Constraint-based analysis of business process models

Changizi, B.

7 Priority

7.1 Introduction

Priority is an important concept in modeling workflows. For instance, modeling compensation and error handling requires a mechanism to express priority of some flow alternatives over others. In the context of Reo, priority can be utilized as a mechanism to impose preferences on the otherwise non-deterministic choices.

Arbab et al. in [ABS15] introduce a compositional approach to model priority and a priority-aware formal semantics for Reo, named Constraint Automata with Priority (CAP), which is an extension of constraint automata.

This approach, which distinguishes between where priority is originated from and where it must be applied i.e. non-deterministic choices, consists of the following elements:

- A primitive to impose priority that is *prioritySync*,
- A mechanism to propagate priority from the location it is imposed through the network,

- A mechanism to block the propagation of priority in desired places using one of the following primitives:
  - *BlockSourceSync*, which stops propagation of priority coming from its source end toward its sink;
  - *BlockSinkSync*, which blocks propagation of priority from its sink end toward its source;
  - *BlockSync* that stops propagation of priority on both ends.
- Means to affect the otherwise non-deterministic choices by priority.

CAP is an expressive formalism for supporting priority in Reo. However, its operations to manipulate CAPs are computationally expensive, if they are implemented in a straight-forward fashion.

The practical needs for dealing with large models of realistic business processes currently complicates direct use of automata-based semantic models. In this chapter, we extend our constraint-based framework presented in Chapter 6 to support priority in Reo. The rest of this chapter is organized as follows: In Section 7.2, we introduce priority flow in Reo along with a constraint-based semantics for it. In Section 7.3, we extend our approach to support numeric priorities. In Section 7.4, we show the application of our constraint-based approach. In Section 7.5, we overview related work. Finally, in Section 7.6, we conclude the chapter and outline future work.

7.2 Priority flow

We distinguish between two types of priority on a node:

- when the node is imposing the priority to be propagated, which we call it *innate* priority,
- when the node has obtained the priority through propagation, we refer to it as *acquired*.

Both ends of *prioritySync* have *innate* priority. When an end with *innate* priority connects to another end that has no priority, the new end will obtain *acquired* priority. When one end of a synchronous type channel (e.g., *sync*, *syncDrain*) has *acquired* priority, the other end has *innate* priority.

However, in the case of non-synchronous channels (e.g., *FIFO*, *asyncDrain*) and also the priority blocking channels, their ends can only have *acquired* priority. We update the constraint-based framework for Reo presented in Chapter 6 to support priority and the priority propagation mechanism, which we informally described above. In the rest of this chapter, we omit data constraints when defining behavior of Reo elements. Data constraints are irrelevant for priority flow and were thoroughly covered in Chapter 6.

Let $\mathcal{N}$ and $\mathcal{M}$ be global sets of ends and state memory variables, respectively. A free variable v has one of the following forms, where $n \in \mathcal{N}$ and $m \in \mathcal{M}$:

- $\tilde{n} \in \{\top, \bot\}$ shows presence or absence of data-flow on n;
- $\mathring{m}, \mathring{m}' \in \{\top, \bot\}$ denotes whether or not the state memory variable m is defined in the source and the target states of the transition, respectively;
- $n^{\triangleright} \in \{\top, \bot\}$ indicates the reason for lack of data-flow on n originating from the primitive or the context (of this primitive), respectively;
- $n^{!\bullet}$, $n^{!\circ} \in \{\top, \bot\}$ models priority flow denoting whether n has *acquired* or *innate* priority. An end n has priority iff $n^{!\bullet} \vee n^{!\circ} = \top$.

A constraint Ψ, which encodes the behavior of a Reo network is defined as:

$$\begin{array}{lll} a & ::= & \tilde{n} \mid n^{!\bullet} \mid n^{!\circ} \mid n^{\triangleright} \mid \mathring{m} \mid \mathring{m}' \quad \text{(atoms)}, \\ \boldsymbol{\psi} & ::= & \top \mid a \mid \neg\boldsymbol{\psi} \mid \boldsymbol{\psi} \wedge \boldsymbol{\psi} \quad \text{(formulae)} \end{array}$$

A solution to $\boldsymbol{\psi}$ is a map from the variable sets V to a value in $\{\bot, \top\}$. The satisfaction rules for a solution $\langle\delta\rangle$ are satisfaction in propositional logic. We denote the set of all solutions for Ψ as $\mathfrak{S}(\Psi)$.

In Chapter 6 we have introduced RCSP. Here we extend the definition of RCSP and its composition operator with the priority notion and some axioms, which assist in incorporating priority in our constraint-based framework.

