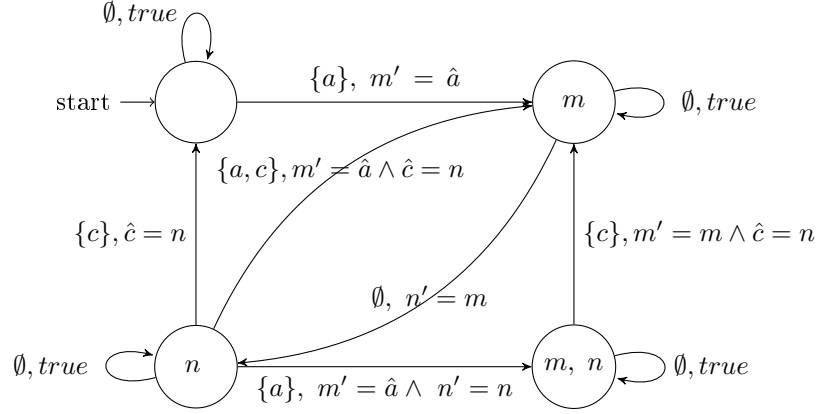






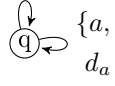
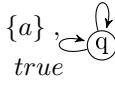
Figure 4.3.2: Constraint automaton of the Reo network of Figure 4.3.1

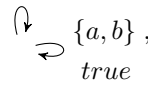
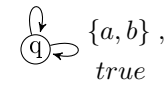
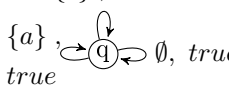
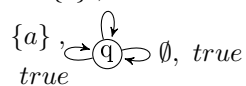
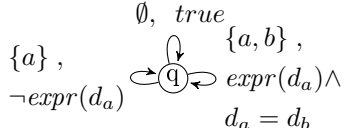
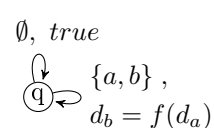
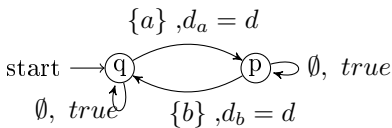
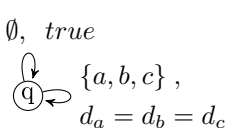
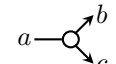
4.4 Constraint automata with priority


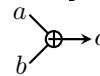
Definition 4.4.1 (Constraint automaton with priority) A constraint automaton with priority is a tuple $\mathcal{P} = (\mathcal{A}, \mathcal{R}, \mathcal{S}, \mathcal{T})$ where

- $\mathcal{A} = (\mathcal{Q}, \mathcal{N}, \mathcal{N}^{mix}, \mathcal{N}^{src}, \mathcal{N}^{snk}, \longrightarrow, \mathcal{Q}_0)$ is a constraint automaton,
- $\mathcal{R} \subset 2^{\mathcal{N}} : \forall R \in \mathcal{R}$ is a subset of \mathcal{N} , such that if a node $n \in R$ connects to the priority imposing channel, *PrioritySync*, the priority affects $\bar{n} \in R$.
- $\mathcal{S} \subset \mathcal{R} \times \mathcal{R}$ is the set pairs of subsets of \mathcal{N} , such that $\forall (X, Y) \in \mathcal{S}$, the priority imposed on the region X can propagate to the region Y ,
- $\mathcal{T} =^{def} (t, \triangleleft) : t \in R \text{ and } \triangleleft \subseteq \longrightarrow \times \longrightarrow$ is a binary relation on the transitions of \mathcal{A} such that $q \xrightarrow{N, g} p \triangleleft \bar{q} \xrightarrow{\bar{N}, \bar{g}} \bar{p}$ implies $q = \bar{q}$ and $(N, g) \neq (\bar{N}, \bar{g})$.

