A model of context-dependent component connectors

Marcello Bonsangue\textsuperscript{a,c}, Dave Clarke\textsuperscript{b,}\textsuperscript{*}, Alexandra Silva\textsuperscript{c}

\textsuperscript{a} LIACS, Leiden University, The Netherlands
\textsuperscript{b} Department of Computer Science, Katholieke Universiteit Leuven, Belgium
\textsuperscript{c} CWI, The Netherlands

\begin{abstract}
Recent approaches to component-based software engineering employ coordinating connectors to compose components into software systems. For maximum flexibility and reusability, such connectors can themselves be composed, resulting in an expressive calculus of connectors whose semantics encompasses complex combinations of synchronisation, mutual exclusion, non-deterministic choice and state-dependent behaviour.

To increase the expressiveness of connectors, notions of context-dependent behaviour have been proposed. Context dependency can be used to express the priority of one behaviour over another and the inhibition of actions due to changing context. The notion of context we consider in this paper is given by the pending activities on the ports of a connector. Context-dependent behaviour occurs whenever the choices available to a connector change non-monotonically as its context changes.

Capturing context-dependent behaviour in formal models is non-trivial, as it is unclear how to propagate context information through composition. In this paper we present an intuitive automata-based formal model of context-dependent connectors, and argue that it is superior to previous attempts at such a model for the coordination language Reo.
\end{abstract}

\section{Introduction}

The holy grail of component-based software engineering is to develop truly reusable software components that can be sold off-the-shelf and reused to build software systems [38]. Research on software composition plays a key role in this quest, as it offers flexible ways of plugging together components. Some approaches to software composition use textual glue code [33,20,35], usually in a scripting language, whereas others offer a more visual approach, where 'channels' or 'connectors' are used to compose components into a system [9,22,1,18].

Connectors play the role of coordinating software systems, yet their functionality is traditionally more limited than scripting languages. This trend has been reversed with investigation into the notion of compositional connectors [1,33]. In such a setting, connectors are formed by composing simpler connectors such as channels together. These 'languages' express various coordination patterns exhibiting combinations of synchronisation, mutual exclusion, non-deterministic choice, and state-dependent behaviour. A number of component connector models exist, including Reo [1], Ptolemy [30], Ptolemy II [31], MoCha [22], Manifold [5], BIP [10], an algebra of stateless connectors [12], and pipe and filter architectures [37]. Although these overlap in philosophy and functionality, only BIP, the algebra of stateless connectors, and Reo enable synchrony and mutual exclusion to propagate through connectors—this means that synchronisation and mutual exclusion constraint between 'ports' of a connector are combined conjunctively when ports are plugged together. Of these, only Reo offers state-dependent behaviour.

\textsuperscript{*} Corresponding author.
\textit{E-mail address: dave.clarke@cs.kuleuven.be} (D. Clarke).

0167-6423/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2011.01.006
The trend is to increase (or improve) the expressiveness of such coordination models by investigating features such as dynamic reconfiguration [26], data-sensitive operations such as data filtering and transformation [14], and context-dependent behaviour [13]. The latter feature is characterised by behaviours which depend upon both the positive and negative occurrences of I/O requests on the boundary ports of the connector. This paper follows this trend, by investigating the notion of context dependency in the setting of the coordination language Reo [1]. Context dependency enables connectors to be more responsive to changes in their environment, and thus increases the expressiveness of connectors enabling them to express, for example, priority and inhibition. Our primary goal is twofold, namely to produce a model of context-dependent connectors which avoids a number of the problems of previous models for Reo, in a manner which can be implemented efficiently. Our work nonetheless is applicable beyond the realm of Reo semantics, such as providing solid guidelines and foundations for future context-dependent extensions of the component connector models mentioned above.

Context-dependent behaviour has already been studied in the context of non-monotonic concurrent constraint programming [17] and generative communication [21], where operators are defined with the ability of observing the absence of data. The extra difficulty present in connector-based models is how to propagate context-dependent behaviour properly.

Contributions. This paper presents a compositional automata model for expressing context-dependent connectors. Following intentional automata [16], the model expresses context dependency by modelling both the I/O requests from the environment and the firings of the connector. It is a simple and intuitive model, in the sense that automata corresponding to basic connectors have a small number of states and transitions, compared to intentional automata. Moreover, because our automata are partial, the model overcomes a problem with totality preservation present in connector colouring [13], the original attempt to add context dependency to Reo.

Connector plugging is achieved by a novel two-step composition operation consisting of a product, modelling the independent execution of distinct connectors, plus a synchronisation operation. Composition propagates context information, which contains both positive and negative information, through the connector. Using this we define a previously elusive notion of enabledness and show that it is also appropriately propagated through composition. We also formally define the notion of context dependency, which had never been formalised for any of the other existing models of Reo. The presented automata model also enables an efficient implementation of context-dependent Reo connectors, combining the benefit of previous automata-based implementations [32] with the context dependency originally developed in the connector colouring model [13]. In addition, we extend the notion of context-dependent automata to include the modelling of data flow, as in constraint automata, and we present a final semantics for our automata model in terms of guarded strings.

This paper extends our Coordination 2009 article [11] with complete proofs, more examples, and new results on enabledness and the final trace semantics for Reo.

Organisation. Section 2 describes the Reo coordination language and highlights problems with its models with regard to context dependency. Section 3 describes guarded strings, the formal basis for traces of context-dependent connectors. Section 4 describes guarded automata, the basis of our formalism, along with its product and synchronisation operations, and the additional conditions required for modelling Reo connectors. Section 5 presents two concrete models of guarded automata, extending both port automata [27] and constraint automata [7] with context dependency. Section 6 describes and justifies various technical conditions present in our model via theorems and counter-examples. Section 7 presents the final semantics of our automata model, Section 8 provides a detailed discussion of existing models of Reo and Section 9 concludes.

2. The coordination language Reo

Reo [1] is a model of component coordination wherein component connectors are constructed by composing more primitive connectors, such as channels, data replicators, stream mergers and routers. Primitives express state-dependent synchronisation and mutual exclusion constraints on their ports, along with the data flow between the ports that synchronise. Primitives can exhibit different behaviours in terms of synchronisation and mutual exclusion of their ports, the direction of data flow, the presence of buffering, state, and whether or not data can be lost. Composition of connectors is achieved by plugging ports together (one-to-one, in the direction of data flow, is sufficient). Composition imposes the constraint that the two ports plugged together synchronise, and thereby synchronisation and mutual exclusion constraints propagate through a connector.

A number of Reo’s primitive connectors are depicted in Fig. 1. These form quite an expressive set of connectors (most connectors appearing in the literature use these or their close relatives). Their semantics are presented later in Fig. 4.

The interaction model presupposed by Reo is that components try to write or take data from the ports they are connected to. The connector then determines when the write or take 'fires', together with passing data along through the channels of the connector. The notion of synchrony is equated with the ports that fire together, and mutual exclusion is when ports cannot fire together. Early formal models of Reo [7,6] express only the sets of write/take actions which can fire together, dubbed as firing. Context-dependent behaviour goes beyond this: such behaviour differs depending upon both the positive and negative occurrences of I/O requests on the boundary ports of the connector. Using this request information as well, connectors can express a notion of priority, when two or more choices are possible, and a notion of inhibition wherein attempts by the components to perform operations block (certain) firings from occurring.
Informal accounts of Reo give a localised description of the context-dependent nature of certain connectors. For instance, the LossySync channel (with ports \(a\) and \(b\)) has the behaviour that if a write request and a take request are present on \(a\) and \(b\), respectively, then data flows from \(a\) to \(b\) (synchronously). If, however, no take on \(b\) is present, then data may flow at \(a\), but it is lost in the channel. In contrast, the Sync channel (with ports \(a\) and \(b\)) is not context dependent: data may only flow synchronously. In fact, we will show in what follows that this channel behaves as a kind of identity when composed with other channels. Notions of priority can also be described in this fashion, by using the context (boundary I/O requests) to break any non-determinism.

The problem with this kind of description, first identified by Clarke et al. [13], is that it relies on the presence of requests on the ports of primitives, but after composition these ports are generally no longer on the boundary of a connector, but have been made internal, and informal accounts do not provide a precise enough description of how context-dependent behaviour propagates through composition. This is a consequence of the impedance mismatch resulting from plugging together two ports: both ports are expecting some environment to initiate interaction, but the environment (some component) is not present at the point where two ports are joined. Although Arbab [1] describes how offers of data (writes) and willingness to accept data (takes) propagate through channels, unfortunately this description is incomplete and imprecise, in particular with regard to how context propagation interacts with non-deterministic choice.

We illustrate next by means of an example how our model, which we call Reo automata, overcomes the problems existing in previous models of Reo. In order to do so we show the contrast between the semantics proposed in this paper and the first semantic model proposed for Reo, namely, constraint automata [7]. Although the technical details might not be fully clear at this stage, we shall intuitively explain what each model attempts to encode and the consequences in the composition of channels.

In the constraint automaton corresponding to a channel the labels in the transitions contain only information about which ports fire at a certain moment. Take, for instance, the automaton in the second row, corresponding to the FIFO1 channel: the FIFO1 changes state from empty \((e)\) to full \((f)\) in case port \(c\) fires. In the case of the LossySync, two transitions are possible: either the data is lost, in which case only port \(a\) fires, or data is passed along the channel, in which case both ports \(a\) and \(b\) fire.

Explained locally, the semantics provided by constraint automata seems to be correct. It is when composing the channels that the problem arises. Looking at the constraint automaton in the third row, which would be the semantics of the composition of a LossySync and a FIFO1, there is a transition labelled by \(\{a\}\) in the state \((e, q)\), which denotes an empty FIFO1. This means that even when the FIFO1 is empty data can be lost, which is not the intended semantics. The problem arises from the fact that the constraint automaton corresponding to LossySync allows data to be lost independently of the context, or more precisely, independently of the presence or absence of a request on port \(b\). We overcome this problem in Reo automata.
by explicitly modelling both the presence and absence of requests on ports. Take the Reo automaton corresponding to the LossySync channel. The intended reading of a label $g \mid f$ of a transition is as follows: the ports $f$ fire given that the conditions encoded in guard $g$ hold. In the aforementioned automaton this means that the transition labelled by $a\overline{b}\overline{c}$ captures that data can only be lost in case the request on $b$ is absent (denoted by $\overline{b}$). This extra information present in the Reo automata model enables the definition of a composition operator which correctly gives semantics to context-dependent connectors, such as the LossySync - FIFO1 presented in the third row of the above table. Note how in the Reo automaton of the third row the only transition possible when the FIFO1 is empty is the one leading to a full FIFO1.

Further problems potentially occur when these two connectors would be composed within a larger connector, such as in the Discriminator depicted in Fig. 2. The context/request information needed to ensure that FIFO1 and the LossySync behave correctly when composed needs to be maintained and propagated in the model of the larger connectors. In technical terms, to be dealt with later in the paper, the enabledness of the empty FIFO1 and the context dependency of the LossySync need to be propagated through the model of the connector. The context information needs to be handled carefully in order to ensure that the resulting model of the connector is the same irrespective of the order in which its constituent channels are composed.

Many models have been proposed for Reo. A complete discussion of these is deferred until Section 8, in order to provide a better comparison with the model proposed in this paper. A new model of Reo is presented in this paper to provide a satisfactory account of context dependency. This paper provides clarity in this issue by presenting not only a new model for Reo, but also formal conditions justifying various choices. These conditions can serve as a baseline for future models of Reo and its successors.

The definition of context dependency we provide is based on the notion of firing monotonicity (Definition 6.4), which captures that, when a connector is able to fire, additional requests on its boundary will not invalidate that firing. A connector is context dependent whenever it is not firing monotonic (Definition 6.6). This captures the notion that firings are dependent on the context given by both the presence and absence of requests on the connector’s ports. Additional properties are imposed, such as reactivity and uniformity (Definition 4.7), which limit the range of possible context-dependent behaviours, but ensure that the above mentioned properties, namely, that the order of composition does not matter and propagation of enabledness holds (see as Lemmas 4.13 and 6.9).

