A model of context-dependent component connectors

M.M. Bonsangue, D. Clarke, A.M. Silva

SEN-E0903
Centrum Wiskunde & Informatica (CWI) is the national research institute for Mathematics and Computer Science. It is sponsored by the Netherlands Organisation for Scientific Research (NWO). CWI is a founding member of ERCIM, the European Research Consortium for Informatics and Mathematics.

CWI’s research has a theme-oriented structure and is grouped into four clusters. Listed below are the names of the clusters and in parentheses their acronyms.

Probability, Networks and Algorithms (PNA)

Software Engineering (SEN)

Modelling, Analysis and Simulation (MAS)

Information Systems (INS)
A Model of Context-Dependent Component Connectors

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

\textsuperscript{a}LIACS, Leiden University, The Netherlands
\textsuperscript{b}Dept. Computer Science, Katholieke Universiteit Leuven, Belgium
\textsuperscript{c}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 a 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 that can be sold off-the-shelf and reused to build software systems [35]. 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 [30, 18, 32], 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, 20, 1, 16].

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, 30]. 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 [27], Ptolemy II [28], MoCha [20], Manifold [5], and pipe and filter architectures [34]. Although these

1998 ACM Computing Classification System: F.1.1
Keywords and Phrases: Reo; Semantics; automata
overlap in philosophy and functionality, Reo is the only one that enables syn-
chrony and mutual exclusion to propagate through connectors.

The trend is to increase (or improve) the expressiveness of such coordina-
tion models by investigating features such as dynamic reconfiguration [24], data
sensitive operations such as data filtering and transformation [12], 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 in-
vestigating the notion of context dependency in the setting of the coordination
language Reo [1]. Context dependency enables connectors to be more respon-
sive 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 con-
nectors 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 [15] and generative com-
unication [19], 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 is an extended version of our Coordination’09 ar-
ticle [10]. It presents a compositional automata model for expressing contex-
dependent connectors. Following intensional automata [14], 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 au-
томata 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 connec-
tors, plus a synchronisation operation. Composition propagates context inform-
ation, which contains both positive and negative information. Using this we
define a previously elusive notion of enabledness and show that it is also appro-
priately 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 [29] with the context dependency
originally developed in the connector colouring model [11]. 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.

2
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 thereby synchronisation and mutual exclusion constraints propagate through a connector.

A number of Reo’s primitive connectors are depicted in Figure 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 Figure 3.

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 de-
scription 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 inter-
action, 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 will-
ingness to accept data (takes) propagate through channels, but unfortunately,
this description is incomplete and imprecise, in particular with regard to how
countext propagation interacts with non-deterministic choice. Clarke et al. [12]
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 under-
stood 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 [14].

The second problem is that degenerate behaviour can arise in certain circumstances (see Section 6). 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 [14] 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 [21, 22] assigns to connectors infinite fair behaviours. In this model, $\tau$-transitions capture the arrival of requests, which are recorded in states. In this model, there are two different non-equivalent

\footnote{Our model also does not deal with causality issues; Costa’s fix is applicable here [14].}
ways of modelling something as simple as a Sync channel. Thus the model differs significantly from other approaches.

Mousavi et al. [31] 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 [23]. 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.

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 8. 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 $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ and $B_\Sigma$ be the free Boolean algebra generated by the following grammar:

$$g ::= \sigma \in \Sigma \mid \top \mid \bot \mid g \lor g \mid g \land g \mid \neg g$$
We refer to the elements of the above grammar as guards and in its representation we frequently omit \( \land \) and write \( g_1g_2 \) instead of \( g_1 \land g_2 \). Given two guards \( g_1, g_2 \in B_\Sigma \), we 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 \( \text{norm}(g) = \lor \land a \), where \( a \in \Sigma \cup \bar{\Sigma} \), with \( \bar{\Sigma} = \{ \sigma \mid \sigma \in \Sigma \} \), and \( \lor \) and \( \land \) 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 \( B_\Sigma \) is a guard \( a_1 \ldots a_k \) such that \( a_i \in \{ \sigma_i, \bar{\sigma}_i \} \), \( 1 \leq i \leq k \). We can think of an atom as a truth assignment. We denote atoms by Greek letters \( \alpha, \beta, \ldots \) and the set of all atoms of \( B_\Sigma \) by \( \text{At}_\Sigma \). Every element of a finite Boolean algebra can be written as a disjunction of atoms. Given \( S \subseteq \Sigma \), we define \( \widehat{S} \in B_\Sigma \) 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—\( \alpha_S = \widehat{S} \land \Sigma \setminus \widehat{S} \). For example, if \( \Sigma = \{ a, b, c \} \), then \( \alpha_{\{a,b\}} = ab\bar{c} \). 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 \( x = \langle \alpha_1, f_1 \rangle \langle \alpha_2, f_2 \rangle \ldots \langle \alpha_n, f_n \rangle \), where \( n \geq 0 \) and each \( \alpha_i \in \text{At}_\Sigma \) and \( f_i \subseteq \Sigma \). Thus, a guarded string is an element of \( (\text{At}_\Sigma \times 2^\Sigma)^* \). For simplicity, we drop the brackets and write \( x = \alpha_1 f_1 \alpha_2 f_2 \ldots \alpha_n f_n \).