Definition 7.2.1 (RCSP) *A Reo Constraint Satisfaction Problem (RCSP) is a tuple $\langle \mathcal{N}, \mathcal{M}, M_0, \mathcal{V}, C \rangle$, where:*

- *$\mathcal{N}$ is a finite set of 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 network.*
- *$\mathcal{V}$ is a set of variables v defined by the grammar*

$$v ::= \tilde{n} \mid n^{\triangleright} \mid \mathring{m} \mid \mathring{m}' \mid n^{!\circ} \mid n^{!\bullet} \text{ for } n \in \mathcal{N} \text{ and } m \in \mathcal{M}.$$

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

Definition 7.2.2 (Composition $\odot$) *The composition of two RCSPs* $\rho_1 = \langle \mathcal{N}_1,\ \mathcal{M}_1,\ M_{0,1},\ \mathcal{V}_1,\ C_1 \rangle$ *and* $\rho_2 = \langle \mathcal{N}_2,\ \mathcal{M}_2,\ M_{0,2},\ \mathcal{V}_2,\ C_2 \rangle$ *is defined as follows:*

$$\rho_1 \odot \rho_2 = \langle \mathcal{N}_1 \cup \mathcal{N}_2,\ \mathcal{M}_1 \cup \mathcal{M}_2,\ M_{0,1} \cup M_{0,2},\ \mathcal{V}_1 \cup \mathcal{V}_2,\ C_1 \wedge C_2 \rangle.$$

Axiom 7.2.1 (Join axiom) *To propagate no-flow reasons, when a source end* c *and a sink end* k *from two networks, the following holds:*

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

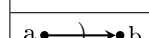
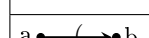
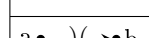
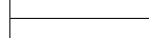
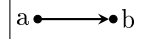
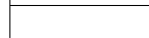
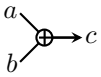
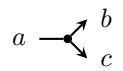
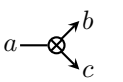
Axiom 7.2.2 (Priority join axiom) *When a source end* c *and a sink end* k *from two networks connect, this holds:*

$$(c^{!^{\circ}} \vee c^{!^{\bullet}} \Leftrightarrow k^{!^{\circ}} \vee k^{!^{\bullet}}) \wedge (c^{!^{\circ}} \wedge k^{!^{\circ}} \Leftrightarrow c^{!^{\bullet}} \vee k^{!^{\bullet}}).$$

Axiom 7.2.3 (Non-deterministic choice axiom) *Let* N *be a set of ends from which a Reo primitive chooses one for communication non-deterministically. The following guarantees that a node* y *with no priority has flow only if no prioritized node, e.g.,* x*, is ready to interact:*

$$(\neg\tilde{x} \wedge (x^{!^{\circ}} \ \vee \ x^{!^{\bullet}}) \wedge \tilde{y} \wedge \neg(y^{!^{\circ}} \ \vee \ y^{!^{\bullet}})) \Rightarrow \neg x^{\triangleright}.$$