3. Preliminaries: guarded strings

Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ and $\mathcal{B}_\Sigma$ be the free Boolean algebra generated by the following grammar:

\[
g ::= \sigma \in \Sigma \mid \top \mid \bot \mid g \lor g \mid g \land g \mid \overline{g}
\]

$\top$ and $\bot$ represent the top and bottom elements of the lattice (sometimes known as true and false).

We refer to the elements of the above grammar as guards and in their representation we frequently omit $\land$ and write $g_1 g_2$ instead of $g_1 \land g_2$. Given two guards $g_1, g_2 \in \mathcal{B}_\Sigma$, define a (natural) order $\leq$ by putting $g_1 \leq g_2 \iff g_1 \land g_2 = g_1$. The intended interpretation of $\leq$ is logical implication $g_1$ implies $g_2$.

Given a guard $g$ there exists an equivalent guard $\text{norm}(g)$ of the form $\bigvee \bigwedge a$, where $a \in \Sigma \cup \overline{\Sigma}$, with $\overline{\Sigma} = \{\sigma \mid \sigma \in \Sigma\}$, and $\bigvee$ and $\bigwedge$ the extensions of $\lor$ and $\land$, respectively, to sets of guards. The guard $\text{norm}(g)$ is usually called the disjunctive normal form of $g$. Since $\text{norm}(g)$ can be written as a disjunction, we use the notation $g' \in \text{norm}(g)$ to refer to an arbitrary disjunct of $\text{norm}(g)$.

An atom of $\mathcal{B}_\Sigma$ is a guard $a_1 \ldots a_k$ such that $a_i \in \{\sigma_i, \overline{\sigma_i}\}$, for $1 \leq i \leq k$. We can think of an atom as a truth assignment. Denote atoms by Greek letters $\alpha, \beta, \ldots$ and the set of all atoms of $\mathcal{B}_\Sigma$ by $\text{At}_\Sigma$. Every element of a finite Boolean algebra can

---

**Fig. 2.** Discriminator Connector. This selects input from one of ports $a$, $b$, and $c$. After all of $a$, $b$, and $c$ have provided input, port $d$ produces the originally selected datum, and the connector returns to its original state. In this connector, exactly one of the LossySync $ax$, $bx$, $cx$ must succeed to fill FIFO1 buffers connected at $x$. 

Further problems potentially occur when these two connectors would be composed within a larger connector, such as in the Discriminator depicted in Fig. 2. The context/request information needed to ensure that FIFO1 and the LossySync behave correctly when composed needs to be maintained and propagated in the model of the larger connectors. In technical terms, to be dealt with later in the paper, the enabledness of the empty FIFO1 and the context dependency of the LossySync need to be propagated through the model of the connector. The context information needs to be handled carefully in order to ensure that the resulting model of the connector is the same irrespective of the order in which its constituent channels are composed.

Many models have been proposed for Reo. A complete discussion of these is deferred until Section 8, in order to provide a better comparison with the model proposed in this paper. A new model of Reo is presented in this paper to provide a satisfactory account of context dependency. This paper provides clarity in this issue by presenting not only a new model for Reo, but also formal conditions justifying various choices. These conditions can serve as a baseline for future models of Reo and its successors.

The definition of context dependency we provide is based on the notion of firing monotonicity (Definition 6.4), which captures that, when a connector is able to fire, additional requests on its boundary will not invalidate that firing. A connector is context dependent whenever it is not firing monotonic (Definition 6.6). This captures the notion that firings are dependent on the context given by both the presence and absence of requests on the connector’s ports. Additional properties are imposed, such as reactivity and uniformity (Definition 4.7), which limit the range of possible context-dependent behaviours, but ensure that the above mentioned properties, namely, that the order of composition does not matter and propagation of enabledness holds (see as Lemmas 4.13 and 6.9).

3. Preliminaries: guarded strings

Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ and $\mathcal{B}_\Sigma$ be the free Boolean algebra generated by the following grammar:

\[
g ::= \sigma \in \Sigma \mid \top \mid \bot \mid g \lor g \mid g \land g \mid \overline{g}
\]

$\top$ and $\bot$ represent the top and bottom elements of the lattice (sometimes known as true and false).

We refer to the elements of the above grammar as guards and in their representation we frequently omit $\land$ and write $g_1 g_2$ instead of $g_1 \land g_2$. Given two guards $g_1, g_2 \in \mathcal{B}_\Sigma$, define a (natural) order $\leq$ by putting $g_1 \leq g_2 \iff g_1 \land g_2 = g_1$. The intended interpretation of $\leq$ is logical implication $g_1$ implies $g_2$.

Given a guard $g$ there exists an equivalent guard $\text{norm}(g)$ of the form $\bigvee \bigwedge a$, where $a \in \Sigma \cup \overline{\Sigma}$, with $\overline{\Sigma} = \{\sigma \mid \sigma \in \Sigma\}$, and $\bigvee$ and $\bigwedge$ the extensions of $\lor$ and $\land$, respectively, to sets of guards. The guard $\text{norm}(g)$ is usually called the disjunctive normal form of $g$. Since $\text{norm}(g)$ can be written as a disjunction, we use the notation $g' \in \text{norm}(g)$ to refer to an arbitrary disjunct of $\text{norm}(g)$.

An atom of $\mathcal{B}_\Sigma$ is a guard $a_1 \ldots a_k$ such that $a_i \in \{\sigma_i, \overline{\sigma_i}\}$, for $1 \leq i \leq k$. We can think of an atom as a truth assignment. Denote atoms by Greek letters $\alpha, \beta, \ldots$ and the set of all atoms of $\mathcal{B}_\Sigma$ by $\text{At}_\Sigma$. Every element of a finite Boolean algebra can
be written as a disjunction of atoms. Given $S \subseteq \Sigma$, define $\bar{S} \in \mathcal{B}_\Sigma$ as the conjunction of all elements of $S$. For instance, for $S = \{a, b, c\}$ one has $\bar{S} = abc$. Define the atom associated with a set $S$ in the expected way—$\alpha_S = \bar{S} \wedge \bar{S} \setminus S$. For example, if $S = \{a, b, c\}$, then $\alpha_{\{a,b\}} = \bar{abc}$. Conversely, the set associated with an atom $\alpha$ is defined as $\alpha^+ = \{\sigma \in \Sigma \mid \alpha \leq \sigma\}$.

A guarded string over $\Sigma$ is a sequence $x = \langle \alpha_1, f_1 \rangle \langle \alpha_2, f_2 \rangle \cdots \langle \alpha_n, f_n \rangle$, where $n \geq 0$ and each $\alpha_i \in \mathcal{A}_\Sigma$ and $f_i \subseteq \Sigma$. Thus, a guarded string is an element of $(\mathcal{A}_\Sigma \times 2^\Sigma)^*$. For simplicity, we drop the brackets and write $x = \alpha_1 f_1 \alpha_2 f_2 \cdots \alpha_n f_n$.

To understand the intuition behind guarded strings (in our setting), assume that $\Sigma$ contains the names of the ports of a connector. Every time step a number of requests (writes or takes) are made to the ports. In the guarded string $\langle \alpha_1, f_1 \rangle \langle \alpha_2, f_2 \rangle \cdots \langle \alpha_n, f_n \rangle$ each atom $\alpha_i$ describes the definite presence or absence of requests on the ports at time step $i$, and $f_i$ describes the ports that fire. Thus, the guarded string contains the requests and subsequent firings of the ports from time steps 1 to $n$.

We will use guarded strings in the automaton model we propose as semantics for context-dependent connectors.Atoms $\alpha \in \mathcal{A}_\Sigma$ will encode whether the port of a connector is enabled and $f \in 2^\Sigma$ contains the ports that actually fire at a certain moment.

4. Guarded automata

In this section, we define a new automata model for context-dependent connectors. We start by introducing a generic automata model that accepts guarded strings and then define a product operation for these automata. Then, suitable restrictions are introduced to single out the class of Reo automata, i.e., automata that are valid models of context-dependent connectors, for which a synchronisation operation can then be defined.

**Definition 4.1 (Guarded Automaton).** A guarded automaton over an alphabet of ports $\Sigma$ is a non-deterministic (and possibly partial) automaton with transition labels $\mathcal{B}_\Sigma \times 2^\Sigma$. Formally, a guarded automaton is a triple $(\Sigma, Q, \delta)$ where $Q$ is a (finite) set of states and $\delta \subseteq Q \times \mathcal{B}_\Sigma \times 2^\Sigma \times Q$ is the transition relation.

Kozen originally proposed the notion of guarded automata [28]. The key difference is that ours has a different type of transition label.

We use the following notation in the representation of guarded automata:

$$q \xrightarrow{q, f} q' \iff (q, g, f, q') \in \delta$$

Intuitively, a transition $q \xrightarrow{q, f} q'$ denotes that the actions in $f$ will occur if the guard $g$ is true. If there is more than one transition from state $q$ to $q'$ we often just draw one arrow and separate the labels by commas.

Example guarded automata over the alphabet $\{a, b\}$ are depicted in Fig. 3.

A guarded automaton can be seen as an acceptor of guarded strings as follows. Given a guarded string $\alpha_1 f_1 \alpha_2 f_2 \cdots \alpha_n f_n$ and a state $q$ in the automaton the string is accepted in state $q$ if there exists $q \xrightarrow{q, f} q' \in \delta$ such that $\alpha_1 \leq g$ and $\alpha_2 f_2 \cdots \alpha_n f_n$ is accepted in $q'$. The empty string $\varepsilon$ is accepted in any state. We denote by $L_q$ the set of guarded strings accepted in a state $q$. Note that our definition of acceptance implies that $L_q$ is always non-empty and prefix-closed.
Another way to compute the language $L_q$ would be to first write every guard $g$ as a disjunction of atoms $\bigvee_i \alpha_i$ (for instance $a = a\overline{b} \lor ab$), replace the transition $q \xrightarrow{g/f} q'$ by the transitions $q \xrightarrow{\alpha_1/f} q'$ and then compute the accepted language of the automaton in the standard way. An interesting remark is that if one writes the automaton only using atoms, as described above, and then determines it using a subset construction, the resulting automaton will have a transition function of type $Q \to (1 + Q)^{A\Sigma \times 2^E}$ [29]. It is then well known [36] that such automata have as final semantics precisely the non-empty and prefix-closed languages $L \subseteq 2^{(A\Sigma \times 2^E)^*}$. (See Section 7.)

Two automata are equivalent if they accept the same language. We also introduce a notion of bisimulation, which implies language equivalence, adapted for our automaton model—as transitions correspond to multiple different firings, bisimulation does not match merely on the transition labels, but rather ensures that each firing from a given state has a firing going to bisimilar states.

**Definition 4.2 (Bisimulation).** Given guarded automata $A_1 = (\Sigma, Q_1, \delta_1)$ and $A_2 = (\Sigma, Q_2, \delta_2)$. We call $R \subseteq Q_1 \times Q_2$ a bisimulation iff for all $(q_1, q_2) \in R$:

1. For all $q_1 \xrightarrow{g/f} q'_1 \in \delta_1$ and $\alpha \in A_\Sigma$ such that $\alpha \leq g$, there exists a $q_2 \xrightarrow{g/f} q'_2 \in \delta_2$ such that $\alpha \leq g'$ and $(q'_1, q'_2) \in R$;
2. For all $q_2 \xrightarrow{g/f} q'_2 \in \delta_2$ and $\alpha \in A_\Sigma$ such that $\alpha \leq g$, there exists a $q_1 \xrightarrow{g/f} q'_1 \in \delta_1$ such that $\alpha \leq g'$ and $(q'_1, q'_2) \in R$.  

We say that two states $q_1 \in Q_1$ and $q_2 \in Q_2$ are bisimilar if there exists a bisimulation relation containing the pair $(q_1, q_2)$, and we write $q_1 \sim q_2$. Two automata $A_1$ and $A_2$ are bisimilar if there exists a bisimulation relation such that every state of one automaton is related to some state of the other automata and we write $A_1 \sim A_2$. The automata depicted in the following figure are bisimilar.

\[ a|a \] \[ \xrightarrow{a} \] \[ q \] \[ \xrightarrow{ab|a, a\overline{b}|a} q_1 \] \[ \xrightarrow{ab|a, a\overline{b}|a} q_2 \] \[ a|a \]

The following theorem captures that bisimilarity implies language equivalence. As usual the converse does not hold: language equivalence does not imply bisimilarity.

