Automata for Context-Dependent Connectors

Marcello Bonsangue[1,3], Dave Clarke[2], and Alexandra Silva[3]

[1] LIACS, Leiden University, The Netherlands
[2] Dept. Computer Science, Katholieke Universiteit Leuven, Belgium
[3] CWI, The Netherlands

Abstract. Recent approaches to component-based software engineering employ coordinating *connectors* to compose components into software systems. For maximum flexibility and reuse, 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. A more expressive notion of connector includes also context-dependent behaviour, namely, whenever the choices the connector can take change non-monotonically as the context, given by the pending activity on its ports, changes. Context dependency can express notions of priority and inhibition. 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.

1 Introduction

The holy grail of component-based software engineering is to develop truly reusable software components which can be sold off-the-shelf and reused to build software systems [31]. 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* [15,26,28], usually in a scripting language, whereas others offer a more visual approach, where ‘channels’ or ‘connectors’ are used to compose components into a system [1,9,14,17].

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 [1,26]. 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 [23], Ptolemy II [24], MoCha [17], Manifold [5], pipe and filter architectures [30]. Although these overlap in philosophy and functionality, Reo is the only one that enables synchrony and mutual exclusion to propagate through connectors.

The trend is to increase (or improve) the expressiveness of such coordination models by investigating features such as dynamic reconfiguration [21], data sensitive operations such as data filtering and transformation [10], and context-dependent behaviour [11]. 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 such models for Reo, in a manner which can be implemented efficiently.

Context-dependent behaviour has already been studied in the context of non-monotonic concurrent constraint programming [13] and generative communication [16], 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 intensional automata [12], 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 intensional automata. Moreover, because our automata are partial, the model overcomes a problem with *totality preservation* present in connector colouring [11].

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. 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 formalized 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 [25] with the context dependency originally developed in the connector colouring model [11].

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 describes and justifies various technical conditions present in our model, including giving properties. Section 6 concludes.

2 The Coordination Language Reo and Its Models

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 hereby 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. 3.

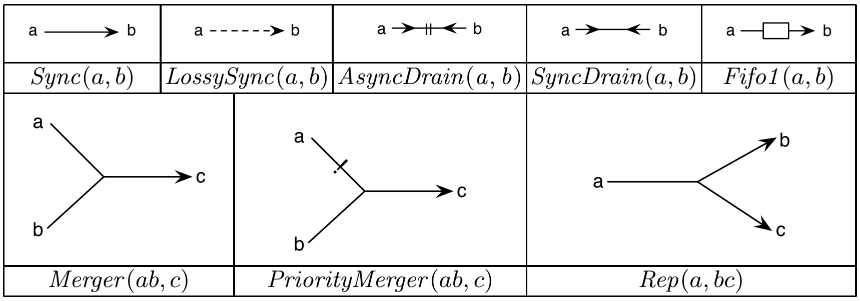


Fig. 1. Basic Reo channels

The interaction model presupposed by Reo is that components try to write or take data from the ports it is 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. Most existing formal models of Reo 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 blocks (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 must only flow synchronously. In fact, we will show in the sequel that this channel behaves as 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.* [11], 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 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 the 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. Arbab [1] describes how *offers* of data (writes) and willingness to *accept* data (takes) propagate through channels, but unfortunately, this description is incomplete and imprecise, in particular with regard to how context propagation interacts with non-deterministic choice. Clarke *et al.* [10] goes as far as arguing that there are no natural intuitive models for Reo, hence no natural or obvious way of implementing it, as our intuition about data flow networks is insufficient to determine how connectors behave. Two consequences of this are, firstly, that the semantics of any Reo connector can only be understood in terms of a specific semantic model and appropriate translation into the model, and, secondly, that the only effective implementations of Reo have been direct implementations of some semantic model; no reference model exists.

2.1 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.

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 [11] 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 negative information about the absence of data flow through the connector. 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 [12].[1] The second problem is that degenerate behaviour can arise in certain circumstances (see Section 5). Colouring tables normally are defined to give a colouring for all possible boundary conditions. However, this *totality* 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 $\{\emptyset\}$.) When composed with any other connector (even when the two parts are not connected), the resulting composite has no behaviour.

Intentional automata [12] 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 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.

The Büchi automata model of Reo [18,19] assigns to connectors infinite fair behaviours. In this model, τ-transitions capture the arrival of requests, which are recorded in states. In this model, there are two different non-equivalent ways of modelling something as simple as a Sync channel. Thus the model differs significantly from other approaches.

Mousavi *et al.* [27] 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 [20]. 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 increases the complexity of the model without gaining any expressiveness.

[1] Our model also does not deal with causality issues; Costa's fix is applicable here [12].

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 6. 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 can tell, 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.

3 Preliminaries: Guarded Strings

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

$$g ::= \ \sigma \in \Sigma \mid \top \mid \bot \mid g \vee g \mid g \wedge g \mid \overline{g}$$

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

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

An *atom* of $\mathcal{B}_\Sigma$ is a guard $a_1 \dots a_k$ such that $a_i \in \{\sigma_i, \overline{\sigma}_i\}$, $1 \leq i \leq k$. We can think of an atom as a truth assignment. We denote atoms by Greek letters $\alpha, \beta, \dots$ and the set of all atoms of $\mathcal{B}_\Sigma$ by $\mathbf{At}_\Sigma$. Every element of a finite Boolean algebra can be written as a disjunction of atoms. Given $S \subseteq \Sigma$, we define $\widehat{S} \in \mathcal{B}_\Sigma$ as the conjunction of all elements of S. For instance, for $S = \{a, b, c\}$ one has

$\widehat{S} = abc$. We define the atom associated with a set S in the expected way — $\alpha_S = \widehat{S} \wedge \widehat{\overline{\Sigma \setminus S}}$. For example, if $\Sigma = \{a, b, c\}$, then $\alpha_{\{a,b\}} = ab\overline{c}$. Conversely, the set associated with an atom α is defined as $\alpha^+ = \{\sigma \in \Sigma \mid \alpha \leq \sigma\}$.