Table 7.2.1: Constraint encoding of Reo with priority

Channel	Constraints
	$\psi_{PrioSync}(a,b) : (\tilde{a} \Leftrightarrow \tilde{b}) \wedge \neg(a^{\triangleright} \wedge b^{\triangleright}) \wedge a^{!\bullet} \wedge b^{!\bullet}$
	$\psi_{BlkSrcSync}(a,b) : (\tilde{a} \Leftrightarrow \tilde{b}) \wedge \neg(a^{\triangleright} \wedge b^{\triangleright}) \wedge \neg b^{!\bullet}$
	$\psi_{BlkSnkSync}(a,b) : (\tilde{a} \Leftrightarrow \tilde{b}) \wedge \neg(a^{\triangleright} \wedge b^{\triangleright}) \wedge \neg a^{!\bullet}$
	$\psi_{BlkSync}(a,b) : (\tilde{a} \Leftrightarrow \tilde{b}) \wedge \neg(a^{\triangleright} \wedge b^{\triangleright}) \wedge \neg a^{!\bullet} \wedge \neg b^{!\bullet}$
	$\psi_{Sync}(a,b) : (\tilde{a} \Leftrightarrow \tilde{b}) \wedge \neg(a^{\triangleright} \wedge b^{\triangleright}) \wedge ((\neg a^{!\bullet} \wedge \neg a^{!\circ} \wedge \neg b^{!\bullet} \wedge \neg b^{!\circ}) \vee (a^{!\bullet} \wedge \neg b^{!\bullet} \wedge b^{!\circ}) \vee (\neg a^{!\bullet} \wedge a^{!\circ} \wedge b^{!\bullet}))$
	$\psi_{LossySync}(a,b) : \tilde{b} \Rightarrow \tilde{a} \wedge \neg a^{\triangleright} \wedge \neg\tilde{a} \Rightarrow b^{\triangleright} \wedge ((\neg a^{!\bullet} \wedge \neg a^{!\circ} \wedge \neg b^{!\bullet} \wedge \neg b^{!\circ}) \vee (a^{!\bullet} \wedge \neg b^{!\bullet} \wedge b^{!\circ}) \vee (\neg a^{!\bullet} \wedge a^{!\circ} \wedge b^{!\bullet}))$
	$\psi_{SyncDrain}(a_1,a_2) : \tilde{a} \Leftrightarrow \tilde{b} \wedge \neg(a^{\triangleright} \wedge b^{\triangleright}) \wedge ((\neg a^{!\bullet} \wedge \neg a^{!\circ} \wedge \neg b^{!\bullet} \wedge \neg b^{!\circ}) \vee (a^{!\bullet} \wedge \neg b^{!\bullet} \wedge b^{!\circ}) \vee (\neg a^{!\bullet} \wedge a^{!\circ} \wedge b^{!\bullet}))$
	$\psi_{AsyncDrain}(a_1,a_2) : \tilde{a} \Rightarrow (\neg\tilde{b} \wedge b^{\triangleright}) \wedge \tilde{b} \Rightarrow (\neg\tilde{a} \wedge a^{\triangleright}) \wedge \neg a^{!\bullet} \wedge \neg b^{!\bullet}$
	$\psi_{FIFO_1}(a,b,m) : (\tilde{a} \Rightarrow \neg\mathring{m} \wedge \mathring{m}') \wedge (\tilde{b} \Rightarrow \mathring{m} \wedge \neg\mathring{m}') \wedge (\neg\tilde{a} \wedge \neg\tilde{b}) \Rightarrow (\mathring{m} \Leftrightarrow \mathring{m}') \wedge (\neg\mathring{m} \Rightarrow b^{\triangleright}) \wedge (\mathring{m} \Rightarrow a^{\triangleright}) \wedge (\neg a^{!\bullet} \wedge \neg b^{!\bullet})$
	$\psi_{Merger}(a,b,c) : (\tilde{a} \vee \tilde{b}) \Rightarrow \tilde{c} \wedge \neg(\tilde{a} \wedge \tilde{b}) \wedge \neg\tilde{c} \Rightarrow ((\neg c^{\triangleright} \wedge a^{\triangleright}) \vee (c^{\triangleright} \wedge \neg a^{\triangleright} \wedge b^{\triangleright}) \vee (c^{\triangleright} \wedge \neg b^{\triangleright} \wedge a^{\triangleright})) \wedge b^{\triangleright})) \wedge (c^{!\circ} \wedge \neg c^{!\bullet} \Rightarrow (a^{!\bullet} \wedge b^{!\bullet})) \wedge (\neg a^{!\bullet} \wedge b^{!\bullet} \wedge (a^{!\circ} \vee b^{!\circ}) \Rightarrow c^{!\bullet})$
	$\psi_{Replicator}(a,b,c) : \tilde{a} \Leftrightarrow (\tilde{b} \wedge \tilde{c}) \wedge \neg\tilde{a} \Rightarrow ((\neg a^{\triangleright} \wedge b^{\triangleright}) \vee (\neg b^{\triangleright} \wedge c^{\triangleright}) \vee (\neg c^{\triangleright} \wedge b^{\triangleright} \wedge a^{\triangleright})) \wedge c^{\triangleright} \wedge a^{\triangleright})) \wedge (a^{!\circ} \wedge \neg a^{!\bullet} \Rightarrow (b^{!\bullet} \wedge c^{!\bullet})) \wedge (\neg b^{!\bullet} \wedge c^{!\bullet} \wedge (b^{!\circ} \vee c^{!\circ}) \Rightarrow a^{!\bullet})$
	$\psi_{Router}(a,b,c) : \tilde{a} \Leftrightarrow (\tilde{b} \vee \tilde{c}) \wedge \neg(\tilde{b} \wedge \tilde{c}) \wedge \tilde{a} \Leftrightarrow (\neg a^{\triangleright} \vee \neg(b^{\triangleright} \vee c^{\triangleright})) \wedge (a^{!\circ} \wedge \neg a^{!\bullet} \Rightarrow (b^{!\bullet} \wedge c^{!\bullet})) \wedge (\neg b^{!\bullet} \wedge c^{!\bullet} \wedge (b^{!\circ} \vee c^{!\circ}) \Rightarrow a^{!\bullet})$

In Chapter 6, we presented the constraints that a primitive imposes on a network as a CSP. Here we extend these constraints with priority capturing variables.

If the variable $p^{!\bullet}$ is *true*, the end p has *innate* priority. For example, in a *prioritySync* channel, both ends have *innate* priority.

A primitive end can also obtain *innate* priority via propagation. For instance, if one end of a *sync* channel has *acquired* priority, which means it is prioritized because a primitive connected to it propagates priority, then the other end will have *innate* priority. We denote *acquired* priority for a primitive end p as: $p^{!\circ} \wedge \neg p^{!\bullet}$.

The priority capturing constraint for a *sync* channel with source end a and sink end b can be specified as follows:

$$\neg(a^{!\circ} \vee a^{!\bullet} \vee b^{!\circ} \vee b^{!\bullet}) \vee (a^{!\circ} \wedge \neg a^{!\bullet} \wedge b^{!\bullet}) \vee (a^{!\bullet} \wedge b^{!\circ} \wedge \neg b^{!\bullet}).$$

The assertion $\neg p^{!\bullet}$ blocks the priority propagation on p. Though, p can still have *acquired* priority through a potential connecting primitive when $p^{!\circ} = \top$.

Table 7.2.1 shows the constraint encoding of Reo channels and nodes in presence of priority flow. The solutions to the CSP expressing the behavior of a Reo network encode possible data-flow through its nodes.

Since a network may later connect to another network, the constraints should account for priority imposed by potential future connections. This information can be discarded when analyzing the behavior of a network in isolation. To exclude such cases, we should restrict the possible values of boundary ends.