**Theorem 4.3.** Let $A_1 = (\Sigma, Q_1, \delta_1)$ and $A_2 = (\Sigma, Q_2, \delta_2)$ be guarded automata and $q_1 \in Q_1, q_2 \in Q_2$. Then, $q_1 \sim q_2 \Rightarrow L_{q_1} = L_{q_2}$.

**Proof.** First suppose $q_1 \sim q_2$. We prove that $x \in L_{q_1} \iff x \in L_{q_2}$, by induction on the length of $x$. The base case follows trivially because the empty word is accepted by any state. For the induction case, take $x = \alpha_1 f_1 \alpha_2 f_2 \cdots \alpha_n f_n$.

\[ x \in L_{q_1} \iff \exists q_1 \xrightarrow{g/f} q'_1 \in \delta_1 : \alpha_1 \leq g \text{ and } \alpha_2 f_2 \cdots \alpha_n f_n \in L_{q'_1} \]

\[ \iff \exists q_2 \xrightarrow{g/f} q'_2 \in \delta_2 : \alpha_1 \leq g' \text{ and } \alpha_2 f_2 \cdots \alpha_n f_n \in L_{q'_2} \]

\[ \iff x \in L_{q_2} \]  

\(4.1. \) Product

In this section we define a product operation for guarded automata. This definition differs from the classical definition of product for automata: the automata have disjoint alphabets and they can either take steps together or independently. However, in the latter case the transition explicitly encodes that the other automaton cannot perform a step in its current state, using the following notion. Notation $q'$ captures precisely the conditions in which automaton $A$ cannot fire in state $q$, by taking the complement of the disjunction of the requests that result in a firing.

**Definition 4.4.** Given a guarded automaton $A = (\Sigma, Q, \delta)$ and $q \in Q$ define

\[ q' = \neg \bigvee [g \mid q \xrightarrow{g/f} q' \in \delta] \]

For instance, for the automata

\[ a|a \] \[ \xrightarrow{a} \] \[ q \] \[ \xrightarrow{ab|a, a\overline{b}|a} q_1 \] \[ \xrightarrow{ab|a, a\overline{b}|a} q_2 \] \[ a|a \]

one has $q'_1 = a \lor \overline{b}$ and $q'_2 = \overline{a}$.

**Definition 4.5 (Product).** Given two guarded automata $A_1 = (\Sigma_1, Q_1, \delta_1)$ and $A_2 = (\Sigma_2, Q_2, \delta_2)$ such that $\Sigma_1 \cap \Sigma_2 = \emptyset$, define the product of $A_1$ and $A_2$ as $A_1 \times A_2 = (\Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, \delta)$ where

\[ \delta_x = \{(q, p) \xrightarrow{g/f} (q', p') \mid q \xrightarrow{g/f} q' \in \delta_1 \text{ and } p \xrightarrow{g/f} p' \in \delta_2\} \]

\[ \cup \{(q, p) \xrightarrow{g/f} (q', p') \mid q \xrightarrow{g/f} q' \in \delta_1 \text{ and } p \in Q_2\} \]

\[ \cup \{(q, p) \xrightarrow{g/f} (q', p') \mid p \xrightarrow{g/f} p' \in \delta_2 \text{ and } q \in Q_1\} \]

Here and throughout, we use $f \cup f'$ as a shorthand for $f \cup f'$. Case (1) accounts for when both automata fire in parallel. Cases (2) and (3) account for when one automata fires and the other is unable to (given by $p^*$ and $q^*$, respectively).
Note that if \( q \) has no outgoing transitions then \( q^\pi = \top \) and if \( q \) has a transition defined for every \( g \in B_2 \) then \( q^\pi = \bot \). Intuitively, if \( q^\pi = \top \) (respectively, \( q^\pi = \bot \)) then the state can never (respectively, always) inhibit the step of a state in another automaton.

The following is an example of the product of two automata.

\[
\begin{array}{c}
\text{abcdabc} \\
\text{abcdabc}
\end{array} \quad \times \quad \begin{array}{c}
\text{cd;cd;cd} \\
\text{cd(\pi \lor \delta)}\text{cd} \\
\text{cd(\pi \lor \delta)}\text{cd}
\end{array} = \begin{array}{c}
\text{abcdabc} \\
\text{abcdabc}
\end{array} \quad \begin{array}{c}
\text{abc} \lor \text{bcd} \\
\text{abc} \lor \text{bcd}
\end{array} \quad \begin{array}{c}
\text{cd;cd;cd} \\
\text{cd(\pi \lor \delta)}\text{cd} \\
\text{cd(\pi \lor \delta)}\text{cd}
\end{array}
\]

Observe that the automaton \( 1 = (\emptyset, \{\cdot\}, \emptyset) \) is a neutral element for product. The product operator satisfies expected properties such as commutativity and associativity. The first property follows directly from the definition. The second one follows from the definition and the following theorem, which states that \( (q_1, q_2)^\pi = q_1^\pi \land q_2^\pi \), for state \( (q_1, q_2) \) resulting from the product of two automata.

**Lemma 4.6.** Let \( A_1 = (\Sigma_1, Q_1, \delta_1) \) and \( A_2 = (\Sigma_2, Q_2, \delta_2) \) be guarded automata such that \( \Sigma_1 \cap \Sigma_2 = \emptyset \) and let \( A_1 \times A_2 = (\Sigma, Q_1 \times Q_2, \delta) \) be their product. For any \( (q_1, q_2) \in Q_1 \times Q_2, \)

\[
(q_1, q_2)^\pi = q_1^\pi \land q_2^\pi
\]

**Proof.** Let \( G_1 = \{ g_1 | q_1 \rightarrow g_1 \} \) and \( G_2 = \{ g_2 | q_2 \rightarrow g_2 \} \). Note that \( q_1^\pi = \neg \bigvee G_1 \) and \( q_2^\pi = \neg \bigvee G_2 \). The result follows by formula manipulation using mainly distributivity rules.

\[
\neg(q_1, q_2)^\pi = \bigvee \{ g | (p_1, p_2) \longrightarrow g \in \delta \}
\]

\[
= \bigvee \{ \{ g_1, g_2 | p_1 \rightarrow g_1, p_2 \rightarrow g_2 \} \}
\]

\[
\bigcup \{ g_1 \land (\neg \bigvee g_2, p_1 \rightarrow g_1, p_2 \rightarrow g_2) \} \quad | \quad p_1 \rightarrow g_1, p_2 \rightarrow g_2
\]

\[
\bigcup \{ g_2 \land (\neg \bigvee g_1, p_1 \rightarrow g_1, p_2 \rightarrow g_2) \} \quad | \quad p_1 \rightarrow g_1, p_2 \rightarrow g_2
\]

\[
\bigcap \{ g_1 \lor (\neg \bigvee g_2, p_1 \rightarrow g_1, p_2 \rightarrow g_2) \} \quad | \quad p_1 \rightarrow g_1, p_2 \rightarrow g_2
\]

\[
\bigcap \{ g_2 \lor (\neg \bigvee g_1, p_1 \rightarrow g_1, p_2 \rightarrow g_2) \} \quad | \quad p_1 \rightarrow g_1, p_2 \rightarrow g_2
\]

\[
= (\bigvee G_1 \land \bigvee G_2) \lor (\bigvee G_1 \land \neg \bigvee G_2) \lor (\bigvee G_2 \land \neg \bigvee G_1)
\]

\[
= (\bigcap \{ g_1 \lor (\neg \bigvee g_2, p_1 \rightarrow g_1, p_2 \rightarrow g_2) \} \lor (\bigvee G_1 \land \neg \bigvee G_2) \lor (\bigvee G_2 \land \neg \bigvee G_1)
\]

\[
= (\bigcap \{ g_2 \lor (\neg \bigvee g_1, p_1 \rightarrow g_1, p_2 \rightarrow g_2) \} \lor (\bigvee G_1 \land \neg \bigvee G_2) \lor (\bigvee G_2 \land \neg \bigvee G_1)
\]

\[
= (\bigcap \{ g_1 \lor (\neg \bigvee g_2, p_1 \rightarrow g_1, p_2 \rightarrow g_2) \} \lor (\bigvee G_1 \land \neg \bigvee G_2) \lor (\bigvee G_2 \land \neg \bigvee G_1)
\]

\[
= (\bigcap \{ g_2 \lor (\neg \bigvee g_1, p_1 \rightarrow g_1, p_2 \rightarrow g_2) \} \lor (\bigvee G_1 \land \neg \bigvee G_2) \lor (\bigvee G_2 \land \neg \bigvee G_1)
\]

\[
= \neg(q_1^\pi \land q_2^\pi) \quad \square
\]

### 4.2. Reo automata

This section focuses on a subclass of guarded automata that constitutes an operational model for context dependency. Intuitively, every transition \( q \rightarrow g \) in an automaton corresponding to some Reo connector represents that, if the connector is in state \( q \) and the boundary requests present at the moment encoded as an atom \( \alpha \) are such that \( \alpha \leq g \), then the ports \( f \) will fire and the connector will evolve to state \( q' \). Not all guarded automata correspond to valid Reo connectors. We are interested only in automata where each guard \( g(f) \) satisfies two criteria: **reactivity**—data flows only on ports where a request is made, capturing Reo’s interaction model; and **uniformity**—which captures two properties, firstly, that the request set corresponding precisely to the firing set is sufficient to cause firing, and secondly, that removing additional unfired requests from a transition will not affect the (firing) behaviour of the connector. These two properties are captured in the following definition.

**Definition 4.7** (Reo Automaton). A Reo automaton over an alphabet \( \Sigma \) is a guarded automaton \( (\Sigma, Q, \delta) \) such that for each \( q \rightarrow g q' \in \delta : \)

\[
- \quad \forall g \leq f \quad : \quad \forall \alpha \leq g' \quad \forall q \rightarrow g q' \in \delta : \quad \alpha \leq g' \quad \square
\]

Another way of thinking about uniformity is as follows. Firstly, the smallest request required to fire transition \( q \rightarrow g q' \) is the set of ports that fire, namely, \( f \). Secondly, there are no gaps in the possible requests between \( f \) and larger requests specified in the guard \( g \) (although it is possible that these possibilities appear in different transitions).
Among the guarded automata depicted in Fig. 3 only the third one is a Reo automaton (in fact, it models a FIFO1 channel). The first automaton is not uniform, because $ab \leq a \leq a$ but there is no transition whose guard $g$ is such that $ab \leq g$. The second automaton in not reactive as $ab \not\leq ab$.

Fig. 4 depicts the guarded automata for the basic channel types listed in Fig. 1. Here it is worth remarking that the automata for LossySync, AsyncDrain and PriorityMerger contain negative information in some of their guards. An alternative model for AsyncDrain is also possible, with transitions $a|a$ and $b|b$. This variant makes a non-deterministic choice between the two possibilities whenever the enabled ports are $ab$. As we will show later the presence of negative information is the key to represent and propagate context-dependent behaviour. Sync and SyncDrain have the same automata. This is because our automata model abstracts away from both the direction of data flow and the data values themselves. In more refined models, these channels would not have the same semantics.

**Lemma 4.8.** Reo automata are closed under product, i.e., product preserves reactivity and uniformity.

**Proof.** Given $A_1 = (\Sigma, Q_1, \delta_1)$ and $A_2 = (\Sigma, Q_2, \delta_2)$ Reo automata, we want to show that the automaton $A_1 \times A_2 = (\Sigma \cup \Sigma, Q_1 \times Q_2, \delta)$ is also reactive and uniform, that is for every transition $q \xrightarrow{\alpha} q'$, $g \leq f$ and for all $g \leq g' \leq f$ and $\alpha \leq g'$ there exists $q \xrightarrow{\alpha} q' \in \delta$ such that $\alpha \leq g''$.

The result follows directly from the definition of $\delta$ and the fact that the original automata are reactive and uniform.