A guarded string over Σ is a sequence $x = \langle \alpha_1, f_1 \rangle \langle \alpha_2, f_2 \rangle \ldots \langle \alpha_n, f_n \rangle$, where $n \geq 0$ and each $\alpha_i \in \mathbf{At}_\Sigma$ and $f_i \subseteq \Sigma$. Thus, a guarded string is an element of $(\mathbf{At}_\Sigma \times 2^\Sigma)^*$. 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, imagine that Σ contains the names of all doctors in a hospital. Every hour there is a meeting to distribute the incoming patients. Each atom α_i describes the definite presence or absence of every doctor in the meeting at hour i and f contains the doctors that got a patient. Thus, the guarded string $\langle \alpha_1, f_1 \rangle \langle \alpha_2, f_2 \rangle \ldots \langle \alpha_n, f_n \rangle$ will contain the activity of the doctors from hours 1 to n.

4 Guarded Automata

In this section, we define a new automata model for context-dependent connectors. We start by introducing a generic automata, acceptor of guarded strings and we define a product operation. 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 synchronization operation is defined.

Definition 1 (Guarded automaton). *A* guarded automaton *over an alphabet of ports Σ is a non-deterministic (and possibly partial) automaton with transition labels $\mathcal{B}_\Sigma \times 2^\Sigma$. Formally, a guarded automaton is a triple (Σ, Q, δ) 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.*

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

$$q \xrightarrow{g|f} q' \iff \langle q, g, f, q' \rangle \in \delta$$

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

Example guarded automata over the alphabet $\{a, b\}$ are depicted in Fig. 2.

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

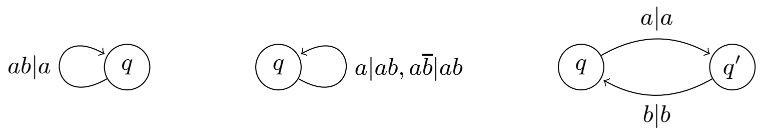


Fig. 2. Examples of guarded automata over the alphabet $\{a, b\}$

string is *accepted* in state q if there exists $q \xrightarrow{g|f_1} q' \in \delta$ such that $\alpha_1 \leq g$ and $\alpha_2 f_2 \cdots \alpha_n f_n$ is accepted in q'. The empty string ε is accepted in any state. We denote by $\mathcal{L}_q$ the set of guarded strings accepted in a state q. Note that our definition of acceptance implies that $\mathcal{L}_q$ is always non-empty and prefix-closed.

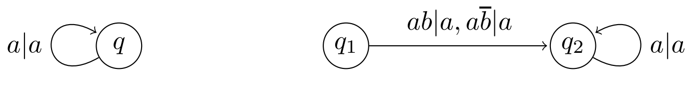
Another way to compute the language $\mathcal{L}_q$ would be to first write every guard g as a disjunction of atoms $\bigvee_I \alpha_i$ (for instance $a = a\overline{b} \vee ab$), replace the transition $q \xrightarrow{g|f_1} q' \in \delta$ by the transitions $q \xrightarrow{\alpha_i|f_1} q'$ and then compute the accepted language of the automata in the standard way. An interesting remark is that if one writes the automaton only using atoms, as described above, and then determinizes it using a subset construction, the resulting automata will have a transition function of type $Q \to (1+Q)^{\mathbf{At}_\Sigma \times 2^\Sigma}$ [22]. It is then well-known [29] that such automata have as final semantics precisely the non empty and prefix closed languages $\mathcal{L} \subseteq 2^{(\mathbf{At}_\Sigma \times 2^\Sigma)^*}$.

Two automata are equivalent if they accept the same language. We also introduce a novel notion of bisimulation, which implies language equivalence.

Definition 2 (Bisimulation). *Given guarded automata $\mathcal{A}_1 = (\Sigma, Q_1, \delta_1)$ and $\mathcal{A}_2 = (\Sigma, Q_2, \delta_2)$. We call $R \subseteq Q_1 \times Q_2$ a* bisimulation *iff for all $\langle q_1, q_2 \rangle \in R$:*

1. *For all $q_1 \xrightarrow{g|f} q_1' \in \delta_1$ and $\alpha \in \mathbf{At}_\Sigma$ such that $\alpha \leq g$, there exists a $q_2 \xrightarrow{g'|f} q_2' \in \delta_2$ such that $\alpha \leq g'$ and $\langle q_1', q_2' \rangle \in R$;*
2. *For all $q_2 \xrightarrow{g|f} q_2' \in \delta_2$ and $\alpha \in \mathbf{At}_\Sigma$ such that $\alpha \leq g$, there exists a $q_1 \xrightarrow{g'|f} q_1' \in \delta_1$ such that $\alpha \leq g'$ and $\langle q_1', q_2' \rangle \in R$.*

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 $\langle q_1, q_2 \rangle$ and we write $q_1 \sim q_2$. Two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are bisimilar if there exists a bisimulation relation such that every state of one automata is related to some state of the other automata and we write $\mathcal{A}_1 \sim \mathcal{A}_2$. The automata depicted in the following figure are bisimilar.