Axiom 7.2.4 (Grounding axiom) *Let $B \subset N$ be the set of boundary nodes in a Reo network. We rule out the solutions that are only present for further expansion of the network by:*

$$\forall b \in B : b^{!\circ} \Rightarrow b^{!\bullet}.$$

Definition 7.2.3 (RLTS) *A Reo Labeled Transition System (RLTS) is a tuple $\mathcal{RLTS}$=($\mathcal{N}$, $\mathcal{M}$, Q, $\rightarrow$, q_0), where:*

- *$\mathcal{N}$ is a set of ends,*
- *$\mathcal{M}$ is a set of state memory variables,*
- *Q is a (finite) set of states of the form $\langle M \rangle$,*
- *M is the set of state memory variables that are valid in the given state, $\rightarrow \subseteq Q \times 2^{\mathcal{N}} \times 2^{\mathcal{N}} \times 2^{\mathcal{N}} \times Q$ is a transition relation, wherein N, R, and I in $(q,\ N,\ R,\ I,\ p) \in \rightarrow$ represent the ends that have flow, those without flow*

for which the reason for no flow is the end not being ready for interaction, and the ends with priority. Note that $n \notin N$ does not always mean $n \in R$ as the reason for data flow can be the network (then, n requires a reason for no flow).

- $q_0 \in Q$ *is the initial state.*

We write $q \xrightarrow{N,\ R,\ I} p$ *instead of* $(q,\ N,\ R,\ I,\ p) \in \rightarrow$. *For* $n \in I, n \notin R \Leftrightarrow n \in N$.

Definition 7.2.4 (Composition $\boxdot$) *We define the composition of* $L_1 = (\mathfrak{N}_1,\ \mathcal{M}_1,\ Q_1,\ \rightarrow_1,\ q_{0_1})$ *and* $L_2 = (\mathfrak{N}_2,\ \mathcal{M}_2,\ Q_2,\ \rightarrow_2,\ q_{0_2})$ *as:*

$$L_1 \boxdot L_2 = (\mathcal{N}_1 \cup \mathcal{N}_2,\ \mathcal{M}_1 \cup \mathcal{M}_2,\ \rightarrow,\ q_{0_1} \times q_{0_2})$$

where $\rightarrow$ *is defined as:*

$$\frac{q_1 \xrightarrow{N_1,R_1,I_1}{}_1 t_1\ q_2 \xrightarrow{N_2,R_2,I_2}{}_2 t_2 N_1 \cap \mathfrak{N}_2 = N_2 \cap \mathfrak{N}_1 R_1 \cap \mathfrak{N}_2 = R_2 \cap \mathfrak{N}_1 I_1 \cap \mathfrak{N}_2 = I_2 \cap \mathfrak{N}_1}{q_1 \times q_2 \xrightarrow{N_1 \cup N_2, R_1 \cup R_2, I_1 \cup I_2} t_1 \times t_2}$$

$$\frac{q_1 \xrightarrow{N_1,R_1,I_1}{}_1 t_1 q_2 \xrightarrow{N_2,R_2,I_2}{}_2 t_2 N_1 \cap \mathfrak{N}_2 = \emptyset}{q_1 \times q_2 \xrightarrow{N_1,R_1,I_1} t_1 \times t_2}$$

and its symmetric rule.

We define few operations on a solution s for $\Psi = \langle \mathcal{N}_\Psi,\ \mathcal{M}_\Psi,\ M_{\Psi 0},\ \mathcal{V}_\Psi,\ C_\Psi \rangle$:

- source(s)=$\langle \{m | m^\circ \in \mathcal{M}_\Psi : s(m^\circ) = \top\} \rangle$,
- target(s)=$\langle \{m | m'^\circ \in \mathcal{M}_\Psi : s(m'^\circ) = \top\} \rangle$,
- flow(s)=$\{n | n \in \mathcal{N}_\Psi :\ s(\tilde{n}) = \top\}$,
- reason-giving(s)=$\{n | n \in \mathcal{N}_\Psi : s(n^\triangleright) = \top\}$,
- priority(s)=$\{n | n \in \mathcal{N}_\Psi : (s(n^{!\circ}) \vee s(n^{!\bullet})) = \top\}$.

We say $s \backsim q \xrightarrow{N,R,I} p$, where

- q = source(s),
- N = flow(s),

- R=reason-giving(s),

- I = priority(s),

- p = target(s).

Definition 7.2.5 (Visualization) *The visualization function* γ *on* $\Psi = \langle \mathcal{N},\ \mathcal{M},\ M_0,\ \mathcal{V},\ C\rangle$ *yields* $\mathcal{L}=(\mathcal{N},\ \mathcal{M},\ Q,\ \rightarrow,\ q_0)$, *where*

- $\mathcal{M} = \{m | s(m^{\circ}) = \top \vee s(m'^{\circ}) = \top, s \in \mathfrak{S}(\Psi)\}$,
- $Q = \bigcup_{s\in\mathfrak{S}(\Psi)}\{source(s), target(s)\}$,
- $\rightarrow = \{(source(s),\ flow(s),\ reason\text{-}giving(s),\ priority(s),\ target(s)) \mid s \in \mathfrak{S}(\Psi)\}$,
- $q_0 = source(s_0)$.