For reactivity, we just illustrate case (1) of the definition.

Let $(q_1, q_2) \xrightarrow{\alpha} q_1', q_2' \in \delta'$. Because both $A_1$ and $A_2$ are reactive we know that $g_1 \leq f_1$ and $g_2 \leq f_2$. Thus, $g_1 g_2 \leq f_1 f_2 = f_1 f_2$.

For uniformity, the most interesting cases are (2) and (3) in the definition of product. We illustrate case (2). Let $(q_1, q_2) \xrightarrow{\alpha} (q_1', q_2) \in \delta'$. Now take any $g'$ such that $g_1 g_2' \leq g' \leq f_1$ and $\alpha \leq g'$.

We first prove that (i) $\alpha \leq g_1$ and (ii) $\alpha \leq g_2'$. For (i), suppose $\alpha \leq \neg g_1$ and we shall achieve a contradiction. We have that $\alpha \leq g' \leq f_1$. Thus, $\alpha \leq \neg g_1 f_1$. But because the automaton is reactive $g_1 \leq f_1$ and hence $\neg g_1 f_1 \leq \bot$, which is impossible. For (i), we observe that $\neg g_1 f_1 \leq g' \leq \neg \vee\{g_1 f_1, g_1 f_1\}$, and we shall achieve a contradiction. Since we have just proved that the former must hold, that is $\alpha \leq q_2'$.

Because $A_1$ is uniform, we know that for all $\alpha \in At_{\Sigma_1}$ such that $\alpha \leq g_1$ there exists $q_1 \xrightarrow{\alpha} q_1' \in \delta_1$ such that $\alpha_1 \leq g_1'$ and thus, for all $\alpha \in At_{\Sigma_1 \cup \Sigma_2}$ such that $\alpha \leq g_1$ it holds that $\alpha \leq g_1'$. Thus, there exists a transition $(q_1, q_2) \xrightarrow{\alpha} (q_1', q_2) \in \delta$ with $\alpha \leq g_1'\alpha_2'$, but by (i) we have that $\alpha \leq g_1$ and by (ii) that $\alpha \leq q_2'$.

**4.3. Synchronisation**

We now define a synchronisation operation which corresponds to connecting two ports in a Reo connector. In order for this operation to be well-defined we need that the transition labels in the automata are normalised (the formal justification for this is presented in Section 6.1). More precisely, we need each guard in a label to be a conjunction of literals. Note that in the automata presented in Fig. 4 for basic Reo channels this is already the case.

**Definition 4.9.** Given a guarded automaton $A = (\Sigma, Q, \delta)$ we define the normalisation of $A$ as $\text{norm}(A) = (\Sigma, Q, \text{norm}(\delta))$ where

$$\text{norm}(\delta) = \{q \xrightarrow{\alpha} q' | q \xrightarrow{\alpha} q' \in \delta \text{ and } g' \in \text{norm}(g)\} \quad \square$$

**Lemma 4.10.** Reo automata are closed under normalisation, i.e., normalisation preserves reactivity and uniformity. Moreover, $A \sim \text{norm}(A)$.

**Proof.** Let $A = (\Sigma, Q, \delta)$ be a Reo automaton. We want to prove that the automaton $\text{norm}(A)$ is also reactive and uniform. Reactivity follows easily from the definition. For uniformity we must show that for every transition $q \xrightarrow{\alpha} q' \in \text{norm}(\delta)$ and for all $g \leq g' \leq f$ and $\alpha \leq g'$ there exists $q \xrightarrow{\alpha} q' \in \text{norm}(\delta)$ such that $\alpha \leq g''$. Let $q \xrightarrow{\alpha} q' \in \text{norm}(\delta)$. We know that there exists $q \xrightarrow{\alpha} q' \in \delta$ such that $g' \in \text{norm}(g)$. Because the original automaton is uniform we know that

$$\forall_{g \leq g_1 \leq f} \forall_{\alpha \leq g_1} \exists q \xrightarrow{\alpha} q' \in \delta \cdot \alpha \leq g''$$

We have

$$g \leq g_1 \leq f \Leftrightarrow \forall_{g \in \text{norm}(g')} g' \leq g_1 \leq f \text{ and } \alpha \leq g'' \Leftrightarrow \exists_{g_2 \in \text{norm}(g')} \alpha \leq g_2$$
Therefore,
\[ \forall g' \in S_1 \exists q \ni g \xrightarrow{q} g' \in \text{norm}(\delta) \cdot \alpha \leq g_2 \]

The result \( A \sim \text{norm}(A) \) follows because the relation \( R = \{(q, q) \mid q \in Q\} \) is a bisimulation. Let \( (q, q) \in R \).

First, take any \( q \xrightarrow{g} q' \in \delta \) and \( \alpha \leq g \). Now note that
\[ \alpha \leq g \iff \alpha \leq \text{norm}(g) \iff \exists g' \in \text{norm}(g) \cdot \alpha \leq g' \]

Thus, by the definition of \( \text{norm}(\delta) \), there exists a transition \( q \xrightarrow{g}[f] q' \in \text{norm}(\delta) \), such that \( \alpha \leq g' \) and \( (q', q') \in R \).

Conversely, take any transition \( q \xrightarrow{g} q' \in \text{norm}(\delta) \) and \( \alpha \leq g' \), with \( g' \in \text{norm}(g) \). Now observe that \( \alpha \leq g' \Rightarrow \alpha \leq \text{norm}(g) \Rightarrow \alpha \leq g \). Thus, there is a transition \( q \xrightarrow{g} q' \in \delta \) such that \( \alpha \leq g \) and \( (q', q') \in R \). □

Now we are ready to define the synchronisation operation of two ports \( a \) and \( b \) (which are then made internal). In the new automaton only transitions where either both \( a \) or \( b \) fire or neither \( a \) nor \( b \) fire are kept—that is, \( a \) and \( b \) synchronise. In order to propagate context information (requests), we require that the guard \( g \) of the transition contains either \( a \) or \( b \), expressed by the condition \( g \leq \overline{ab} \) — meaning that for all \( \alpha \) such that \( \alpha \leq g \) either \( \alpha \leq a \) or \( \alpha \leq b \), which more or less corresponds an internal node acting like a self-contained pumping station \( [1] \). This also corresponds to the condition in connector colouring \([13]\) that the reason for no flow on a node must come from an external place (see Section 6.5), meaning that an internal node cannot actively block behaviour.

Definition 4.11 (Synchronisation). Given a guarded automaton \( A = (\Sigma, Q, \delta) \). We define the synchronisation of \( a \) and \( b \) \((a, b \in \Sigma)\) as \( \partial_{a,b} A = (\Sigma \setminus \{a,b\}, Q, \delta') \) where
\[ \delta' = \{ g \xrightarrow{a,b \setminus \{a,b\}} q' \mid q \xrightarrow{g} q' \in \text{norm}(\delta) \text{ s.t. } a \in f \iff b \in f \text{ and } g \not\leq \overline{ab} \} \]

Here, \( g \setminus ab \) is the guard obtained from \( g \) by deleting all occurrences of \( a \) and \( b \).

Lemma 4.12. Reo automata are closed under synchronisation, i.e., synchronisation preserves reactivity and uniformity.

Proof. Let \( A = (\Sigma, Q, \delta) \) be a Reo automaton and \( a, b \in \Sigma \). We want to show that the automaton \( \partial_{a,b} A = (\Sigma \setminus \{a,b\}, Q, \delta') \) is also reactive and uniform. Reactivity follows directly from the definition. For uniformity, we must show that for every transition \( q \xrightarrow{g} q' \in \delta' \) and for all \( g \leq g' \leq \hat{f} \) and \( \alpha \leq g' \) there exists \( q \xrightarrow{g}[f] q' \in \delta \) such that \( \alpha \leq g'' \).

Take a transition \( q \xrightarrow{g} q' \in \delta' \). We know that \( g \leq \overline{ab} \) and that \( a \in f \iff b \in f \). Thus,
\[ g \setminus ab \leq g' \leq f \setminus \{a,b\} \iff g \leq g' \leq \hat{f} \text{ or } g \leq g' \setminus ab \leq \hat{f} \]

Because the original automaton is uniform we know that there exists a transition \( q \xrightarrow{g''} q' \in \text{norm}(\delta) \) such that for all \( \alpha \leq g'' \) (\( \alpha \leq ab \), \( \alpha \leq \hat{f} \)),

Now we only have to prove that this transition is in \( \delta' \), i.e., \( g'' \not\leq \overline{ab} \). This follows immediately from the fact that \( \forall a,b \in \Sigma \leq g'' \) and \( g' \not\leq \overline{ab} \). □

The product and synchronisation operations can be used to obtain, in a compositional way, the guarded automaton of a Reo connector built from primitive connectors for which the automata are known. Given two Reo automata \( A_1 \) and \( A_2 \) over disjoint alphabets \( \Sigma_1 \) and \( \Sigma_2 \), \( \{a_1, \ldots, a_k\} \subseteq \Sigma_1 \) and \( \{b_1, \ldots, b_k\} \subseteq \Sigma_2 \) we construct \( \partial_{a_1,b_1} \partial_{a_2,b_2} \cdots \partial_{a_k,b_k} (A_1 \times A_2) \) as the automaton corresponding to a connector where port \( a_i \) of the first connector is connected to port \( b_i \) of the second connector, for all \( i \in \{1, \ldots, k\} \). Note that the ‘plugging’ order does not matter because of \( \delta \) is commutative and it interacts well with product. In addition, the sync channel \( \text{Sync}(a, b) \) acts as identity (modulo renaming). These properties are captured in the following lemma.

Lemma 4.13. Given Reo automata \( A_1 = (\Sigma_1, Q_1, \delta_1) \) and \( A_2 = (\Sigma_2, Q_2, \delta_2) \). Then:

1. \( \partial_{a,b} \partial_{c,d} A_1 = \partial_{c,d} \partial_{a,b} A_1 \), if \( a, b, c, d \in \Sigma_1 \).
2. \( \partial_{a,b} A_1 \times A_2 \sim \partial_{a,b} (A_1 \times A_2) \), if \( a, b \in \Sigma_1 \) and \( \Sigma_1 \cap \Sigma_2 = \emptyset \).
3. \( \partial_{a,b} (A_1 \times \text{Sync}(a, b)) \sim A_1[b/c] \), if \( a, b \notin \Sigma_1 \) and \( c \in \Sigma_1 \).

where \( A_1[b/c] \) is \( A \) with all occurrences of \( c \) replaced by \( b \).