Theorem 1. *Let $\mathcal{A}_1 = (\Sigma, Q_1, \delta_1)$ and $\mathcal{A}_2 = (\Sigma, Q_2, \delta_2)$ be guarded automata and $q_1 \in Q_1, q_2 \in Q_1$. Then, $q_1 \sim q_2 \Rightarrow \mathcal{L}_{q_1} = \mathcal{L}_{q_2}$.*

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. In the latter case the transition explicitly encodes that the other automaton cannot perform a step in the current state, using the following notion:

Definition 3. *Given a guarded automaton $\mathcal{A} = (\Sigma, Q, \delta)$ and $q \in Q$ we define*

$$q^\sharp = \neg \bigvee \{ g \mid \; q \xrightarrow{g|f} q' \; \in \delta \}.$$

This captures precisely the conditions in which $\mathcal{A}$ cannot fire in state q. Note that if q has no outgoing transitions then $q^\sharp = \top$ and if q has a transition defined for every $g \in \mathcal{B}_\Sigma$ then $q^\sharp = \bot$. Intuitively, if $q^\sharp = \top$ (resp. $q^\sharp = \bot$) then the state can never (resp. always) inhibit the step of a state in another automaton, in the context of the product, defined below. For instance, in the automata

$$\begin{tikzcd} q_1 \arrow[loop left, "ab|a"] & & & q_2 \arrow[loop right, "ab|ab,\, a\overline{b}|ab"] \end{tikzcd}$$

one has $q_1^\sharp = \overline{a} \vee \overline{b}$ and $q_2^\sharp = \overline{a}$.

Definition 4 (Product). *Given two guarded automata $\mathcal{A}_1 = (\Sigma_1, Q_1, \delta_1)$ and $\mathcal{A}_2 = (\Sigma_2, Q_2, \delta_2)$ such that $\Sigma_1 \cap \Sigma_2 = \emptyset$, we define the* product *of $\mathcal{A}_1$ and $\mathcal{A}_2$ as $\mathcal{A}_1 \times \mathcal{A}_2 = (\Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, \delta)$ where*

$$\delta = \{ \; (q,p) \xrightarrow{gg'|ff'} (q',p') \; \mid \; q \xrightarrow{g|f} q' \; \in \delta_1 \textit{ and } \; p \xrightarrow{g'|f'} p' \; \in \delta_2 \} \tag{1}$$

$$\cup \; \{ \; (q,p) \xrightarrow{gp^\sharp|f} (q',p) \; \mid \; q \xrightarrow{g|f} q' \; \in \delta_1 \textit{ and } p \in Q_2 \} \tag{2}$$

$$\cup \; \{ \; (q,p) \xrightarrow{gq^\sharp|f} (q,p') \; \mid \; p \xrightarrow{g|f} p' \; \in \delta_2 \textit{ and } q \in Q_1 \} \tag{3}$$

Here and throughout, we use ff' 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 $p^\sharp$ and $q^\sharp$, respectively).

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

$$\begin{tikzcd} q_1 \arrow[loop left, "ab|ab"] \arrow[r, phantom, "\times"] & q_2 \arrow[loop right, "cd|cd,\, c\overline{d}|c"] \arrow[r, phantom, "="] & (q_1, q_2) \arrow[loop right, "\begin{array}{c} abcd|abcd \\ abc\overline{d}|abc \\ ab\overline{c}|ab \\ cd(\overline{a} \vee \overline{b})|cd \\ c\overline{d}(\overline{a} \vee \overline{b})|c \end{array}"] \end{tikzcd}$$

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 fact that $(q_1, q_2)^\sharp = q_1^\sharp \wedge q_2^\sharp$.

4.2 Reo Automata

In this section we focus on a subclass of guarded automata that constitutes an operational model for context dependency. Intuitively, every transition $q \xrightarrow{g|f} q'$

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 α, 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 5 (Reo automaton). *A* Reo automaton *over an alphabet Σ is a guarded automaton (Σ, Q, δ) such that for each $q \xrightarrow{g|f} q' \in \delta$:*

- $g \leq \widehat{f}$ *(reactivity)*
- $\forall g \leq g' \leq \widehat{f} \cdot \forall \alpha \leq g' \cdot \exists\, q \xrightarrow{g''|f} q' \in \delta \cdot\ \alpha \leq g''$ *(uniformity)*

Among the guarded automata depicted in Fig. 2 only the third one is a Reo automaton (in fact, it models a FIFO1 channel). The first automaton is not uniform, because $ab \leq a \leq a$ and there is no transition whose guard g is such that $a\overline{b} \leq g$. The second automaton in not reactive: $a\overline{b} \not\leq ab$.

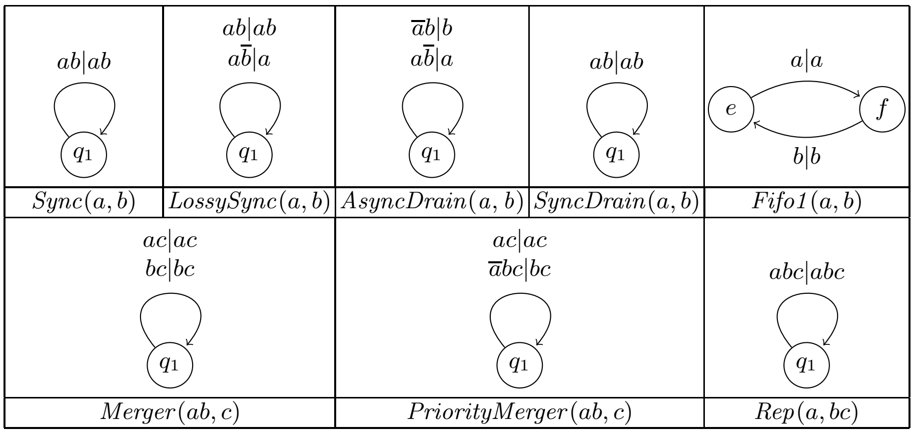


Fig. 3. Guarded automata for basic Reo channels

In Fig. 3 we depict 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. As we will show later this is the key to represent and propagate context-dependent behaviour, which all these channels exhibit.