Theorem 7.2.1 *Let* Ψ_1 *and* Ψ_2 *be two RCSPs, we show that* $\gamma(\Psi_1 \odot \Psi_2) = \gamma(\Psi_1) \boxdot \gamma(\Psi_2)$.

Proof Let $\gamma(\Psi_1)=(\mathfrak{N}_1, \mathcal{M}_1, Q_1,\ \rightarrow_1,\ q_{0_1})$, $\gamma(\Psi_2) = (\mathfrak{N}_2,\ \mathcal{M}_2, Q_2,\ \rightarrow_2,\ q_{0_2})$, and $\gamma(\Psi_1\ \odot\ \Psi_2) = (\mathfrak{N}, Q, \rightarrow,\ q_0)$.

It is trivial to see that $\mathfrak{N} = \mathfrak{N}_1 \cup \mathfrak{N}_2$, $\mathcal{M} = \mathcal{M}_1 \cup \mathcal{M}_2$, $Q = Q_1 \times Q_2$, $q_0 = q_{0_1} \times q_{0_2}$.

Assume $\exists s \in \mathfrak{S}(\Psi_1 \odot \Psi_2)$, $s_1, \in \mathfrak{S}_1$, $s_2 \in \mathfrak{S}_2$, $t_1 : q_1 \xrightarrow{N_1,R_1,I_1}{}_1 p_1$, $t_2 : q_2 \xrightarrow{N_2,R_2,I_2}{}_2 p_2$ s.t. $s_1 \backsim t_1$ and $s_2 \backsim t_2$, but $\nexists\ t : q \xrightarrow{N,R,I} p \in \rightarrow$ s.t. $s \backsim t$.

Therefore, $N_1 \cap \mathfrak{N}_2 \neq N_2 \cap \mathfrak{N}_1 \wedge N_1 \cap \mathfrak{N}_2 \neq \emptyset$ or $(N_1 \cup N_2) \cap (R_1 \cup R_2) \neq \emptyset$. The latter is impossible. For the former, either $n \in N_1, n \notin N_2$ or $n \in N_2, n \notin N_1$, which is not possible as it means $s(n) = \top \wedge s(n) = \bot$. Similarly, we can show it is impossible to have a t in $\gamma(\Psi_1 \odot \Psi_2)$, when there is no $s \in \mathfrak{S}$ s.t. $s \backsim t$.

RLTS is comparable with *Reo automata* [BCS12], a context-dependent formal semantics of Reo. A transition in *Reo automata* is labeled with a *guard*, which is a Boolean predicate in disjunctive normal form expressing positive and negative information about presence or absence of I/O requests, and a *firing* set that models the occurring I/O operations in the transition. The second set in RLTS transitions (the set of ends that provide reason for no flow) correspond to the negated elements of the guards in *Reo automata*, while the set of ends with flow relates to both the *firing* set and the positive elements of the guards. Unlike *Reo automata*, RLTS supports priority.

7.3 Numeric priority

Here, we extend our approach to support numeric priorities. This enables us to deal with more than one level of priorities such as in a process where the normal flow may be interrupted by both *exception* and *error*.

In BPMN, an *error* event has the highest priority, and the *exception* has priority over the normal flow. In this extension, the range for priority variables of an end n, $n^{!\circ}$ and $n^{!\bullet}$, is $\mathbb{N}$ (natural numbers) $\cup \{0\}$, where 0 indicates no priority. The larger number is the higher priority it represents. Each *prioritySync* channel comes with a user defined priority value, which propagates through its ends. To propagation of a higher priority over a lower priority or no priority, we constrain priority variables to be greater than or equal to their initial values.

$\langle\delta\rangle \vDash x \geq P$ iff $\delta(x) \geq P$, $\langle\delta\rangle \vDash x > P$ iff $\delta(x) > P$, $\langle\delta\rangle \vDash x = P$ iff $\delta(x) = P$, where $x \in \{x^{!\bullet}, x^{!\circ}\}, P \in \mathbb{N} \cup \{0\}$.

The new constraint-based encodings of the *replicator* and *router* nodes in this table are constructed in accordance with Axiom 7.2.3.

Definition 7.3.1 (NPRLTS) *A Numeric Priority Reo Labeled Transition System is a tuple* $(\mathcal{N},\ \mathcal{M},\ Q,\ \rightarrow,\ q_0)$*, where:*

- $\mathcal{N}$ *is a set of ends,*
- $\mathcal{M}$ *is a set of state memory variables,* Q *is a (finite) set of states of the form* $\langle M\rangle$*,* M *is the set of state memory variables that are valid in the given state,* $\rightarrow \subseteq Q \times 2^{\mathcal{N}} \times 2^{\mathcal{N}} \times \mathcal{N} \mapsto \mathbb{N} \times Q$ *is a transition relation, wherein* N*,* R*, and* f_I *in* $(q,\ N,\ R,\ f_I,\ p) \in \rightarrow$ *are the ends having flow, those without flow for which the reason for no flow is the end not being ready for interaction, and a partial map of nodes with priority to their priority values, respectively.*
- $q_0 \in Q$ *is the initial state.*

We write $q \xrightarrow{N,R,f_I} p$ *instead of* $(q,\ N,\ R,\ f_I,\ p) \in \rightarrow$*. For all* $q \xrightarrow{N,R,f_I} p$*:* $f(n) > 0, n \notin N \Leftrightarrow n \in R$*. We redefine priority*$(s)$ *as* $\{(n,p) | n \in \mathcal{N}_\Psi : s(n^{!\circ}) = p \vee s(n^{!\bullet}) = p\}$.