Proof. Property 1 follows easily from the definition. For 2., first, observe that
\[ \partial_{a,b} A_1 = \{ q \xrightarrow{\text{g}[\setminus \{a,b\}]} q' \mid q \xrightarrow{g} q' \in \text{norm}(\delta_1), a \in f \iff b \in f \text{ and } g \not\leq \overline{ab} \} \]
and thus \((\partial_{a,b} A_1) \times A_2 = (\Sigma_1 \setminus \{a, b\} \cup \Sigma_2, Q_1 \times Q_2, \delta)\) where

\[
\delta = \{(q, p) \xrightarrow{\Sigma \cup \{a, b\}'} (q', p') \mid \begin{array}{l}
q \xrightarrow{\Sigma} q' \in \text{norm}(\delta_1), \\
p \xrightarrow{\{a, b\}'} p' \in \delta_2, \\
ap \in f \iff b \in f \text{ and } g \not\subseteq \overline{\alpha b}
\end{array} \}
\quad (1)
\]
\[
\cup \{(q, p) \xrightarrow{\Sigma \cup \{a, b\}'} (q', p) \mid q \xrightarrow{\Sigma} q' \in \text{norm}(\delta_1), \\
p \in Q_2, \\
a \in f \iff b \in f \text{ and } g \not\subseteq \overline{\alpha b}
\}
\quad (2)
\]
\[
\cup \{(q, p) \xrightarrow{\Sigma} (q', p') \mid p \xrightarrow{\{a, b\}'} p' \in \delta_2, \\
q \in Q_1, \\
and q' \not\subseteq \overline{\alpha b}
\}
\quad (3)
\]

Now, note that \(\partial_{a,b}(A_1 \times A_2) = (\Sigma_1 \setminus \{a, b\} \cup \Sigma_2, Q_1 \times Q_2, \delta)\) where

\[
\delta = \{(q, p) \xrightarrow{\Sigma \cup \{a, b\}'} (q', p') \mid \begin{array}{l}
q \xrightarrow{\Sigma} q' \in \text{norm}(\delta_1), \\
p \xrightarrow{\{a, b\}'} p' \in \text{norm}(\delta_2), \\
ap \in f \iff b \in f \text{ and } g \not\subseteq \overline{\alpha b}
\end{array} \}
\quad (1)
\]
\[
\cup \{(q, p) \xrightarrow{\Sigma \cup \{a, b\}'} (q', p) \mid q \xrightarrow{\Sigma} q' \in \text{norm}(\delta_1), \\
p \in Q_2, \\
a \in f \iff b \in f \text{ and } g \not\subseteq \overline{\alpha b}
\}
\quad (2)
\]
\[
\cup \{(q, p) \xrightarrow{\Sigma} (q', p') \mid p \xrightarrow{\{a, b\}'} p' \in \delta_2, \\
q \in Q_1, \\
and g' \not\subseteq \overline{\alpha b}
\}
\quad (3)
\]

One can easily see that \(\partial_{a,b}(A_1 \times A_2) = \text{norm}(\partial_{a,b}A_1 \times A_2)\) and thus, by Lemma 4.10:

\[
\left(\partial_{a,b}A_1\right) \times A_2 \sim \partial_{a,b}(A_1 \times A_2)
\]

For 3., we have \(\partial_{a,c}A \times \text{Sync}(a, b) = (\Sigma[b/c], \{q \cdot c \mid q \in Q\}, \delta')\), where

\[
\delta' = \{(q, \cdot) \xrightarrow{\Sigma[b/c]'} (q', \cdot) \mid q \xrightarrow{\Sigma} q' \in \delta, c \in f, f = f'c, g = g'c\}
\quad (1)
\]
\[
\cup \{(q, \cdot) \xrightarrow{\{g \cup c\}f} (q', \cdot) \mid q \xrightarrow{\{g \cup c\}f} q' \in \delta, c \not\in f \text{ and } g \not\subseteq \overline{\alpha c}\}
\quad (2)
\]
\[
\cup \{(q, \cdot) \xrightarrow{\{g \cup c\}f} (q', \cdot) \mid q \xrightarrow{\{g \cup c\}f} q' \in \delta, c \not\in f\}
\quad (3)
\]

Now, note that the relation

\[
R = \{(q, \cdot), q \mid q \in Q\}
\]

is a bisimulation. Let \((q, \cdot), q \in R\).

First, take any transition \((q, \cdot) \xrightarrow{\Sigma} q' \in \delta\) and \(c \leq g_1\). If it comes from (1) or (2), there exists \(q \xrightarrow{\Sigma} q' \in \delta[b/c]\) and \(g \leq g_1 \leq \overline{f}c\). Because \(A\) is uniform then we know that there exists \(q \xrightarrow{\Sigma} q' \in \delta[b/c]\) such that \(c \leq g''\).

If the transition comes from (3), note that \(g_1 = (g \cup c)\overline{b} \leq (g \cup c)\overline{g} \leq \overline{f}c\). Thus, since \(A\) is uniform, we know that there exists \(q \xrightarrow{\Sigma} q' \in \delta[b/c]\) such that \(c \leq g''\).

Conversely, take any transition \((q, \cdot) \xrightarrow{\Sigma} q' \in \delta[b/c]\) and \(c \leq g\). If \(b \in f\), there exists a transition \((q, \cdot) \xrightarrow{\Sigma} (q', \cdot) \in \delta\) and \(c \leq g' = g\). If \(b \not\in f\) and \(g \leq \overline{f},\) there exists a transition \((q, \cdot) \xrightarrow{\{g \cup c\}f} (q', \cdot) \in \delta\) and \(c \leq g \cup c\overline{g} = g\). If \(b \not\in f\) and \(g \leq \overline{f},\) there exists a transition \((q, \cdot) \xrightarrow{\{g \cup c\}f} (q', \cdot) \in \delta\) and \(c \leq g \cup c\overline{g}.\) \(\square\)

Moreover, we remark that \(\sim\) is a congruence with respect to the product and synchronisation operations.

**Lemma 4.14 (Congruence).** Given Reo automata \(A = (\Sigma_A, Q, \delta), A_1 = (\Sigma, Q_1, \delta_1)\) and \(A_2 = (\Sigma, Q_2, \delta_2)\) such that \(A_1 \sim A_2\). Then,

1. \(A_1 \times A_2 \sim A_1 \times A_2\)
2. \(\partial_{a,b} A_1 \sim \partial_{a,b} A_2\)
Proof. For item 1., $A = (\Sigma_A, Q, \delta), A_1 = (\Sigma, Q_1, \delta_1)$ and $A_2 = (\Sigma, Q_2, \delta_2)$ be Reo automata such that $A_1 \sim A_2$ and $\Sigma_A \cap \Sigma = \emptyset$.

Furthermore, let $A_1 \times A = (\Sigma \cup \Sigma_A, Q_1 \times Q, \delta')$ and let $A_2 \times A = (\Sigma \cup \Sigma_A, Q_2 \times Q, \delta'')$.

We have to prove that for every $(q_1, q) \in Q_1 \times Q$ there exists $(q_2, q) \in Q_2 \times Q$ such that $q_1 \sim q_2$ and, conversely, for every $(q_2, q) \in Q_2 \times Q$ there exists $(q_1, q) \in Q_1 \times Q$ such that $q_1 \sim q_2$.

By hypothesis we have $A_1 \sim A_2$, that is for every $q_1 \in Q_1$ there exists $q_2 \in Q_2$ such that $q_1 \sim q_2$ and vice versa. We define

$$R = \{(q_1, q), (q_2, q) \mid q \in Q, q_1 \sim q_2\}$$

and prove that it is a bisimulation, which yields the intended result.

Before showing the proof we introduced the following notation. We shall write $\alpha \in \mathcal{At}_{\Sigma \cup \Sigma_A}$ as $\alpha = \alpha_1 \alpha_2$ with $\alpha_1 \in \mathcal{At}_\Sigma$ and $\alpha_2 \in \mathcal{At}_{\Sigma_A}$.

Let $(\langle q_1, q \rangle, \langle q_2, q \rangle) \in R$. There are 3 possibilities for the state $(q_1, q)$ of the automaton $A_1 \times A$ to make a transition:

(i) $(q_1, q) \xrightarrow{g_i \mid f_i} \langle q_1', q' \rangle$ if $q_1 \xrightarrow{g_i \mid f_i} q_1'$ and $q \xrightarrow{g_i \mid f_i} q'$;

(ii) $(q_1, q) \xrightarrow{g_1 \mid f_1} \langle q_1', q_1 \rangle$ if $q_1 \xrightarrow{g_1 \mid f_1} q_1'$;

(iii) $(q_1, q) \xrightarrow{\delta \mid g} \langle q_1, q' \rangle$ if $q \xrightarrow{\delta \mid g} q'$.

We have to show that the state $(q_2, q)$ of the automaton $A_2 \times A$ can match any of the above transitions with a compatible transition, which is a transition such that for any $\alpha \in \mathcal{At}_{\Sigma \cup \Sigma_A}$, $\alpha$ implies both guards of the matching transitions.

For (i), we observe that the fact that $q_1 \sim q_2$ implies that for any transition $q_1 \xrightarrow{g_i \mid f_i} q_1'$ there exists a transition $q_2 \xrightarrow{g_i \mid f_i} q_2'$ such that for any $\alpha \in \mathcal{At}_\Sigma$ with $\alpha_1 \leq g_i \Rightarrow \alpha_1 \leq g_2$ (by 1) and $q_1' \sim q_2'$. The transition $(q_1, q) \xrightarrow{g_i \mid f_i} (q_1', q'')$ will then be matched by $(q_2, q) \xrightarrow{g_i \mid f_i} (q_2', q'')$, since any $\alpha = \alpha_1 \alpha_2 \in \mathcal{At}_{\Sigma \cup \Sigma_A}$ such that $\alpha \leq g_1 g$ satisfies $\alpha \leq g_2 g$ (by 1) and $(\langle q_1', q'' \rangle, \langle q_2', q'' \rangle) \in R$ (by 1).

For (ii), the key observation is the fact that $q_1 \sim q_2$ implies that for all $\alpha \in \mathcal{At}_\Sigma$, if $\alpha_1 \leq q_1''$ then $\alpha_1 \leq q_2''$. Hence, $(q_1, q) \xrightarrow{g_1 \mid f_1} (q_1', q')$ if $q \xrightarrow{g_1 \mid f_1} q'$ is matched by $(q_2, q) \xrightarrow{g_2 \mid f_2} (q_2', q')$ as a direct consequence of $q_1 \sim q_2$.

For (iii), the key observation is the fact that $q_1 \sim q_2$ implies that for all $\alpha \in \mathcal{At}_\Sigma$, if $\alpha_1 \leq q_2''$ then $\alpha_1 \leq q_2''$. Hence, $(q_1, q) \xrightarrow{\delta \mid g} (q_1', q')$ if $q \xrightarrow{\delta \mid g} q'$ is matched by $(q_2, q) \xrightarrow{g \mid \delta} (q_2', q)$.

For item 2., we observe that $A_1 \sim A_2$ implies, by Lemma 4.10, that $\text{norm}(A_1) \sim \text{norm}(A_2)$. Moreover, for any transitions $q_1 \xrightarrow{\delta_1} q_1', q_2 \xrightarrow{g_1 \mid f_1} q_2'$ as any automaton $a_{\delta_1,b}A_1$ can be matched by the transition $q_2 \xrightarrow{g_1 \mid f_1} q_2'$ of the automaton $a_{\delta_1,b}A_2$, where $q_2$ refers to any state $q_2$ of the automaton $\text{norm}(A_2)$ satisfying $q_1 \sim q_2$. □

5. Two example models

The model presented thus far is defined abstractly in terms of Boolean algebras, whereas existing automata-based models of Reo are defined more concretely in terms of some specific underlying model. In this section, we take two existing models of Reo connectors, namely port automata [27] and constraint automata [7], and present context-dependent variants of these using our formalism by describing how the one-step behaviour in these models is represented as a Boolean algebra.

5.1. Context-dependent port automata: pure synchronisation

The port automata model of Reo [27] is an automata-based model used to study the decomposition of automata into more primitive ones. Port automata abstract away from data flow, and thus present exclusively the synchronisation present in a Reo connector. For example, the following is a port automata for a LossySync$(a, b)$ channel:

$$\begin{array}{c}
\langle a, b \rangle \\
\lambda \\
\{a\}
\end{array}$$

Our model can be used to provide a context-dependent variant of port automata, by basing our guarded automata on the power set Boolean algebra, $2^X$, where $X$ is the set of ports of the connector, and $\land = \cup, \lor = \cap, \neg = X \setminus$, as usual. Each transition of such an automaton has the form $q \xrightarrow{A|B} q'$, where $A \subseteq 2^X$ and $B \in 2^X$. Elements of $A$ represents the set of ports at which a write or take is being attempted and $B$ represents the ports that fire synchronously. A context-dependent LossySync is represented as:

$$\begin{array}{c}
\{a, b\} \\
\{a\} \\
\{a\}
\end{array}$$
On the other hand, a non-deterministic, hence not context-dependent, \textit{LossySync} channel, as represented by the port automaton above, would be modelled as:

\[
\{(a, b)\}; \{(a, b)\} \xrightarrow{\{\{a, b\}; \{a\}\}} \{(a, b)\}; \{(a)\}
\]

In this setting reactivity and uniformity (Definition 4.7) become much simpler to state: for any transition \( q \xrightarrow{A_{IR}} q' \) in the automaton, we have

\textbf{reactivity} \( B \in A \) — meaning that only ports where an attempt to fire is made can fire.

\textbf{uniformity} for all \( C \in 2^X \) such that \( B \subseteq C \subseteq A_0 \in A \), there is also a transition \( q \xrightarrow{A_{IR}} q' \) in the automaton such that \( C \in A' \).