Lemma 1. *Reo automata are closed under product,* i.e., *product preserves reactivity and uniformity.*

4.3 Synchronization

We now define a synchronization 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 normalized (the formal justification for this is presented in Section 5.1). More precisely, we need each guard in a label to be a conjunction of literals. Note that in the automata presented in Figure 3 for basic Reo channels this is already the case.

Definition 6. *Given a guarded automaton $\mathcal{A} = (\Sigma, Q, \delta)$ we define the* normalization *of $\mathcal{A}$ as $norm(\mathcal{A}) = (\Sigma, Q, norm(\delta))$ where*

$$norm(\delta) = \{\, q \xrightarrow{g'|f} q' \mid q \xrightarrow{g|f} q' \in \delta \text{ and } g' \in norm(g)\}$$

Lemma 2. *Reo automata are closed under normalization,* i.e., *normalization preserves reactivity and uniformity. Moreover, $\mathcal{A} \sim norm(\mathcal{A})$.*

Now we are ready to define the synchronization operation of two ports a and b (that are then made internal). In the new automaton only transitions where either both a and b or neither a nor b fire are kept—that is, a and b synchronize. In order to propagate context information (requests), we require that the guard contains either a or b, expressed by the condition $g \not\leq \overline{a}\overline{b}$, which more or less corresponds an internal node acting like a *self-contained pumping station* [1], meaning that an internal node cannot actively block behaviour. This also corresponds to the condition in connector colouring [11] that the reason for no flow on a node must come from an external place (see Section 5.5).

Definition 7 (Synchronization). *Given a guarded automaton $\mathcal{A} = (\Sigma, Q, \delta)$. We define the* synchronization *of a and b $(a, b \in \Sigma)$ as $\partial_{a,b}\mathcal{A} = (\Sigma, Q, \delta')$ where*

$$\delta' = \{\, q \xrightarrow{g\backslash_{ab}|f\backslash\{a,b\}} q' \mid q \xrightarrow{g|f} q' \in norm(\delta) \text{ s.t. } a \in f \Leftrightarrow b \in f \text{ and } g \not\leq \overline{a}\overline{b}\}$$

Here, $g\backslash_{ab}$ is the guard obtained from g by deleting all ocurrences of a and b.

Lemma 3. *Reo automata are closed under synchronization,* i.e., *synchronization preserves reactivity and uniformity.*

The product and synchronization 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 $\mathcal{A}_1$ and $\mathcal{A}_2$ over disjoint alphabets Σ_1 and Σ_2, $\{a_1, \ldots, a_k\} \subseteq \Sigma_1$ and $\{b_1, \ldots, b_k\} \subseteq \Sigma_2$ we construct $\partial_{a_1,b_1}\partial_{a_2,b_2}\cdots\partial_{a_k,b_k}(\mathcal{A}_1 \times \mathcal{A}_2)$ 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 ∂ is commutative and it interacts well with product. In addition, the sync channel $Sync(a, b)$ acts as identity (modulo renaming). These properties are captured in the following lemma.

Lemma 4. *Given Reo automata $\mathcal{A}_1 = (\Sigma_1, Q_1, \delta_1)$ and $\mathcal{A}_2 = (\Sigma_2, Q_2, \delta_2)$. Then:*

1. $\partial_{a,b}\partial_{c,d}\mathcal{A}_1 = \partial_{c,d}\partial_{a,b}\mathcal{A}_1$, *if* $a, b, c, d \in \Sigma_1$.
2. $(\partial_{a,b}\mathcal{A}_1) \times \mathcal{A}_2 \sim \partial_{a,b}(\mathcal{A}_1 \times \mathcal{A}_2)$, *if* $a, b \in \Sigma_1$ *and* $\Sigma_1 \cap \Sigma_2 = \emptyset$.
3. $\partial_{a,c}(\mathcal{A}_1 \times Sync(a, b)) \sim \mathcal{A}_1[b/c]$, *if* $a, b \notin \Sigma_1$ *and* $c \in \Sigma_1$.

where $\mathcal{A}[b/c]$ is $\mathcal{A}$ with all occurrences of c replaced by b.

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

5 Discussion

The model presented above contains many technical details. In order to justify them, we present a theorem and/or counter-example to illustrate their purpose. In the examples we mark in **bold** transitions in the product automaton which are deleted in the synchronization step because the condition $b \in f \Leftrightarrow c \in f$ fails, and we mark in gray the transitions that are removed because $g \leq \overline{b}\overline{c}$.

The following definition will come in handy.

Definition 8 (Firings). *Let $\mathcal{A} = (\Sigma, Q, \delta)$ be a guarded automaton. Given $q \in Q$ and $\alpha \in \mathbf{At}_\Sigma$ define the set of possible* firings *in q induced by α as*

$$\mathbf{firings}_{\mathcal{A}}(q, \alpha) = \{(f, q') \mid q \xrightarrow{g|f} q' \in \delta \ \wedge \ \alpha \leq g\}.$$

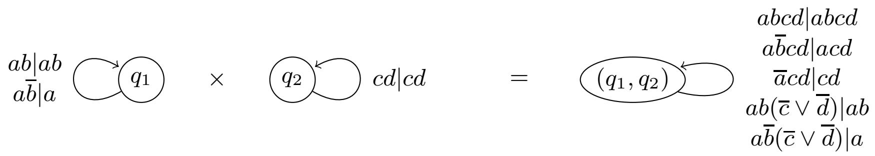
We will drop the subscript $\mathcal{A}$ whenever the automaton is clear from the context.