Table 4.4.1: Priority constraint automata of commonly used Reo primitives

$\{a, b\}, d_a = d_b$  $\emptyset, true$ $Q_0 = \{q\},$ $R = \{\{a, b\}\},$ $S = 1,$ $T = \{\{a, b\} :$ $q \xrightarrow{\{a, b\}, d_a = d_b} q \triangleleft q \xrightarrow{\emptyset, true} q\}$ CAP corresponding to $a \bullet \dashrightarrow b$	$\{a, b\}, d_a = d_b$  $\emptyset, true$ $Q_0 = \{q\},$ $R = \{\{a\}, \{b\}\},$ $S = 1 \cup \{(\{b\}, \{a\})\},$ $T = \emptyset$ CAP corresponding to $a \bullet \rightarrow b$
$\{a, b\}, d_a = d_b$  $\emptyset, true$ $Q_0 = \{q\},$ $R = \{\{a, b\}\},$ $S = 1 \cup \{(\{a\}, \{b\})\},$ $T = \emptyset$ CAP corresponding to $a \bullet \leftarrow b$	$\{a, b\}, d_a = d_b$  $\emptyset, true$ $Q_0 = \{q\},$ $R = \{\{a\}, \{b\}\},$ $S = 1 \cup \{(\{a\}, \{b\}), (\{b\}, \{a\})\},$ $T = \emptyset$ CAP corresponding to $a \bullet \leftrightarrow b$
$\emptyset, true$  $\{a, b\}, d_a = d_b$ $Q_0 = \{q\},$ $R = \{\{a, b\}\},$ $S = 1,$ $T = \emptyset$ CAP corresponding to $a \bullet \rightarrow b$	$\{a, b\}, d_a = d_b$  $\{a\}, true$ $Q_0 = \{q\},$ $R = \{\{a, b\}\},$ $S = 1,$ $T = \{\emptyset : q \xrightarrow{\{a, b\}, d_a = d_b} q \triangleleft q \xrightarrow{\{a\}, true} q,$ $\emptyset : q \xrightarrow{\{a, b\}, d_a = d_b} q \triangleleft q \xrightarrow{\emptyset, true} q,$ $\emptyset : q \xrightarrow{\{a\}, true} q, \triangleleft q \xrightarrow{\emptyset, true} q\}$ CAP corresponding to $a \bullet \dashrightarrow b$

$\emptyset, true$  $Q_0 = \{q\},$ $R = \{\{a, b\}\},$ $S = 1,$ $T = \emptyset$ CAP corresponding to $a \longleftrightarrow b$	$\emptyset, true$  $Q_0 = \{q\},$ $R = \{\{a, b\}\},$ $S = 1,$ $T = \emptyset$ CAP corresponding to $a \longleftrightarrow b$
$\{b\}, true$  $Q_0 = \{q\},$ $R = \{\{a, b\}\},$ $S = 1,$ $T = \emptyset$ CAP corresponding to $a \rightsquigarrow b$	$\{b\}, true$  $Q_0 = \{q\},$ $R = \{\{a, b\}\},$ $S = 1,$ $T = \emptyset$ CAP corresponding to $a \rightsquigarrow b$
$\emptyset, true$  $Q_0 = \{q\}, R = \{\{a, b\}\},$ $S = 1, T = \emptyset$ CAP corresponding to $a \xrightarrow{p} b$	$\emptyset, true$  $Q_0 = \{q\}, R = \{\{a, b\}\},$ $S = 1, T = \emptyset$ CAP corresponding to $a \xrightarrow{f} b$
$\{a\}, d_a = d$  $Q_0 = \{q\}, R = \{\{a\}, \{b\}\},$ $S = 1, T = \emptyset$ CAP corresponding to $a \square b$	$\emptyset, true$  $Q_0 = \{q\}, R = \{\{a, b, c\}\},$ $S = 1, T = \emptyset$ CAP corresponding to 

\emptyset, true $\{a, c\}, \{a, b\},$ $d_a = d_c \quad d_a = d_b$ $Q_0 = \{q\},$ $R = \{\{a, b, c\}\},$ $S = 1,$ $T = \emptyset$ CAP corresponding to 	$\emptyset, \{a, b, c\},$ $\text{true} \quad d_c = \langle d_a, d_b \rangle$ $Q_0 = \{q\},$ $R = \{\{a, b, c\}\},$ $S = 1,$ $T = \emptyset$ CAP corresponding to 
--	---

Observe that the nodes in R connect to each other by priority propagating channels such as Sync, PrioritySync, SyncDrain. The connections of the regions paired in S is, however, via priority blocking channels like BlockingSinkSync, BlockingSourceSync and AsyncDrain. The sets \mathcal{R} , \mathcal{S} and the tag t in \mathcal{T} are auxiliary concepts for composition of CAPs. Table 4.4.1 shows CAPs corresponding to Reo elements.