Uniformity can be more simply stated if all relevant behaviour is not distributed over multiple transitions in the automaton, but rather occurs in one transition \( q \xrightarrow{A_{IR}} q' \). In this case uniformity states that for all \( C \in 2^X \) such that \( B \subseteq C \subseteq A_0 \in A \), we have also \( C \in A \).

5.2. Context-dependent constraint automata: synchronisation and data flow

Constraint automata were the original automata-based model of Reo connectors [7]. Transitions in this model describe both the ports of a connector that synchronise along with data flow at those ports. Specifically, each transition label consists of two components: the set of ports at which data flows, and a constraint over those ports describing the data flow. For example, the semantics of a \textit{FIFO1} buffer can be represented by the following automata, where there is a state \textit{full} \( d \) and pair of transitions for each \( d \in \text{Data} \):

\[
\text{empty} \xrightarrow{\{a, d_a = d\}} \text{full}(d) \xrightarrow{\{b, d_b = d\}} \text{empty}
\]

This automaton states, firstly, that in state \textit{empty} data can flow on port \( a \) alone, the datum must match the constraint \( d_a = d \), meaning that the datum on port \( a \) has value \( d \) for some \( d \in \text{Data} \), and the automaton goes into state \textit{full}(\( d \)). Similarly, in state \textit{full}(\( d \)), data can flow on port \( b \) alone, and the datum must match constraint \( d_b = d \), meaning that the datum on port \( b \) is \( d \), the previously buffered value. Note that there will be one transition for each \( d \) with start state \textit{empty}, whereas there is only one transition with start state \textit{full}(\( d \)). So even though the constraints on both transitions are of the same shape, this automaton does capture the behaviour of a \textit{FIFO1} buffer, as when the channel is full in state \textit{full}(\( d \)) only datum \( d \) can flow.

We will describe the data flow using a Boolean algebra, so that we can develop a context-dependent variant of constraint automata using our model. Given a set of ports of a connector \( X \) and a non-empty set of data \( \text{Data} \), let \( X \rightarrow \text{Data} \) denote the partial functions from \( X \) to \( \text{Data} \). This models the flow of data on the ports of the connector: if \( f : X \rightarrow \text{Data} \) and \( f(x) = d \), where \( x \in X \) and \( d \in \text{Data} \), then datum \( d \) flows on port \( x \); if \( f(x) \) is undefined, then no data flows on \( x \).

Constraint automata have transitions of the form \( q \xrightarrow{N, \delta} q' \) where \( N \subseteq X \) and \( \delta \) is a constraint over \( N \), describing the data flow on ports \( N \). We assume that \( \delta \) is specified by the following grammar:

\[
\delta = T \mid \bot \mid \delta \land \delta \mid \delta \lor \delta \mid \neg \delta \mid d_a = d \mid d_a = d_o \mid P(d_a)
\]

where \( d_a \) represents the datum at port \( a, d \in \text{Data} \), and \( P(-) \) is some monadic predicate over \( \text{Data} \), corresponding to a set \( P_1 \subseteq \text{Data} \).

Interpret each of these formulae in the power set Boolean algebra \( 2^{X \times \text{Data}} \) as follows:

\[
\begin{align*}
\mathcal{I}(T) &= 2^{X \times \text{Data}} \\
\mathcal{I}(\bot) &= \emptyset \\
\mathcal{I}(\delta_1 \land \delta_2) &= \mathcal{I}(\delta_1) \cup \mathcal{I}(\delta_2) \\
\mathcal{I}(\delta_1 \lor \delta_2) &= \mathcal{I}(\delta_1) \cap \mathcal{I}(\delta_2) \\
\mathcal{I}(\neg \delta) &= 2^{X \times \text{Data}} \setminus \mathcal{I}(\delta) \\
\mathcal{I}(d_a = d) &= \{ f \in X \rightarrow \text{Data} \mid f(a) = d \} \\
\mathcal{I}(d_a = d_o) &= \{ f \in X \rightarrow \text{Data} \mid f(a) = f(b) \} \\
\mathcal{I}(P(d_a)) &= \{ f \in X \rightarrow \text{Data} \mid f(a) \in P_1 \}
\end{align*}
\]

Notice that there is no domain restriction made in the latter 3 cases, apart from including the mentioned ports.

Next write \( \text{just}(N) \) to denote that data flows exactly on ports \( N \), where \( N \subseteq X \), and define this as follows:

\[
\mathcal{I}(\text{just}(N)) = \{ f \in X \rightarrow \text{Data} \mid \text{dom}(f) = N \}
\]
Now we can provide an interpretation of the label on a transition in terms of the Boolean algebra. Interpret the label of a transition \( q \xrightarrow{N,\delta} q' \) by the set \( \delta(\text{just}(N)) \cap \delta(\delta) \). Every element of this set describes a data flow for the transition of the automaton as an element.

In our setting, context-dependent constraint automata will have transitions of the form \( q \xrightarrow{\text{just}(N)} q' \) where \( g \in 2^{X \rightarrow \text{Data}} \) (or alternatively, as an expression in the Boolean algebra), and \( f \in X \rightarrow \text{Data} \) such that \( f \in g \) (reactivity). Uniformity is as above for port automata.

Here is the context-dependent constraint automata for FIFO1, where this is a full state and connecting transitions for each \( d \in \text{Data} \):

Next is the LossySync:

\[
\{(a \rightarrow d, b \rightarrow e) \mid e \in \text{Data}\} \cup \{(a \rightarrow e, b \rightarrow d) \mid e \in \text{Data}\}
\]

6. Discussion

The model presented above contains many technical details to faithfully capture the desired context-dependent semantics. In order to justify the choices, we present theorems counter-examples to illustrate their purpose. In the examples we mark in bold transitions in the product automaton which are deleted in the synchronisation step because the condition \( b \in f \iff c \in f \) fails, and we mark in grey the transitions that are removed because \( g \leq b c \).

The following definition will come in handy.

**Definition 6.1 (Firings).** Let \( A = (\Sigma, Q, \delta) \) be a guarded automaton. Given \( q \in Q \) and \( \alpha \in \text{At}_\Sigma \), define the set of possible firings in \( q \) induced by \( \alpha \) as

\[
\text{firings}_\alpha(q, \alpha) = \{ (f, q') \mid q \xrightarrow{f} q' \in \delta \land \alpha \leq g \}
\]

We will drop the subscript \( A \) whenever the automaton is clear from the context. \( \square \)

6.1. Uniformity, normalisation and the Sync channel

A desirable property of a model of (context-dependent) connectors is that the Sync channel acts like an identity (modulo port renaming) whenever plugged into another connector (Lemma 4.13). The following example demonstrates that this property fails to hold without the uniformity property of Definition 4.7. Consider a channel \( \text{Loser}(a, b) \) which fires port \( a \) only if a request of port \( b \) is also present. Its guarded automaton is non-uniform, as it should have transition \( a|a \). Composing with an synchronous channel gives an automaton which should be \( \text{Loser}(a, d) \) if Sync behaved like the identity, but it does not:

\[
\text{Loser}(a, b) = \{(q_1, a) \mid (q_1, a) \text{ is the initial state of } A \}
\]

A similar reason justifies the fact that we have to normalise the automaton before applying the synchronisation operator. Suppose we want to compose a lossy synchronous channel with a synchronous channel. The automaton for the product \( \text{LossySync}(a, b) \times \text{Sync}(c, d) \) is:

Now applying \( \partial_{b,c} \) with and without normalising results in different automata:

The Sync channel behaves like an identity only in the second case.
Normalisation corresponds to splitting disjunctions, such as taking the guard $\alpha \lor \beta$ of some transition and producing two different transitions with guards $\alpha \beta$ and $\alpha \beta$, as we did above. Splitting the disjunction enables finer distinctions to be made, preventing the disjunct $\alpha \beta$ (corresponding to the LossySync spontaneously losing its data) being included in the resulting automaton because of its pairing with the disjunct $\alpha \beta$ (corresponding to the absence of a request on the Sync channel’s output end, which will be propagated to the LossySync and cause it to lose data).

6.2. Totality vs. inhibition

Two notions of totality can be defined for connector models. We phrase them in terms of guarded automata, although they apply to other models too. The first, totality, captures that for every boundary condition, a firing is possible even if it is that nothing fires. The second, firing up closure, captures that whenever firing is possible for some request set, all larger request sets also result in some, not necessarily the same, firing. It turns out that although either of these properties may be desirable, neither of them hold for Reo automata. A positive consequence is that a notion of request-based inhibition is possible.

Definition 6.2 (Totality). A guarded automaton $A = (\Sigma, Q, \delta)$ is said to be total if and only if for all states $q \in Q$ and for all requests $\alpha \in \text{At}_\Sigma$, $\text{firings}(q, \alpha) \neq \emptyset$. □

The presentation of connector colouring model of Reo semantics [13] requires that the colouring tables are total. Unfortunately, composition does not preserve totality. Consider the $\text{Rep-AsyncDrain}$ in Fig. 5. In the connector colouring model its colouring table is not total, which might lead to unexpected behaviours during composition, such as when a $\text{FullFIFO1}$ is plugged into the $\text{Rep-AsyncDrain}$; the composite has an empty colouring table, corresponding to “no behaviour possible”. If this is further composed with other connectors, the colouring table would remain empty, even if no connection is made with the $\text{FullFIFO1-Rep-AsyncDrain}$ composite.

We do not require totality, and due to the use of negative information in the product, composition with $\text{Rep-AsyncDrain}$ causes no problems, as its automata is one with no transitions (Fig. 5), which behaves neutrally in the composition (since $(q_1, q_2)^* = \top$).

On this point, the difference between our model of $\text{Rep-AsyncDrain}$ and the connector colour model is, informally, that our model gives behaviour $\emptyset$, whereas connector colouring gives behaviour $\emptyset$—the empty behaviour vs. no behaviour. One is the neutral element of composition, the other is the zero. Thus composition with the empty behaviour (in our model) may allow behaviour in other parts of the connector, whereas composition with no behaviour (in connector colouring) results in no behaviour.

Now consider the following weaker notion of totality.

Definition 6.3 (Firing Upclosed). A guarded automaton $A = (\Sigma, Q, \delta)$ is said to be firing upclosed if and only if for all states $q \in Q$ and for all $\alpha \in \text{At}_\Sigma$, if $\text{firings}(q, \alpha) \neq \emptyset$, then for all $\alpha_1$ such that $\alpha_1 \subseteq \alpha$, we have $\text{firings}(q, \alpha_1) \neq \emptyset$. □

It turns out that in general Reo automata do not preserve this property either. Consider the following example connector $\partial_a \partial_b \partial_{c,e} (\text{PriorityMerger}(ab, c) 	imes \text{Rep}(c', b'd))$ and its accompanying automaton, where $a$ is the higher priority port:\footnote{Note that this connector contains a causal loop, which should produce no data. A more complex variant without the causality problem can be easily produced, by inserting a $\text{SyncSpout}(a, b)$ plugged to a $\text{SyncDrain}(b', c)$ between $b$ and $b'$.}

![Diagram](attachment:diagram.png)

This automaton is not firing upclosed, as although $d|d$ produces a firing, $ad$ does not. In fact, a request on $a$ acts to inhibit the firing of $d$, without itself being fired. This kind of behaviour was not considered in previous models of Reo.

We tried to find an alternative definition of synchronisation, $\hat{\alpha}$, which preserved firing upclosure. Unfortunately, all our attempts failed to satisfy the desired equivalence $\partial_a \partial_b \partial_{c,e} A \sim \hat{\delta}_{c,e} \partial_a \partial_b A$. Embracing partiality—that is, the absence of firing upclosure—open the door to connectors which act as request-based inhibitors, as in the previous example.
6.3. Context dependency and negative guards

We now formally define the notion context dependency. This has never been formalised for any of the other existing models of Reo. The essence of context dependency is that additional requests may disable possible firings, typically, though not necessarily, enabling other firings instead. If no other firings are possibly, then the additional requests act as inhibitors, as discussed above. The dual of inhibition, namely that additional requests can cause firings not involving those requests, is not permitted by the uniformity condition, as such behaviour prevents our model from having the desired algebraic properties, as was also discussed in the previous section.