5.1 Uniformity, Normalization and the Sync Channel

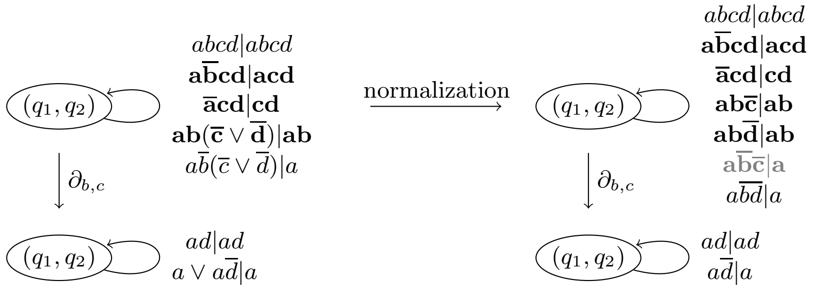
A desirable property of a model of (context-dependent) connectors is that the Sync channel acts like an identity (modulo port renaming) whenever plugged into another connector (Lemma 4). The following example demonstrates that this property fails to hold without the uniformity property of Definition 5. Consider a channel $Loser(a, b)$ 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 a synchronous channel gives an automaton which should be $Loser(a, d)$ if Sync behaved like the identity:

$$\begin{tikzcd} Loser(a,b) = q_1 \arrow[loop right, "ab|a"] \end{tikzcd} \qquad \begin{tikzcd} \partial_{b,c}(Loser(a,b) \times Sync(c,d)) = (q_1, q_1) \arrow[loop right, "a|a"] \end{tikzcd}$$

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



Now applying $\partial_{b,c}$ with and without normalizing results in different automata:


(q_1,q_2)
$abcd|abcd$
$\mathbf{a\overline{b}cd|acd}$
$\mathbf{\overline{a}cd|cd}$
$\mathbf{ab(\overline{c} \vee \overline{d})|ab}$
$a\overline{b}(\overline{c} \vee \overline{d})|a$
$\partial_{b,c}$
(q_1,q_2)
$ad|ad$
$a \vee a\overline{d}|a$
normalization
(q_1,q_2)
$abcd|abcd$
$\mathbf{a\overline{b}cd|acd}$
$\mathbf{\overline{a}cd|cd}$
$\mathbf{ab\overline{c}|ab}$
$\mathbf{ab\overline{d}|ab}$
$\mathbf{a\overline{b}\overline{c}|a}$
$a\overline{b}\overline{d}|a$
$\partial_{b,c}$
(q_1,q_2)
$ad|ad$
$a\overline{d}|a$

The Sync channel behaves like an identity only in the second case.

5.2 Totality and Inhibition

Two notions of totality can be defined for connectors. We phrase them in terms of guarded automata, although they apply to other models too.

Definition 9 (Totality). *A guarded automaton $\mathcal{A} = (\Sigma, Q, \delta)$ is said to be* total *if and only if for all states $q \in Q$ and for all $\alpha \in \mathbf{At}_\Sigma$,* $\mathbf{firings}(q,\alpha) \neq \emptyset$*.*

The presentation of connector colouring [11] requires that the colouring tables are total. Unfortunately, composition does not preserve totality. Consider the Rep-AsyncDrain in Fig. 4. In the connector colouring model its colouring table is not total, which might lead to unexpected behaviours during composition. For example, when a FullFifo$_1$ is plugged into the 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 remains empty, even if no connection is made with the FullFifo$_1$-Rep-AsyncDrain composite.

We do not require totality, and due to the use of negative information in the product, composition with Rep-AsyncDrain causes no problems, as its automata is one with no transitions (Fig. 4), which behaves neutrally in the composition (since $(q_1, q_2)^\sharp = \top$).

We also find it unnecessary to specify any behaviour that does not result in a firing (though we do permit τ-transitions, represented by $\top|\emptyset$). The following

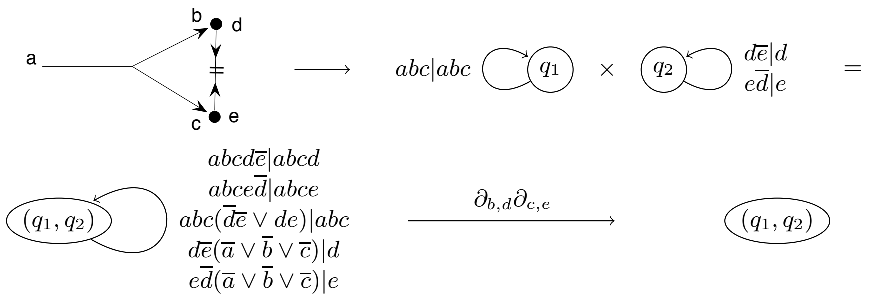
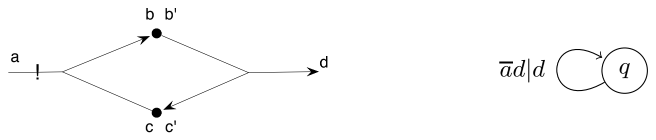

a
b
d
c
e
$abc|abc$
q_1
$\times$
q_2
$d\overline{e}|d$
$e\overline{d}|e$
$=$
(q_1, q_2)
$abcd\overline{e}|abcd$
$abce\overline{d}|abce$
$abc(\overline{de} \vee de)|abc$
$d\overline{e}(\overline{a} \vee \overline{b} \vee \overline{c})|d$
$e\overline{d}(\overline{a} \vee \overline{b} \vee \overline{c})|e$
$\partial_{b,d}\partial_{c,e}$
(q_1, q_2)

Fig. 4. Guarded automaton for $\partial_{b,d}\partial_{c,e}(Rep(a, bc) \times ASyncDrain(d, e))$

definition captures a sensible notion, which is weaker than totality. It states that if some request set α causes a firing, then all larger request sets also cause a firing (though not necessarily the same one).