Definition 7.3.2 (Extended Visualization) *The visualization function* γ *on* $\Psi = \langle \mathcal{N}_\Psi,\ \mathcal{M}_\Psi,\ M_{\Psi_0},\ \mathcal{V},\ C\rangle$ *yields* $\mathcal{L} = (\mathcal{N}_L,\ \mathcal{M}_L,\ Q,\ \rightarrow,\ q_0)$*, where*

- $\mathcal{N}_L = \{n | s(\tilde{n}) = \top,\ s \in \mathfrak{S}(\Psi)\}$,
- $\mathcal{M}_L = \{m | s(m^\circ) = \top\ \vee\ s(m'^\circ) = \top,\ s \in \mathfrak{S}(\Psi)\}$,

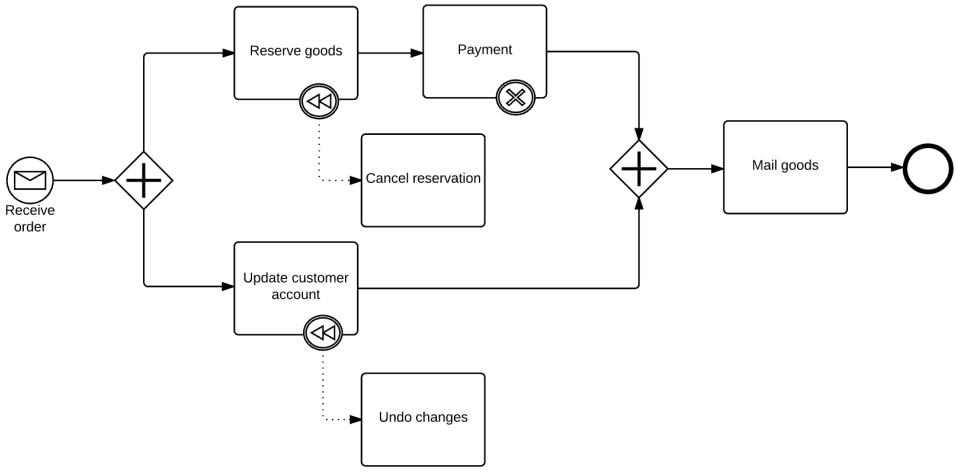


Figure 7.4.1: An example of a sales process modeled in BPMN

- $Q = \bigcup_{s \in \mathfrak{S}(\Psi)} \{source(s),\ target(s)\}$,
- $\rightarrow = \{(source(s),\ flow(s),\ reason\text{-}giving(s),\ priority(s),\ target(s)) \mid s \in \mathfrak{S}(\Psi)\}$, $q_0 = source(s_0)$.

7.4 Case study

In this section, we present the applications of our approach on a priority-aware model. Figure 7.4.1 depicts a sales process, which starts by receiving an order from a customer. It proceeds by reserving the ordered items for the customer. Then, the customer's credit gets charged and the customer's account is updated, meanwhile if the payment encounters a problem, a *cancellation* event is triggered, which causes compensation for any of the performed actions. Finally, if no problem occurs, the ordered items are shipped and the process ends.

Figure 7.4.2 shows a Reo network that simulates this process. Here, we use alphabet characters to refer to nodes (e.g. B, C) and channels (e.g. BC, BD). To address a node end or a channel end, we append a number to the name of an end, unless it is the only end (e.g. it is a boundary end). For instance, the end BC_2, which is the source end of the channel BC connects to the end B_2 on the node B. In [CKA10], the authors defined a procedure to map BPMN models to Reo networks.

The process starts by reading a token from the *writer* W_2, which resembles

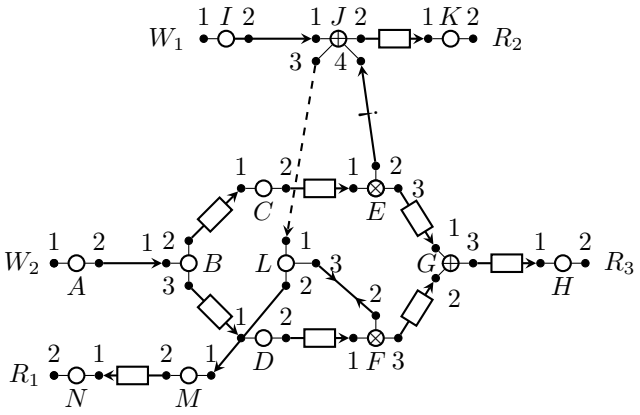


Figure 7.4.2: The process of a sample on-line shop modeled in Reo

receiving an order. Though a Reo network can be used for modeling infinite data flow, in the BPMN standard, when a *start* event is triggered, a new instance of the process is instantiated. Therefore, the Reo network is designed to handle only one request. The end A_1 reads a token from the writer W_2 and duplicates it into the BC and BD $FIFO_1$ channels. The token continues to the CE $FIFO_1$ channel. If the payment succeeds, the token enters the EG $FIFO_1$ channel waiting for a token from the other input of the *merge* node G to enter the GH $FIFO_1$ channel and finally to be consumed by the *reader* R_3.