Context dependency will be defined in terms of the following notion of monotonicity, which captures that firing behaviour does not change when additional requests are made. As we shall see, a context-dependent connector need not be firing monotonic.

Definition 6.4 (Firing Monotonic). Let \( A = (\Sigma, Q, \delta) \) be a guarded automaton. \( A \) is firing monotonic if and only if for all states \( q \in Q \) and for all \( \alpha_1, \alpha_2 \in \text{At}_\Sigma \) if \( \alpha_1^+ \subseteq \alpha_2^+ \), then \( \text{firings}(q, \alpha_1) \subseteq \text{firings}(q, \alpha_2) \). That is, \( \text{firings}(q, \_\_ \_) \) is monotonic for all \( q \in Q \).

We have for Reo automata:

Lemma 6.5. A firing monotonic Reo automaton is firing upclosed.

Proof. Let \( A = (\Sigma, Q, \delta) \) be a firing monotonic Reo automaton, let \( q \in Q \) and let \( \alpha \in \text{At}_\Sigma \) such that \( \text{firings}(q, \alpha) \neq \emptyset \). Now, take \( \alpha_1 \in \text{At}_\Sigma \) such that \( \alpha^+ \subseteq \alpha_1^+ \). Because \( A \) is firing monotonic we know that \( \text{firings}(q, \alpha) \subseteq \text{firings}(q, \alpha_1) \). Thus, \( \text{firings}(q, \alpha_1) \neq \emptyset \) and \( A \) is firing upclosed. □

Note that the converse does not hold: the LossySync channel is not firing monotonic, yet it is firing upclosed.

Definition 6.6 (Context Dependency). A guarded automaton \( A \) is context dependent if and only if it is not firing monotonic. □

Thus an automaton exhibits context-dependent behaviour in state \( q \) whenever there exist \( \alpha_1, \alpha_2 \in \text{At}_\Sigma \) such that \( \alpha_1^+ \subseteq \alpha_2^+ \) and \( \text{firings}(q, \alpha_1) \not\subseteq \text{firings}(q, \alpha_2) \). Intuitively, this means that the state \( q \) has a transition that will be blocked in the presence of certain additional requests.

In the following automata, the state \( q \) exhibits context-dependent behaviour, because \( \text{firings}(q, ab\bar{a}) = \{(q, a)\} \not\subseteq \{(q, ab)\} = \text{firings}(q, ab) \), whereas the state \( p \) does not.

\[
\begin{array}{c}
\text{\( a\bar{b}[a, ab]ab \)}}
\end{array}
\]

\[
\begin{array}{c}
\text{\( p\)}}
\end{array}
\]

As another example, consider the following automata, an asynchronous drain (left) and a non-deterministic asynchronous drain (right):

\[
\begin{array}{c}
\text{\( a\bar{b}[b, a\bar{b}]a \)}}
\end{array}
\]

\[
\begin{array}{c}
\text{\( p\)}}
\end{array}
\]

The automaton for an asynchronous drain is context dependent. That is, when the request is \( a \) alone, \( a \) can fire, but when the request expands to \( ab \), no transition can fire. Intuitively, this means that this variant of an asynchronous drain lets the environment determine which transition to take. The non-deterministic alternative can fire a transition even when the request is \( ab \), in which case a non-deterministic choice between the two alternatives is made. Only the non-deterministic asynchronous drain can be expressed in models that do not express context dependency.

The following lemma show that negative information in guards is required to express context dependency.

Lemma 6.7. Let \( A = (\Sigma, Q, \delta) \) be a guarded automaton for which no negative atoms appear in the guards. Then \( A \) is firing monotonic.

Proof. Let \( q \in Q \) and let \( \alpha_1, \alpha_2 \in \text{At}_\Sigma \) such that \( \alpha_1^+ \subseteq \alpha_2^+ \). Note that if a guard \( g \) only has positive atoms then the following holds

\[
\alpha_1 \leq g \iff g \subseteq \alpha_1^+ \iff g \subseteq \alpha_2^+ \iff \alpha_2 \leq g
\]

Then, we reason

\[
\text{firings}(q, \alpha_1) = \{(f, q') | q \xrightarrow{g/f} q' \text{ and } \alpha_1 \leq g\} \quad \text{(def. of firings)}
\]

\[
\subseteq \{(f, q') | q \xrightarrow{g/f} q' \text{ and } \alpha_2 \leq g\} \quad \text{(by (1))}
\]

\[
= \text{firings}(q, \alpha_2) \quad \text{(def. of firings)} \quad \square
\]
It is interesting to remark that firing monotonicity is not preserved by product. As a counter-example consider the product of the automata corresponding to two \textit{FIFO1} channels — \textit{FIFO1}(a, b) × \textit{FIFO1}(c, d). The original automata are firing monotonic whereas the product automaton is not (the automaton appears in Fig. 7). When two connectors are put in parallel using the product operator we need them to be aware of each other in order to ensure that context is properly propagated when their ports are connected. Taking a product of the automata of two connectors is as if the connectors are placed together in a box without connecting any ports.

As we showed in Section 5.2, constraint automata [7] can be embedded in a natural way into our model by transforming every transition labelled by \textit{F} into a transition labelled by \textit{\hat{F}}\textit{F}. As a consequence of the previous lemma, this makes explicit the fact that constraint automata do not exhibit context-dependent behaviour. Naturally, the embedding is not compositional, as the notions of composition for constraint automata and for our automata model are too different.

6.4. Enabledness and product

We now formally define the notion of enabledness, which captures that a port can fire whenever a request is made on that port (in a given state). This property has not been previously formalised for existing models of Reo. We also show that this property is propagated through product, though this would not be the case if negative information were not included in the definition of product.
Intuitively, a port $a$ is enabled whenever all request sets containing $a$ match some guard $g$ and furthermore $a$ subsequently fires.

**Definition 6.8 (Enabledness).** Let $A = (Σ, Q, δ)$ be a guarded automaton. A port $a ∈ Σ$ is enabled in a state $q$ if for all $α ∈ Aσ$ such that $α ≤ a$, (1) $\text{firings}(q, α) ≠ ∅$ and (2) for all $(f, ...) ∈ \text{firings}(q, α)$ we have $a ∈ f$. □

Including negative information in the definition of product (using $q^\circ$) preserves enabledness through product.

**Lemma 6.9.** Let $A_1 = (Q_1, δ_1)$ and $A_2 = (Q_2, δ_2)$ be guarded automata with $Q_1 ∩ Q_2 = ∅$. Assume that in $A_1$ the port $a \in Q_1$ is enabled in state $q \in Q_1$. Then in $A_1 × A_2$, the port $a$ is enabled in all states $(q, q')$, where $q' \in Q_2$.

**Proof.** It is obvious that condition (1) follows as a consequence of the transition $(q, q') \xrightarrow{g \circ} (q_1, q_1') \in δ_1 × δ_2$. For condition 2, the proof follows by case analysis of the product definition. The most interesting case is the third clause, where $(q, q') \xrightarrow{g' \circ} (q, q_1')$. Here, the key observation is that if $α ≤ g$ for some $q \xrightarrow{g} q_1 \in δ_1$ then $α ≤ q_1'$ and thus $(f', (q, q_1')) \notin \text{firings}(q, q_1, α)$. □

Without negative information in the product, enabledness is not preserved, as the following counter-example demonstrates. Port $a$ of $\text{LossySync}(a, b)$ is enabled. If we remove the $q^\circ$ from the definition of product, then the naive definition of product $(×)$ following the definition in constraint automata directly, then $a$ is no longer enabled in $\text{LossySync}(a, b) × \text{Sync}(c, d)$, because a transition with guard $cd|\overline{cd}$ is present in the resulting automaton. This transition matches request set $acd$, but $a$ does not fire.

6.5. Justification of the $g \not< \overline{a}b$ condition in $∂_{a,b}$

The $\text{LossySync-FIFO1}$ example (Fig. 6) alone motivated research into context-dependent models of Reo. When the FIFO buffer is empty, data must flow through the $\text{LossySync}$ into the buffer, as the buffer’s port $c$ is enabled. Our product and synchronisation operations ensure this. What existing research lacks is a general and formal characterisation of the requirements underlying this example. We believe that until now, the required technical machinery was missing.

The following definition generalises the requirements underlying the $\text{LossySync-FIFO1}$ example. It captures the idea that, under certain circumstances, a particular port must fire if a request is made on it. We define a port to be sensitive for some state and request set whenever that port fires when it too is in the request set.

**Definition 6.10.** Let $A = (Σ, Q, δ)$ be a guarded automaton. We say that a port $a \in Σ$ is $g$-sensitive for state $q \in Q$ and request set $R \subseteq Σ$ whenever $a ∈ f$ for all $(f, ...) ∈ \text{firings}(q, α_{R∪{a}})$ and $\text{firings}(q, α_{R∪{a}}) ≠ ∅$. □

This property holds for port $b$ in $\text{LossySync}(a, b)$ with request set $\{a\}$, and for port $c$ in $\text{FIFO1}(c, d)$ in state empty $e$ for all request sets. In contrast, port $a$ of $\text{Merge}(ab, c)$ is not sensitive for request set $\{b, c\}$.

The following lemma captures the property underlying the $\text{LossySync-FIFO1}$ example. It says that if both $a$ and $b$ are mutually enabled in the presence of request set $R$, then they will both fire when synchronised, excluding the alternative possibility that neither fires. Were this property not to hold, then the $\text{LossySync}$ would lose data even when connected to an empty FIFO. Constraint automata [7] would include both alternatives, and is thus not sufficiently sensitive.

**Lemma 6.11.** Let $A_i = (Q_i, δ_i)$ be Reo automata, for $i \in \{1, 2\}$, with $Q_1 ∩ Q_2 = ∅$, and $a_i \in Σ_i$, $q_i \in Q_i$, $R_i \subseteq Σ_i$, such that $a_i \notin R_i$. If $a_i$ is $(q_i, R_i)$-sensitive, for $i \in \{1, 2\}$, then

\[
\text{firings}_{\sigma_1, a_1 \times a_2}(\{(q_1, q_2), α_{R_1∪R_2}\}) = \{(f \setminus \{a_1, a_2\}, q') | (f, q') ∈ \text{firings}_{A_1 \times A_2}(\{(q_1, q_2), α_{R_1∪R_2∪\{a_1, a_2\}}\})
\]

**Proof.** First, note that

\[
δ_{a_1, a_2}(A_1 × A_2) = \{(q_1, q_2) \xrightarrow{g_1|f_1|a_1|a_2|f_2(\{a_1, a_2\})} (q_1', q_2') | q_1 \xrightarrow{g_1|f_1} q_1' ∈ δ_1, q_2 \xrightarrow{g_2|f_2} q_2' ∈ δ_2, g_1g_2 \not< a_1a_2\).
\]

This is a direct consequence of sensitivity: since $a_i \in f_i$ for $q_i \xrightarrow{g_i} q_i' \in δ_i$ ($i = 1, 2$), transitions in the product automaton of type $(q_1, q_2) \xrightarrow{g_1|f_1} (q_1', q_2)$ or $(q_1, q_2) \xrightarrow{g_2|f_2} (q_1, q_2')$ will immediately be ruled out in $∂_{a_1, a_2}$ by the condition $a_1 \in f_i ⇔ a_2 \in f_i$. Thus, we have:

\[
\text{firings}_{\sigma_1, a_1 \times a_2}(\{(q_1, q_2), α_{R_1∪R_2}\})
\]

\[
= \{((q_1', q_2'), f_2(\{a_1, a_2\}) | q_1 \xrightarrow{g_1|f_1} q_1' ∈ δ_1, q_2 \xrightarrow{g_2|f_2} q_2' ∈ δ_2, α_{R_1∪R_2} \leq g_1g_2 \setminus a_1a_2, g_1g_2 \not< a_1a_2)\}
\]
6.6. Choice of operations

The original model of constraint automata [7] included one operation for composing automata, namely a join, which played a similar role to both of our operations combined. Having a separate product and synchronisation operation enables a more fine-grained analysis of automata, which we believe was required to obtain the results presented here. Barbosa et al. [8] go even further, presenting 5 operations (parallel, interleaving, hook, left join and right join). Our product merely places two connectors next to each other, without restricting their behaviour, whereas Barbosa et al.’s model forces a choice between parallel or interleaving composition. Left join and right join (approximately the counterpart of replicator and merger) are modelled by primitive automata in our model, not as operations. Their hook operation is the same as our synchronisation.