Definition 10 (Firing upclosed). *A guarded automaton $\mathcal{A} = (\Sigma, Q, \delta)$ is said to be* firing upclosed *if and only if for all states $q \in Q$ and for all $\alpha \in \mathbf{At}_\Sigma$, if* **firings**$(q, \alpha) \neq \emptyset$*, then for all α_1 such that $\alpha^+ \subseteq \alpha_1^+$ we have* **firings**$(q, \alpha_1) \neq \emptyset$.

This is a nice property, but it turns out that, in general, composing Reo automata does not preserve firing upclosure. Consider the following example connector $\partial_{b,b'}\partial_{c,c'}PriorityMerger(ab, c) \times Rep(c', b'd)$ and its accompanying automaton, where a is the higher priority port: [2]



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, $\hat{\partial}$, which preserved Firing upclosed. Unfortunately, all our attempts failed to satisfy the required equivalence $\hat{\partial}_{a,b}\hat{\partial}_{c,d}\mathcal{A} \sim \hat{\partial}_{c,d}\hat{\partial}_{a,b}\mathcal{A}$. 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.

5.3 Context Dependency and Negative Guards

We now formally define the notion context-dependency. This has never been formalized for any of the other existing models of Reo.

[2] 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 $SyncSpout(a, b)$ plugged to a $SyncDrain(b', c)$ between b and b'.

Definition 11 (Firing Monotonic). *Let $\mathcal{A} = (\Sigma, Q, \delta)$ be a guarded automaton. $\mathcal{A}$ is* firing monotonic *if and only if for all states $q \in Q$ and for all $\alpha_1, \alpha_2 \in \mathbf{At}_\Sigma$ if $\alpha_1^+ \subseteq \alpha_2^+$, then $\mathbf{firings}(q, \alpha_1) \subseteq \mathbf{firings}(q, \alpha_2)$. That is, $\mathbf{firings}(q, \_)$ is monotonic for all $q \in Q$.*

Definition 12 (Context Dependent). *A guarded automaton $\mathcal{A}$ 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 $\alpha_1, \alpha_2 \in \mathbf{At}_\Sigma$ such that $\alpha_1^+ \subseteq \alpha_2^+$ and $\mathbf{firings}(q, \alpha_1) \not\subseteq \mathbf{firings}(q, \alpha_2)$. 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 $\mathbf{firings}(q, a\bar{b}) = \{(q, a)\} \not\subseteq \{(q, ab)\} = \mathbf{firings}(q, ab)$, whereas the state p does not.

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

Lemma 5. *Let $\mathcal{A}$ be a guarded automaton for which no negative atoms appear in the guards. Then $\mathcal{A}$ is firing monotonic.*

Lemma 6. *Firing monotonicity is preserved by product and synchronisation.*

Constraint automata [7] can be embedded in a natural way into our model by transforming every transition labelled by F into a transition labelled by $\hat{F}|F$. As a consequence of the previous lemmas, this makes explicit the fact that constraint automata do not exhibit context dependent behaviour.

In addition we have, for Reo automata:

Lemma 7. *A firing monotonic Reo automaton is firing upclosed.*

The LossySync channel is not firing monotonic, yet it is firing upclosed.

5.4 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.

Definition 13 (Enabledness). *Let $\mathcal{A} = (\Sigma, Q, \delta)$ be a guarded automaton. A port $a \in \Sigma$ is* enabled *in a state q if for all $\alpha \in \mathbf{At}_\Sigma$ such that $\alpha \leq a$, (1) $\mathbf{firings}(q, \alpha) \neq \emptyset$ and (2) for all $(f, \_) \in \mathbf{firings}(q, \alpha)$ we have $a \in f$.*

Intuitively, a port a is enabled whenever all request sets containing a match some guard g *and* a subsequently fires. Including negative information in the definition of product (using $q^\sharp$) preserves *enabledness* through product.

Lemma 8. *Let $\mathcal{A}_1 = (\Sigma_1, Q_1, \delta_1)$ and $\mathcal{A}_2 = (\Sigma_2, Q_2, \delta_2)$ be guarded automata with $\Sigma_1 \cap \Sigma_2 = \emptyset$. Assume that in $\mathcal{A}_1$ the port $a \in \Sigma_1$ is enabled in state $q \in Q_1$. Then in $\mathcal{A}_1 \times \mathcal{A}_2$, the port a is enabled in all states (q, q'), where $q' \in Q_2$.*

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

5.5 Justification of the $g \not\leq \overline{a}\overline{b}$ Condition in $\partial_{a,b}$

The LossySync-Fifo1 example (Fig. 5) alone motivated the research into context-dependent models. When the Fifo buffer is empty, data must flow through the 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.

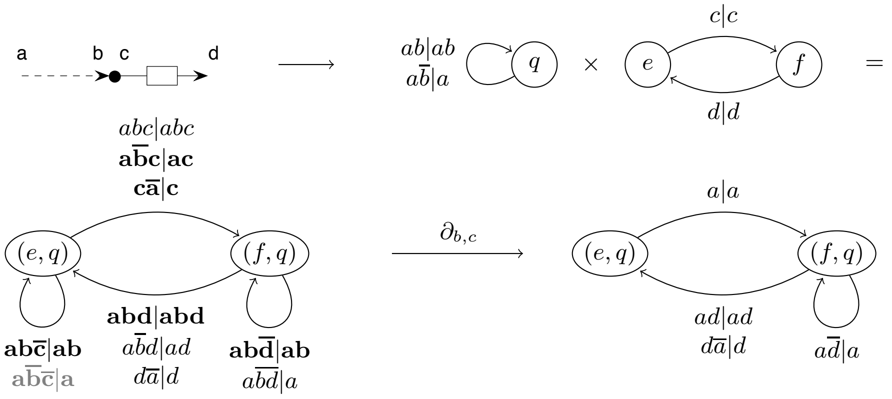


Fig. 5. LossySync-Fifo1

Definition 14. *Let $\mathcal{A} = (\Sigma, Q, \delta)$ be a guarded automaton. We say that a port $a \in \Sigma$ is (q, R)-sensitive for state $q \in Q$ and request set $R \subseteq \Sigma$ whenever $a \in f$ for all $(f, \_) \in$ **firings**$(q, \alpha_{R\cup\{a\}})$.*