Similar to CA, the *product-automaton* operator (\bowtie) computes the CAP corresponding to a Reo network from CAPs of its substituent elements.

Let \mathcal{P}_1 and \mathcal{P}_2 be the two CAPs, $\tau_1, \lambda_1 \in \longrightarrow_1$, $\tau_1 \triangleleft \lambda_1$ and $\tau_2, \lambda_2 \in \longrightarrow_2$. If τ_1 and τ_2 synchronize to form a transition $\tau \in \longrightarrow_{P_1 \bowtie P_2}$, λ_1 and λ_2 synchronize to form a transition $\lambda \in \longrightarrow_{P_1 \bowtie P_2}$, the relation of $\tau \triangleleft \lambda$ is *full lifting* of the $\tau_1 \triangleleft \lambda_1$.

Since the priority blocking channels can affect the propagation of the priority, the priority relations that full lifting defines are not always valid on the product of the automata. We need to eliminate invalid transitions that are results of improper propagation of the priority.

The following three cases are the only valid propagation of the priority [ABS15]:

- *Propagation over empty transitions*: If λ is an empty transition, then λ_1 and λ_2 are also empty transitions. In this case, full lifting brings a new priority imposition as: $\tau \triangleleft \lambda$.
- *Propagation by containment*: If λ_1 is a proper transition, then λ is a proper transitions, which contains λ_1 . Therefore, full lifting is a natural growth of the previously imposed priority that preserves the priority relation as: $\tau \triangleleft \lambda$.
- *Propagation by seepage*: If λ_1 is an empty transition, but λ is a proper transition, then λ_2 is also a proper transition. Under this condition, full lifting

is not always valid. Therefore, we need more restriction to preserve the new priority relation that full lifting impose that is $\tau \triangleleft \lambda$. The seepage relation \mathcal{S} and the tag t of the transition help to check the validity of full lifting for this case. So, the full lifting is valid if there exists a finite sequence of regions $r_0, \dots, r_i, r_{i+1}, \dots, r_n$ such that $r_i \in R, (r_i, r_{i+1}) \in \mathcal{S}, r_0 = t$ and r_n includes all nodes involved in the transition λ_2 . Note that \mathcal{S} is the seepage relation that defines the allowed propagation of the priority through regions. Observe that if $t_1 = \emptyset$, then $t = \emptyset$. Since $\emptyset \notin \mathcal{R}$, such a sequence does not exist and the full lifting is not valid.

Following is the definition of the CAP product operator.

Definition 4.4.2 (Product-automaton) Let $\mathcal{P}_i = (\mathcal{A}_i, \mathcal{R}_i, \mathcal{S}_i, \mathcal{T}_i)$, $i = 1, 2$ be two CAPs, where $\mathcal{A}_i = (\mathcal{Q}_i, \mathcal{N}_i, \mathcal{N}_i^{mix}, \mathcal{N}_i^{src}, \mathcal{N}_i^{snk}, \longrightarrow, \mathcal{Q}_{0,i})$, such that:

$$\mathcal{N}_1 \cap \mathcal{N}_2 \subseteq \mathcal{N}_1^{src} \cap \mathcal{N}_2^{snk} \cup \mathcal{N}_1^{snk} \cup \mathcal{N}_2^{src}$$

The definition of the product-automaton $\mathcal{P}_1 \bowtie \mathcal{P}_2 = (\mathcal{A}_1 \bowtie \mathcal{A}_2, \mathcal{R}, \mathcal{S}, \mathcal{T})$ follows:

Listing 4.1: Calculating \mathcal{R}

```

 $\mathcal{R} := \emptyset$ 
for each  $r_1 \in \mathcal{R}_1$ 
  if  $\exists r_2 \in \mathcal{R}_2 : r_1 \cap r_2 \neq \emptyset$ 
     $\mathcal{R} := \mathcal{R} \cup r_1 \cup r_2$ 
  else
     $\mathcal{R} := \mathcal{R} \cup r_1$ 
for each  $r_2 \in \mathcal{R}_2$ 
  if  $\nexists r_1 \in \mathcal{R}_1 : r_1 \cap r_2 \neq \emptyset$  then
     $\mathcal{R} = \mathcal{R} \cup r_2$ 

```

Listing 4.2: Calculating seepage relation \mathcal{S}

```

 $\mathcal{S} := \emptyset$ 
for each  $(u_1, v_1) \in \mathcal{S}_1$ 
   $\mathcal{S} += (big(u_1), big(v_1))$ 
  for each  $(u_2, v_2) \in \mathcal{S}_2$ 
     $\mathcal{S} += (big(u_2), big(v_2))$ 
   $\mathcal{S} += \mathcal{I}$ 

```


Let $(t_1, \tau_1 \triangleleft_1 \lambda_1) \in \mathcal{T}_1$. The transition λ_1 is either empty or proper:

$$\begin{array}{ll}
\forall \tau_2 \in \longrightarrow_2 : \tau_1 \cap \tau_2 \neq \emptyset & \text{if } \lambda_1 \text{ is empty} \\
big(r_1) : \tau_1 \parallel \tau_2 \triangleleft \emptyset & \\
\forall \tau_2 \in \longrightarrow_2 : \tau_1 \cap \tau_2 = \emptyset & \\
\text{if exists a sequence such that} & \text{otherwise} \\
\forall \tau_2 : \tau_1 \cap \tau_2 \neq \emptyset & \lambda_1 \text{ is proper} \\
\forall \lambda_2 : \lambda_1 \cap \lambda_2 \neq \emptyset & \\
big(r_1) : \tau_1 \parallel \tau_2 \triangleleft \lambda_1 \parallel \lambda_2 &
\end{array} \tag{4.1}$$

4.5 Connector coloring

The connector coloring semantics [CCA07] denote the existence or absence of data-flow through the primitive ends by marking them with different colors.

Let *Colors* be a set of colors. In a set of two colors, $Colors = \{-, -\}$, $-$ denotes an occurrence and $-$ represents an absence of data-flow. Two colors are adequate to express the formal semantics of many Reo networks. However, they cannot express the semantics of context-dependent Reo networks.

Such a network presented in Example 4.2.2 is when the sink end of a *lossySync* channel connects to an empty *FIFO*₁ channel; in this case, the semantics of this network according to the two-color set includes the case where the *lossySync* loses its incoming data item, while the *FIFO*₁ channel is empty. This is an unacceptable behavior for a so-called context-dependent *lossySync* channel: it must lose its incoming data only if its sink end cannot dispense it. In the sequel, when we refer to a *lossySync* we mean its context sensitive version.

The three coloring semantics, $Colors = \{-, \triangleleft, \triangleright\}$, addresses this problem by propagating negative information regarding the absence of data-flow. It replaces $-$ with \triangleleft and \triangleright meaning that the associated primitive end, respectively, *provides* or *requires* a reason for no-flow.

Considering that no-flow can occur only when at least one of the involved primitive ends *provides* a reason for it, and that an empty *FIFO*₁ cannot *provide* a reason for no-flow on its source end, the invalid behavior described above does not arise in the three coloring semantics.

Definition 4.5.1 (Coloring) *A coloring $l : \mathcal{P} \rightarrow Colors$ is a total function from the primitive ends to a set of colors. We refer to the global set of colorings as \mathcal{L} .*

Definition 4.5.2 (Coloring composition) *The composition of colorings l_1 and l_2 , denoted $l_1 \bullet l_2$, is defined as:*

$$l_1 \bullet l_2 = \{ \\
\begin{array}{l}
c_1 \cup c_2 | c_1 \in l_1, c_2 \in l_2, p_1 \in \text{dom}(c_1), p_2 \in \text{dom}(c_2), \\
p_1 \text{ and } p_2 \text{ are the source and sink ends of a node } n, \\
\neg (c_1(p_1) = \triangleleft \wedge c_2(p_2) = \triangleright)
\end{array} \\
\}$$

Definition 4.5.3 (Coloring table) A coloring table over the primitive set $P \subseteq \mathcal{P}$ is a set of colorings with the domain P .