6.7. ‘Hiding’

Constraint automata [7] models of Reo include a ‘hiding’ operation, which compresses τ transitions in the automata, which are transitions labelled by \( T \) in our model. See Fig. 7. This can be used to obtain an automaton for a FIFO2 channel from the composite of two FIFO1 channels. An alternative variant defined by Costa [16] is equally applicable, and perhaps more robust.

6.8. Maximal concurrency

Guarded automata (and thus also Reo automata) exhibit a kind of maximal concurrency property with respect to maximally enabled transitions, i.e., transitions labelled with \( T \). Consider for example four FIFO1 channels, the first one from port a to port b, the second from c to d, a third from e to f and, finally, the fourth one from g to h. Synchronising ports b and c results in the Reo connector and Reo automaton described in Fig. 7. It has four states, each representing whether the first or second FIFO1 is either full (f) or empty (e). Clearly, the synchronisation of ports f and g will result in a similar Reo automaton with transition labels renamed to ports e and f. The product of these two automata will have 16 states. In particular there will be a τ transition from the state \((f, e), (f, e)\) to the state \((e, f), (e, f)\) denoting the shift of the data from buffers 1 and 3 to buffers 2 and 4, respectively. What is more important is that there will be no transitions from state \((e, f), (f, e)\) or \((f, e), (e, f)\). That is, all the enabled transitions from \((f, e), (f, e)\) fire together, even if the two connectors are unrelated.

The property is analogous to maximal concurrency of Petri nets expressed using so-called step semantics [39]. In the Petri net setting the net is still available to express the notion of maximal concurrency, whereas in our setting the topology of the connector is not even considered, so it is not clear how to express precisely the notion of maximal concurrency, nor is it clear how to specify a semantics that avoids having unrelated connectors fire together. These are topics for future research.
7. The final (trace) semantics for Reo

In this section, we show how the definition of guarded automaton above can be rewritten in order to be seen as a partial deterministic automaton. This will allow us to provide a final semantics: it is well known that partial deterministic automata with transition labels in \( A \) have as final semantics non-empty and prefix-closed languages \( L \subseteq 2^{\sum} \) [36]. Thus, we will be providing a trace semantics for the original non-deterministic automata. Consequently, the guarded automata presented in this paper are acceptors of non-empty, prefix-closed languages \( L \subseteq GS_{\sum} = 2^{(A_{\sum} \times 2^{\sum})^*} \).

This section will not focus in the Reo subclass of guarded automata, but all the results presented here are valid for that subclass.

Note that the automata presented in this paper have labels in \( L_{\sum} \times 2^{\sum} \). This means that determinisation using a subset construction similar to that for ordinary automata is not enough in order to obtain a partial deterministic automaton of the right type. Our definition of deterministic automaton will then differ from the classical one in the sense that we not only require each state to have a single transition for each label but we will also process the transition labels in order to replace guards by appropriate atoms. A deterministic guarded automaton is then a triple \((\sum, Q, \Delta)\) where

\[
\Delta : Q \rightarrow (1 + Q)^{A_{\sum} \times 2^{\sum}}.
\]

Given a guarded automaton \( A = (\sum, Q, F, \delta) \) we define the corresponding partial deterministic automaton as \( Det(A) = (\sum, 2^Q, \Delta) \), where

\[
\Delta(S)(\alpha, o) = \begin{cases} \kappa_1(\ast) & \text{if } S' = \emptyset \\ \kappa_2(S') & \text{otherwise} \end{cases}
\]

where \( S' = \{ q' \mid (q, g, o, q') \in \delta, \alpha \leq g, q \in S \} \)

Here, \( \kappa_1 : 1 \rightarrow 1 + Q \) and \( \kappa_2 : Q \rightarrow 1 + Q \) denote the usual injection functions and \( 1 = \{ \ast \} \).

One can easily prove that both automata are language equivalent: it follows easily by induction on the length of guarded strings that

\[
L_S = \bigcup_{q \in S} L_q.
\]

As an example of determinisation consider the automaton representing the FIFO buffer. Determinisation would yield the following automaton (we do not draw undefined transitions). Note also that state \( \{ e, f \} \) is included for illustration purposes—it is not reachable from the initial state \( e \).

One can now determine the language accepted by each state and obtain:

\[
L_e = ((ab|a + a|ab)(ab|b + \bar{a}|b))^*((ab|a + a|ab)(a + \epsilon))
\]

\[
L_f = ((ab|b + \bar{a}|b)(ab|a + a|ab))^*((ab|b + \bar{a}|b)(a + \epsilon))
\]

\[
L_{\{e, f\}} = (ab|a + a|ab)aL_f + (ab|b + \bar{a}|b)aL_e + \epsilon
\]

The equality \( L_{\{e, f\}} = L_e + L_f \) can be easily derived, using the axioms of regular expressions.

As another example of determinisation take the following guarded automaton. Observe not only the explosion in the number of states, but also the number of transitions (recall that label \( ab|b, \bar{a}b|b \) corresponds to two transitions).

The corresponding deterministic automaton is the following:
8. Formal models of Reo

Numerous models have been proposed in the literature to capture the state-dependent, synchronisation and mutual exclusion constraints imposed by a Reo connector over its ports. Providing a semantic model which captures the desired context-dependent nature of Reo connectors in a compositional manner has, however, been a challenge. Models either express no context dependency or are inadequate at doing so. In all cases, apart from David Costa's Ph.D. thesis [16], no formal definition of context dependency has been provided. The present paper not only presents a definition of context dependency, but also theorems establishing properties an automata model should satisfy both so that the composition operators and so forth interact sensibly, along with an analysis of the underlying enabledness and sensitivity conditions which appear at the heart of context dependency.

Constraint automata [7] have transitions whose labels capture the synchronisation (and data flow) between ports, implicitly expressing mutual exclusion, by describing the sets of ports that fire together (the ‘firing set’) at the exclusion of the ports not mentioned in the set. In their basic form, however, constraint automata cannot express context dependency.

A coalgebraic model of Reo [6] was provided in terms of relations on timed data streams (so-called Abstract Behaviour Types [2]). These were shown to be more or less equivalent to constraint automata, and thus unable to express context dependency. Moreover, the underlying time streams are infinite, so the model excludes not only finite behaviour, but also connectors which exhibit finite behaviour on any of their ports.

Connector colouring [13] describes the behaviour of a connector in a compositional fashion by colouring the parts where data flows and where it does not flow with different colours, requiring simply that colours match at connected ports. The model also captures context-dependent behaviour by propagating information about the absence of data flow through the connect. This model was extended to cover both state changes and the passing of data using tile logic [3]. Nonetheless this model and its extension suffer from a number of problems. The first is that some colourings are non-causal, but this can easily be fixed by tracking the causality relation. The second problem is that degenerate behaviour can arise in certain circumstances (see Section 6.2). Colouring tables normally are defined to give a colouring for all possible boundary conditions. However, this total property is not preserved by composition. Furthermore, composition with a non-total colouring table can result in no behavioural description for connectors, whereas often the semantics should be that no flow is possible. (By analogy, this is the difference between $\emptyset$ and $\{\emptyset\}$.) As previously stated, when composed with any other connector (even when the two parts are not connected), the resulting composite has no behaviour.

Intentional automata [16] express context dependency by labelling transitions with a request set and a firing set, where the request set models the context and the firing set models the subsequent behaviour. In addition, states record pending requests—namely, requests that have arrived but have not fired. This means that there are quite a large number of states in the automata managing the buffering and firing of such pending requests, and automata rapidly become difficult to manipulate and not suitable for model checking purposes. For example, one Sync channel requires 3 states, and 2 disconnected Sync channels require 9 states. In constraint automata and our model, only 1 state is required in both cases. Although intentional automata express a notion of context dependency (depending crucially on pending requests rather than non-monotonicity), no properties analogous to the ones proven in this paper (such as Lemmas 4.13, 6.9 and 6.11) have been proven for intentional automata.

The Büchi automata model of Reo [23,24] assigns to connectors infinite fair behaviours. In that model, $\tau$-transitions capture the arrival of requests, which are recorded in states. In that model, there are two different non-equivalent ways of modelling something as simple as a Sync channel: Thus the model differs significantly from other approach.

Mousavi et al. [34] describe Reo’s semantics using structural operational semantics. To capture context-dependent behaviour (of lossy synchronous channels) a global maximal progress rule is employed to remove undesired behaviours. This was subsequently encoded into Alloy [25]. The kind of context-dependent behaviour which can be captured by this rule is limited, as it cannot express the preference between two unrelated behaviours.

Barbosa et al. [8] present models of Reo-like connectors. The semantics is given by process algebra expressions, where both the presence and absence of signals can be specified. Complex connectors are then built from simpler ones using one of five combinators: parallel composition, interleaving, hook, right and left join. However, these composition operations increase the complexity of the model without gaining any expressiveness. No properties analogous to the ones proven in this paper were presented.

Unlike constraint automata, our model can express context dependency using a request and firing set, as in intentional automata. We abstract away from data flow constraints, but indicate how to add them back into the model in Section 9. Our model is significantly more compact than intentional automata, in terms of both the number of states and transitions, as information about pending requests is not stored in states—it can easily be calculated. In contrast to the Büchi model, our model expresses only finite behaviours and records request sets in transition labels along with the firing sets, instead of in the states, resulting in more intuitive models. Furthermore, our model expresses only the positive behaviour, and does not rely crucially on the Büchi acceptance criteria to rule out unwanted ‘paths’ in automata. The semantics of our model is based on finite strings, which are much simpler than relations on timed data streams underlying the coalgebraic model. Our model also overcomes the totality problem of connector colouring by, ironically, not insisting that the transition relation is total, and by interpreting the absence of a transition simply as no behaviour for the given context. In contrast to Mousavi et al.’s model, our approach achieves an expressive notion of context dependency in a compositional manner without recourse to
a global rule. Our composition operation is a compact two-step operation, much simpler than the five operations proposed by Barbosa et al... As far as we understand, merely just adding information recording the absence of signals is insufficient to adequately deal with context-dependent behaviour.

Overall, we claim that our automata are simpler and more intuitive than existing models of context-dependent connectors. In addition, we prove numerous relevant properties about our model, not even considered by others.

9. Conclusion and future work

We have presented a new semantic model for context-dependent Reo connectors. The automata corresponding to primitive channels are very compact and intuitive. As a novelty, when compared to previous approaches, our model takes negative information into account in the composition operations. This has allowed us to provide a ‘correct’ behavioural description of connectors (such as the Repl-AsyncDrain example) which were not possible in other models. Moreover, we provided a detailed justification for the various properties of our model. We hope that our research will contribute to a more axiomatic description of Reo connectors. We also extended our model to take account of the actual data flowing through connectors, thus providing within our more general framework a context-dependent variant of constraint automata, the hitherto definitive semantic model of Reo. Moreover, our model can be used to give a significantly simpler account of quantitative Reo [4], though we do not present the details here. Recently, we incorporated our automata model into CWI's Eclipse Coordination Tools. This enables the generation of Java implementations of our automata for composing components and services.

Recently Kozen demonstrated that Kleene algebra with tests (KAT) are to guarded automata what regular expressions are to ordinary finite automata. Therefore, we would like to explore how KAT expressions can be used to specify and synthesise Reo connectors. This would give an algebraic description of Reo connectors, for which reasoning could be automated. More generally, since our automata can be seen as ordinary labelled transition systems with structured labels, we are interested in the application of temporal logic and model checking.

Other issues that demand attention include using our results to provide an axiomatic basis for Reo semantics, and exploring the maximal concurrency property of our model, including finding more realistic models that do not have this property.

References


2 http://reo.project.cwi.nl/.