If the payment fails, performed actions need to be compensated. A token from the writer W_1 indicates a payment failure, so the process needs to be canceled. So, the token leaving the CE $FIFO_1$ channel goes through the EJ *prioritySync* channel. The *replicate* node J duplicates the token to the JK $FIFO_1$ and the JL *lossySync* channel. The *reader* R_2 consumes the token from the JK $FIFO_1$ channel, while the token from the JL *lossySync* channel moves forward to the MN $FIFO_1$ channel.

The token from the BD $FIFO_1$ channel goes through the DF $FIFO_1$ channel for a possible compensation. The token from the DF $FIFO_1$ channel may either go to the *join* node G to join the flow of a successful payment, or to be consumed by the LF *syncDrain*. In the latter case, it goes to the MN $FIFO_1$ channel. Then, the process ends by a read action of the *reader* R_1.

We compute the behavior of the given Reo network using our constraint-based framework. The steps for obtaining the RLTS are as follows: First, we form the RCSP of the network by traversing through its primitives. Then, we solve the obtained RCSP and extract transitions from obtained solutions.

To show how priority can affect the behavior of our example, we first investigate

the behavior of the network in absence of priority, wherein the normal flow of the process can continue even in case of a payment failure. This is because the *router* nodes E chooses one of its outgoing flows in a non-deterministic fashion.

We would like to check if for all transitions t, which belong to the RLTS of the network, the following holds: $\{CE,\ DF\} \subseteq source(t) \wedge E_1 \in flow(t) \wedge W_1 \notin reason-giving(t) \Rightarrow W_1 \in target(t)$. To violate this property, it is enough to find a transition from a state wherein both CE and DF $FIFO_1$ channels are full, there is flow on end E_1, W_1 is ready to communicate, but W_1 does not have flow.

Abstraction: For checking this assertion, we abstract from the ends without flow on transitions with the same source (q), target (p), ends with flow (N_1), but different ends without flow (N_2) by replacing them with $q \xrightarrow{N_1,N'_2} p$, where $N'_2 = \{W_1\}$ if $W_1 \in N_2$, otherwise $N'_2 = \{\}$. This abstraction reduces the number of transitions in the RLTS without affecting the result of the verification for the given assertion. We can take this one step further and remove the information about ends without flow from all the states except the state wherein CE and DF $FIFO_1$ channels are full.

Figure 7.4.3 shows the abstract (with respect to the given assertion) RLTS of the network of Figure 7.4.2 in absence of priority, where the transition t_4 violates the assertion. Here, we use short labels (e.g. t_4) on transitions and states. The original labels are represented in Table 7.4.1. In addition, the ends with a similar name are grouped e.g. $B_{1,2,3}$ (referring to ends B_1, B_2, and $B3$). This is only a presentation modification to save space. We show that the transition t_4 can not exist when the priority is considered in the model.

$$0 : \mathring{CE} \wedge \mathring{DF} \wedge \tilde{E}_1 \wedge \neg {W_1}^{\triangleright} \wedge \neg \tilde{W}_1 \ (the\ assertion)$$

$$1 : \frac{\Psi_{PrioritySync}(EJ_{2,4})}{EF_2^{!\bullet}}$$

$$2 : \frac{1\ \&\text{join of } EJ_2 \& E_2}{E_2^{!\bullet}}$$

$$3 : \frac{2; \Psi_{router}(E_{1,2,3})}{\tilde{E}_1 \wedge \neg \tilde{E}_2 \Rightarrow {E_2}^{\triangleright}}$$

$$4 : \frac{3\ \&\ \text{coloring}\ \&\ \text{join}}{{E_2}^{\triangleright} \Rightarrow {W_1}^{\triangleright}}$$

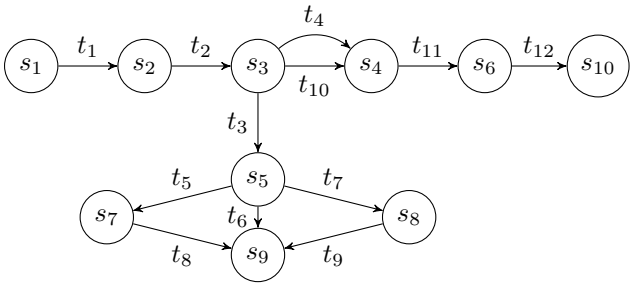


Figure 7.4.3: The RLTS corresponding to Reo network of Figure 7.4.2 with no priority channel