This property holds for port b in $LossySync(a, b)$ in the request set $\{a\}$, and for port c in $Fifo1(c, d)$ in state *empty* for all request sets. In contrast, port a of $Merge(ab, c)$ is not sensitive for request set $\{b, c\}$.

The following lemma captures the property underlying the LossySync-Fifo1 example:

Lemma 9. *Let $\mathcal{A} = (\Sigma, Q, \delta)$ be a Reo automaton, $a, b \in \Sigma$, $q \in Q$, and $R \subseteq \Sigma$ a request such that $a, b \notin R$. If a is $(q, R \cup \{b\})$-sensitive and b is $(q, R \cup \{a\})$-sensitive, then* $\mathbf{firings}_{\partial_{a,b}\mathcal{A}}(q, \alpha_R) = \{(f \setminus \{a, b\}, q') \mid (f, q') \in \mathbf{firings}_{\mathcal{A}}(q, \alpha_{R\cup\{a,b\}})\}$.

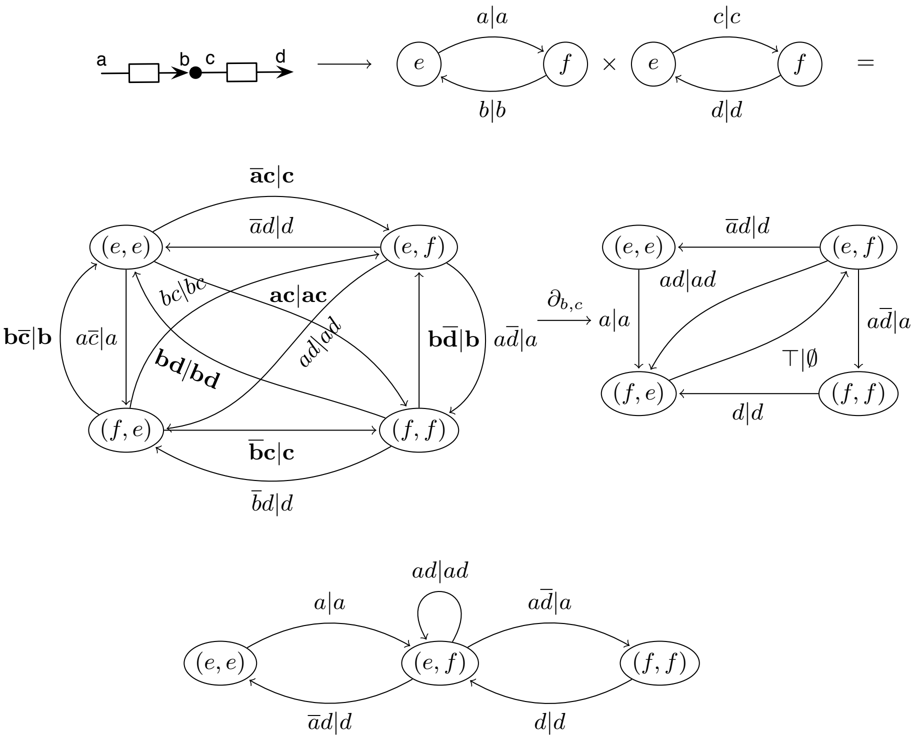


Fig. 6. Two Fifo1 buffers plugged together, their automaton, and the result of performing 'hiding'—a Fifo2 buffer

This 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 both do not fire. Constraint automata [7] would include both.

We believe that this kind of analysis is only the beginning in the key issue of more deeply understanding the interaction between synchronisation and context dependency [11,19,12].

5.6 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, 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.

5.7 'Hiding'

Constraint automata [7] models of Reo include a 'hiding' operation, which compresses τ transitions in the automata, which are transitions labelled by $\top|\emptyset$ in our model. See Figure 6. This can be used to obtain an automaton for a FIFO2 channel from the composite of two FIFO1 channels. The alternative variant defined by Costa [12] is equally applicable, and perhaps more robust.

6 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.

In this paper, we have not taken into account the actual data flowing through the connectors. This was in order to not distract the reader from the actual novelty of the paper. In fact, data constraints form a boolean algebra and can be added exactly in the same way as we have dealt with guards. Moreover, our model can be used to give a significantly simpler account of quantitative Reo [4]. At present, we are incorporating our automata model into CWI's *Eclipse Coordination Tools*[3] This will enable the generation of Java implementations of our automata for composing components and services.

Kleene algebra with tests [22] (KAT) are to guarded automata what regular expressions are to ordinary finite automata. Therefore, we want to explore how KAT expressions can be used to specify and synthesize Reo connectors. This will give us an algebraic description of Reo connectors, for which reasoning can be automated. More generally, since our automata can be seen as ordinary labelled transition systems with structured labels, we are interested in the connection with temporal logic and model checking.