Definition 4.5.4 (Next function) The next function $\eta : \mathcal{L} \times 2^{\mathcal{L}} \rightarrow 2^{\mathcal{L}}$ maps a pair of a coloring and a coloring table to a colorings table.

Definition 4.5.5 (Coloring semantics) A coloring semantics of a Reo network is a tuple $CC = \langle \mathcal{P}, 2^{\mathcal{L}}, l_0, \eta \rangle$, where:

- \mathcal{P} is the set of primitive ends,
- $l_0 \in \mathcal{L}$ is the initial set of possible colorings,
- $2^{\mathcal{L}}$ is a set of colorings,
- η is a next function that maps a pair of a coloring and a coloring table into a coloring table.

Example 4.5.1 Table 4.5.1 depicts the CC for the network shown in Figure 4.5.1. The two flows described in the table correspond to the cases; i) when there is a write request of the end a , then the ends a, b_1 and b_2 have a flow, but the end c provides a reason for no flow, ii) when there is no write request present on the end a , therefore the ends a and b_2 require a reason for no flow and the ends b_1 and c provides a reason for no flow. Since CC is context-sensitive, it can capture the semantics of the given network correctly.

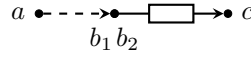


Figure 4.5.1: A context-dependent Reo connector

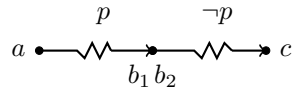
Table 4.5.1: Connector coloring semantics of the Reo network of Figure 4.5.1

a	b_1	b_2	c
—	—	—	▷
▷	▷	▷	▷

Table 4.5.2: Connector coloring semantics of commonly used Reo primitives

<table><tr><td>a</td><td>b</td></tr><tr><td>×</td><td>×</td></tr><tr><td>○</td><td>●</td></tr><tr><td>●</td><td>○</td></tr></table> <p>CC corresponding to $a \bullet \longrightarrow \bullet b$</p>	a	b	×	×	○	●	●	○	<table><tr><td>a</td><td>b</td></tr><tr><td>×</td><td>×</td></tr><tr><td>○</td><td>×</td></tr><tr><td>×</td><td>○</td></tr></table> <p>CC corresponding to $a \bullet \text{---} \twoheadrightarrow b$</p>	a	b	×	×	○	×	×	○	<table><tr><td>a</td><td>b</td></tr><tr><td>×</td><td>×</td></tr><tr><td>○</td><td>●</td></tr><tr><td>●</td><td>○</td></tr></table> <p>CC correspondence to $a \bullet \longleftrightarrow \bullet b$</p>	a	b	×	×	○	●	●	○
a	b																									
×	×																									
○	●																									
●	○																									
a	b																									
×	×																									
○	×																									
×	○																									
a	b																									
×	×																									
○	●																									
●	○																									
<table><tr><td>a</td><td>b</td></tr><tr><td>×</td><td>×</td></tr><tr><td>○</td><td>●</td></tr><tr><td>●</td><td>○</td></tr></table> <p>CC correspondence to $a \longleftarrow \bullet b$</p>	a	b	×	×	○	●	●	○	<table><tr><td>a</td><td>b</td></tr><tr><td>×</td><td>×</td></tr><tr><td>○</td><td>●</td></tr><tr><td>●</td><td>○</td></tr></table> <p>CC correspondence to $a \bullet \dashleftarrow \bullet b$</p>	a	b	×	×	○	●	●	○	<table><tr><td>a</td><td>b</td></tr><tr><td>×</td><td>×</td></tr><tr><td>○</td><td>●</td></tr><tr><td>●</td><td>○</td></tr></table> <p>CC correspondence to $a \bullet \longleftrightarrow \bullet b$</p>	a	b	×	×	○	●	●	○
a	b																									
×	×																									
○	●																									
●	○																									
a	b																									
×	×																									
○	●																									
●	○																									
a	b																									
×	×																									
○	●																									
●	○																									
<table><tr><td>a</td><td>b</td></tr><tr><td>×</td><td>×</td></tr><tr><td>○</td><td>●</td></tr><tr><td>●</td><td>○</td></tr></table> <p>CC correspondence to $a \bullet \boxed{} \rightarrow b$</p>	a	b	×	×	○	●	●	○	<table><tr><td>a</td><td>b</td></tr><tr><td>×</td><td>×</td></tr><tr><td>○</td><td>●</td></tr><tr><td>●</td><td>○</td></tr></table> <p>CC correspondence to $a \bullet \boxed{\bullet} \rightarrow b$</p>	a	b	×	×	○	●	●	○									
a	b																									
×	×																									
○	●																									
●	○																									
a	b																									
×	×																									
○	●																									
●	○																									