$$5 : \frac{2 \text{ \& } 4 \text{ coloring \& join}}{\neg \tilde{W}_1 \Rightarrow {W_1}^{\triangleright}}$$

$$6 : \frac{0\&5}{\bot}$$

7.5 Related work

Several works, e.g., [FPHA02, BK92, Bau97] use priorities to model scheduling policies. Many workflow languages rely on Petri nets [vdAtH02, YSSW08]. Priority flow in Petri net-based process models is managed with the help of inhibitor arcs and transition priorities [Pad15]. Inhibitor arcs allow a transition to fire only if the adjacent place is empty. *Prioritized Petri nets* [Bal01] introduce a partial order on transitions. Given a set of enabled transitions, the transitions with higher priority fire before the transitions with lower priority. Others, e.g., [LP16, RMP+12] use a partial order on transitions to model priority. Our earlier approach in modeling priority using binary variables supports a limited form of priority compared to the mentioned Petri nets approaches. However, the proposed extension bridges this gap by defining priorities as non-zero natural numbers. An advantage of our model is its compositionality. Compared to the aforementioned methods, Reo fits in the realm of component-based or service-oriented architecture in a compositional way. Reo is an extensible language, where new behavioral aspects can be added. An effort to express the behavior of Reo networks via constraints is reported in [CPLA10]. It demonstrates the efficiency of the constraint-based approach. It models synchronization and data flow constraints, but no priority flow was considered. In [CKA12],

Table 7.4.1: The transition labels and prioritized ends (P) of the RLTS of Figure 7.4.3

s_1	$\langle\rangle$
s_2	$\langle BC, BD\rangle$
s_3	$\langle CE, DF\rangle$
s_4	$\langle EG, FG\rangle$
s_5	$\langle MN, JK\rangle$
s_6	$\langle GH\rangle$
s_7	$\langle JK\rangle$
s_8	$\langle MN\rangle$
s_9	$\langle\rangle$
s_{10}	$\langle\rangle$
t_1	$N_1 : \{W_2, A_{1,2}, B_{1,2,3}, AB_{1,2}, BC_2, BD_3\}, N_2 : \{\}$
t_2	$N_1 : \{BC_1, BD_1, C_{1,2}, D_{1,2}, CE_2, DF_2\}, N_2 : \{\}$
t_3	$N_1 : \{W_1, CE_2, DF_3, IJ_{2,3}, I_{1,2}, J_{1,2,3,4}, JK_2, JL_{1,3},$ $L_{1,2,3}, LF_{2,3}, LM_{1,2}, F_{1,2}, M_{1,2}, MN_2\}, N_2 : \{\}$
t_4	$N_1 : \{EG_3, FG_3, E_{1,3}, F_{1,3}, CE_1, DF_1\},$ $N_2 : \{W_1\}$
t_5	$N_1 : \{R_1, N_{1,2}, MN_1\}, N_2 : \{\}$
t_6	$N_1 : \{R_{1,2}, N_{1,2}, MN_1, K_{1,2}, JK_1\}, N_2 : \{\}$
t_7	$N_1 : \{R_2, K_{1,2}, JK_1\}, N_2 : \{\}$
t_8	$N_1 : \{R_2, K_{1,2}, JK_1\}, N_2 : \{\}$
t_9	$N_1 : \{R_1, N_{1,2}, MN_1\}, N_2 : \{\}$
t_{10}	$N_1 : \{EG_3, FG_3, E_{1,3}, F_{1,3}, CE_1, DF_1\},$ $N_2 : \{W_1\}$
t_{11}	$N_1 : \{EG_1, FG_2, G_{1,2,3}, GH_3\}, N_2 : \{\}$
t_{12}	$N_1 : \{R_3, H_{1,2}, GH_1\}, N_2 : \{\}$
P	$\{W_1, I_{1,2}, J_{1,2,3,4}, JK_2, EJ_{2,4}, E_{1,2}, JL_{1,3}, L_{1,2,3},$ $LF_{2,3}, F_{2,3}, LM_{1,2}, M_{1,2}, MN_2\}$

a framework is presented to encode semantics of Reo networks as CSP with predicates in the form of binary propositions and numerical constraints. An advantage of this method is handling data constraints symbolically and, hence, mitigating the state explosion problem of automata models. We extended this framework to handle priority constraints, taking a step forward toward implementing a tool-set that covers all behavioral aspects of Reo. Among the formal semantics of Reo, connector coloring comes with a limited notion of priority based on the context information. The context information affects otherwise non-deterministic data-flow choices. In [KAT16], an automata-based semantics is proposed, which associates a preference for each transitions. A transition of lower preference is fired iff no more preferred transition can occur.

7.6 Conclusions and future work

In this chapter, we addressed the problem of priority flow modelling using the Reo coordination language. We extended the unified constraint-based semantics of Reo with binary and numeric priority constraints. Furthermore, we showed correctness of our approach for the binary case. We also illustrated the use of our framework for modeling business processes with priority flow.

As part of our ongoing work, we are using this framework to encode other aspects of the semantics of Reo, specifically, timed behavior. A promising area for future work is to use our framework for constraint-based model checking of Reo networks with priority.