References

1. Arbab, F.: Reo: a channel-based coordination model for component composition. Mathematical Structures in Computer Science 14(3), 329–366 (2004)
2. Arbab, F.: Abstract behavior types: a foundation model for components and their composition. Sci. Comput. Program. 55(1-3), 3–52 (2005)

[3] `http://reo.project.cwi.nl/`

3. Arbab, F., Bruni, R., Clarke, D., Lanese, I., Montanari, U.: Tiles for Reo. In: WADT (2009) (to appear)
4. Arbab, F., Chothia, T., van der Mei, R., Meng, S., Moon, Y., Verhoef, C.: From Coordination to Stochastic Models of QoS. In: Field, J., Vasconcelos, V.T. (eds.) COORDINATION 2009. LNCS, vol. 5521, pp. 268–287. Springer, Heidelberg (2009)
5. Arbab, F., Herman, I., Spilling, P.: An overview of Manifold and its implementation. Concurrency - Practice and Experience 5(1), 23–70 (1993)
6. Arbab, F., Rutten, J.: A coinductive calculus of component connectors. In: Wirsing, M., Pattinson, D., Hennicker, R. (eds.) WADT 2003. LNCS, vol. 2755, pp. 34–55. Springer, Heidelberg (2003)
7. Baier, C., Sirjani, M., Arbab, F., Rutten, J.: Modeling component connectors in Reo by constraint automata. Sci. Comput. Program. 61(2), 75–113 (2006)
8. Barbosa, L., Barbosa, M.: A perspective on service orchestration. In: Science of Computer Programming (2008) (accepted for publication)
9. Barbosa, M., Barbosa, L., Campos, J.: Towards a coordination model for interactive systems. Electr. Notes Theor. Comput. Sci. 183, 89–103 (2007)
10. Clarke, D., Proença, J., Lazovik, A., Arbab, F.: Deconstructing Reo. In: FOCLASA 2008 (2008) (to appear)
11. Clarke, D., Costa, D., Arbab, F.: Connector colouring I: Synchronisation and context dependency. Sci. Comput. Program. 66(3), 205–225 (2007)
12. Costa, D.: Formal Models for Context Dependent Connectors for Distributed Software Components and Services. Ph.D thesis (2009) (to appear)
13. de Boer, F., Kok, J., Palamidessi, C., Rutten, J.: Non-monotonic concurrent constraint programming. In: ILPS, pp. 315–334 (1993)
14. Fiadeiro, J., Lopes, A.: Community on the move: Architectures for distribution and mobility. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2003. LNCS, vol. 3188, pp. 177–196. Springer, Heidelberg (2004)
15. Fournet, C., Gonthier, G.: The join calculus: A language for distributed mobile programming. In: Barthe, G., Dybjer, P., Pinto, L., Saraiva, J. (eds.) APPSEM 2000. LNCS, vol. 2395, pp. 268–332. Springer, Heidelberg (2002)
16. Gelernter, D.: Generative communication in Linda. ACM Trans. Program. Lang. Syst. 7(1), 80–112 (1985)
17. Scholten, J.: Mobile channels for exogenous coordination of distributed systems: semantics, implementation and composition. Ph.D thesis, LIACS, Faculty of Mathematics and Natural Sciences, Leiden University (January 2007)
18. Izadi, M., Bonsangue, M.: Recasting constraint automata into Büchi automata. In: Fitzgerald, J.S., Haxthausen, A.E., Yenigun, H. (eds.) ICTAC 2008. LNCS, vol. 5160, pp. 156–170. Springer, Heidelberg (2008)
19. Izadi, M., Bonsangue, M., Clarke, D.: Modelling component connectors: Synchronisation and context-dependency. In: Proceedings of SEFM 2008. IEEE Computer Society Press, Los Alamitos (2008) (to appear)
20. Khosravi, R., Sirjani, M., Asoudeh, N., Sahebi, S., Iravanchi, H.: Modeling and analysis of Reo connectors using Alloy. In: Lea, D., Zavattaro, G. (eds.) COORDINATION 2008. LNCS, vol. 5052, pp. 169–183. Springer, Heidelberg (2008)
21. Koehler, C., Arbab, F., de Vink, E.: Reconfiguring distributed Reo connectors. In: WADT (2009) (to appear)
22. Kozen, D.: On the coalgebraic theory of Kleene algebra with tests. TR 10173, Computing and Information Science, Cornell University (March 2008)

23. Lee, B., Lee, E.: Hierarchical concurrent finite state machines in Ptolemy. In: ACSD, pp. 34–40. IEEE Computer Society, Los Alamitos (1998)
24. Liu, X., Xiong, Y., Lee, E.: The Ptolemy ii framework for visual languages. In: HCC, p. 50. IEEE Computer Society, Los Alamitos (2001)
25. Maraikar, Z., Lazovik, A., Arbab, F.: Building mashups for the enterprise with SABRE. In: Bouguettaya, A., Krüger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 70–83. Springer, Heidelberg (2008)
26. Misra, J., Cook, W.: Computation orchestration: A basis for wide-area computing. Journal of Software and Systems Modeling (May 2006)
27. Mousavi, M., Sirjani, M., Arbab, F.: Formal semantics and analysis of component connectors in Reo. Electr. Notes Theor. Comput. Sci. 154(1), 83–99 (2006)
28. Nierstrasz, O.: Piccola - a small compositional language (invited talk). In: Ciancarini, P., Fantechi, A., Gorrieri, R. (eds.) FMOODS. IFIP Conference Proceedings, vol. 139. Kluwer, Dordrecht (1999)
29. Rutten, J.: Coalgebra, concurrency, and control. In: Boel, R., Stremersch, G. (eds.) Discrete Event Systems (analysis and control), Proceedings of WODES 2000, pp. 31–38. Kluwer, Dordrecht (2000)
30. Shaw, M., Garlan, D.: Software Architecture. Prentice Hall, Englewood Cliffs (1996)
31. Szyperski, C.: Component Software: Beyond Object-Oriented Programming, 2nd edn. Addison-Wesley Professional, Reading (2002)