Example 4.5.2 Table 4.5.3 shows the CC of the Reo network shown in Figure 4.5.2. The absence of data constraints in the CC, leads to incorrect behavior, as shown in the first row of the table, where there is flow on both b_1 and c .



43
Figure 4.5.2: A data-aware Reo connector

Table 4.5.3: Connector coloring semantics of the Reo network of Figure 4.5.2

a	b_1	b_2	c
—	—	—	—
—	—	—	\triangleright
—	\triangleright	\triangleright	\triangleright
\triangleright	\triangleright	\triangleright	\triangleright

4.6 Reo automata

Bonsangue et al. [BCS12] present *Reo automata* (RA), an automata-based formal model, to deal with context-dependency in Reo.

Intuitively, a Reo automaton is a non-deterministic automaton whose transitions are labeled in the form of $g|f$, where g is a binary predicate, called *guard*, and f a set of nodes that fire synchronously. A transition can be taken only when its guard g is true.

Let $\Sigma = \{\sigma_1, \dots, \sigma_k\}$ be a set of nodes, $\bar{\sigma}$ be the negation of σ , and \mathcal{B}_Σ be the free Boolean algebra generated by the following grammar:

$$g ::= \sigma \in \Sigma \mid \top \mid \perp \mid g \vee g \mid g \wedge g \mid \bar{g}$$

The above grammar produces *guards*. Often $g_1 \wedge g_2$ is written as $g_1 g_2$. A natural order \leq is defined between two guards $g_1, g_2 \in \mathcal{B}_\Sigma$ as

$$g_1 \leq g_2 \Rightarrow g_1 \wedge g_2 = g_1$$






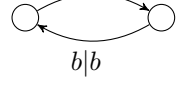

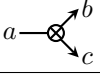


The intended interpretation of \leq is logical implication: $g_1 \Rightarrow g_2$. An atom of \mathcal{B}_Σ is a guard $a_1 \dots a_k$ such that $a_i \in \Sigma \cup \bar{\Sigma}$ with

$$\Sigma = \{\sigma_i \mid \sigma_i \in \Sigma\}, 1 \leq i \leq k$$

Definition 4.6.1 (Reo automaton [BCS12]) A Reo automaton is a triple (Σ, Q, δ) where:

- Σ is the set of nodes,
- Q is a set of states,

Table 4.6.1: Reo automata for basic Reo primitives

$ab ab$  RA corresponding to $a \bullet \longrightarrow \bullet b$	$ab ab$ $\bar{a}b a$  RA corresponding to $a \bullet \dashrightarrow \bullet b$	$ab ab$  RA corresponding to $a \bullet \longleftrightarrow \bullet b$
$ab ab$  RA corresponding to $a \longleftrightarrow \bullet b$	$\bar{a}b b$ $\bar{a}b a$  RA corresponding to $a \longleftrightarrow \vdash \bullet b$	$a a$  RA corresponding to $a \longleftrightarrow \boxed{} \longrightarrow \bullet b$
$ac ac$ $\bar{a}bc$  RA corresponding to 	$ac ac$ $bc bc$  RA corresponding to 	

- $\delta \subseteq Q \times \mathcal{B}_\Sigma \times 2^\Sigma \times Q$ is the transition relation such that for transitions labeled as $\mathcal{B}_\Sigma \times 2^\Sigma$ such that for each $q \xrightarrow{g|f} p \in \delta$:

$$\begin{aligned}
& - g \leq \hat{f} \\
& - g \leq g' \leq \hat{f}. \forall \alpha \leq g'. \exists q \xrightarrow{g''|f} p \in \Sigma. \alpha \leq g''
\end{aligned}$$

Table 4.6.1 depicts the Reo automata corresponding to the most common Reo elements.

4.7 Complexity

Analyzing the complexity of the calculations on CAP or other formal semantics of a Reo network in a formal fashion is beyond the scope of this dissertation. However, here we roughly estimate the time complexity of the product of CA. We have chosen CA because it is one of the most basic formal semantics for Reo. Calculating the complexity of CA product can provide an insight into the complexity of composing more sophisticated automata based semantics such as CAP.

Let R be a Reo network that is constructed by connecting n smaller networks in a step-wise fashion, meaning that one join occurs at a time, $\mathcal{A}_{1..i-1} = (Q_{1..i-1}, \mathcal{N}_{1..i-1}, \rightarrow_{1..i-1}, q_{0_{1..i-1}})$ be the CA of $R_{1..i-1}$ network at the i -th step before the i -th network is added, and $\mathcal{A}_i = (Q_i, \mathcal{N}_i, \rightarrow_i, q_{0_i})$ be the CA of R_i , the i -th network.

Note that at the first step, only A_1 exists. At the second step A_1 is connected to A_2 to form $A_{1..2}$.

Computing $\mathcal{A}_{1..i-1} \bowtie \mathcal{A}_i$ requires all transitions of $\mathcal{A}_{1..i-1}$, $t_{1..i-1}$, to be checked against the transitions of \mathcal{A}_i , t_i . For each t_i , the common ports of the transition and $\mathcal{N}_{1..i-1}$ need to be found. The time complexity of this operation is $O(T_{1..i-1} \times P_{1..i-1} \times P_i)$, where $T_{1..i-1}$ is the number of transitions of $\mathcal{A}_{1..i-1}$, $P_{1..i-1}$, and P_i are the number of elements in $\mathcal{N}_{1..i-1}$ and \mathcal{N}_i , respectively.

In addition, for the each $t_{1..i-1}$ all the common ports of the transition with \mathcal{N}_i is calculated. With a similar complexity of $O(T_i \times P_{1..i-1} \times P_i)$, where T_i is the number of transitions of \mathcal{A}_i .

Based on the outcome of these operations, we may need to create a couple of new states by merging the source and target states of $t_{1..i-1}$ and t_i . We assume that the creating these states takes a constant time. This assumption is based on the fact that constraint automata states are atomic entities.

However, in the case of CASM, the time complexity of creating a new state in the product of two CASMs depends on the number of state variables. Without considering transition guards, the complexity of computing $\mathcal{A}_{1..i}$ is:

$$O(T_{1..i-1} \times P_{1..i-1} \times P_i + T_i \times P_{1..i-1} \times P_i + T_{1..i-1} \times T_i) =$$

$$O\left(\prod_{j=1}^{i-1} T_j \times \prod_{k=1}^i P_k + \prod_{l=1}^i P_l \times T_i + \prod_{m=1}^i T_m\right)$$

Assuming that the number of transitions and the port names in each \mathcal{A}_i is \mathcal{T} and \mathcal{P} , respectively, the complexity can be written as $O(\mathcal{T}^n \times \mathcal{P}^n)$. As the formula shows the CA product is a very computationally expensive operation.

The problem of solving transition guards is a constraint satisfaction problem, which is a known NP-Complete problem. It is known that verifying a solution to an NP-complete problem is possible in polynomial time, but the time to find the solutions increases rapidly by the growth in the size of constraints.

Later in this dissertation, we provide an alternative approach for obtaining the formal semantics of a Reo network using constraint solvers. Our approach enables us to benefit from all the advances in research to keep this problem tractable for

practical use.