To understand the intuition behind guarded strings, imagine that \( \Sigma \) contains the names of all doctors in a hospital. Every hour there is a meeting to distribute the incoming patients. Each atom \( \alpha_i \) 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 synchronisation operation is defined.

4.1 Definition (Guarded automaton). A guarded automaton over an alphabet of ports \( \Sigma \) is a non-deterministic (and possibly partial) automaton with transition labels \( B_\Sigma \times 2^\Sigma \). Formally, a guarded automaton is a triple \( (\Sigma, Q, \delta) \) where \( Q \) is a (finite) set of states and \( \delta \subseteq Q \times B_\Sigma \times 2^\Sigma \times Q \) 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 Figure 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 string is *accepted* in state $q$ if there exists $q \xrightarrow{g|f_1} q'$ 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 $a = ab \lor ab$), replace the transition $q \xrightarrow{g|f_1} q'$ by the transitions $q \xrightarrow{\alpha_i f_1} q'$ 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 determinises it using a subset construction, the resulting automata will have a transition function of type $Q \rightarrow (1 + Q)^{At \times 2^E}$ [26]. It is then well-known [33] that such automata have as final semantics precisely the non empty and prefix closed languages $L \subseteq 2^{At \times 2^E}$.

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

**4.2 Definition (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 $\langle q_1, q_2 \rangle \in R$:

1. For all $q_1 \xrightarrow{g|f} q'_1 \in \delta_1$ and $\alpha \in At_\Sigma$ 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 $\alpha \in At_\Sigma$ 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 $A_1$ and $A_2$ 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 $A_1 \sim A_2$. The automata depicted in the following figure are bisimilar.

4.3 Theorem. 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 $x \in L_{q_1} \Leftrightarrow x \in L_{q_2}$ 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 $x = \alpha_1 f_1 \alpha_2 f_2 \cdots \alpha_n f_n$.

\[
x \in L_{q_1} \Leftrightarrow \exists q_1 \xrightarrow{\alpha_1 f_1} q_1' \in \delta_1 \cdot \alpha_1 \leq \gamma \text{ and } \alpha_2 f_2 \cdots \alpha_n f_n \in L_{q_1'}
\]

\[
x \in L_{q_2} \Leftrightarrow \exists q_2 \xrightarrow{\alpha_1 f_1} q_2' \in \delta_2 \cdot \alpha_1 \leq \gamma' \text{ and } \alpha_2 f_2 \cdots \alpha_n f_n \in L_{q_2'}
\]

\[
x \in L_{q_1} \Leftrightarrow \exists q_2 \xrightarrow{\alpha_1 f_1} q_2' \in \delta_2 \cdot \alpha_1 \leq \gamma' \text{ and } \alpha_2 f_2 \cdots \alpha_n f_n \in L_{q_2'}
\]

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

4.4 Definition. Given a guarded automaton $A = (\Sigma, Q, \delta)$ and $q \in Q$ we define $q^\sharp = \neg \bigvee \{ g \mid q \xrightarrow{g} q' \in \delta \}$.

This captures precisely the conditions in which $A$ 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 $q^\sharp = \top$ (respectively, $q^\sharp = \bot$) then the state can never (respectively, always) inhibit the step of a state in another automaton, in the context of the product, defined below. For instance, in the automata

$$ab|a \quad ab|a \quad ab|ab$$
one has $q_1^t = \pi \lor b$ and $q_2^t = \pi$.

4.5 Definition (Product). Given two guarded automata $A_1 = (\Sigma_1, Q_1, \delta_1)$ and $A_2 = (\Sigma_2, Q_2, \delta_2)$ such that $\Sigma_1 \cap \Sigma_2 = \emptyset$, we 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

$$
\delta = \{(q, p) \xrightarrow{gg} (q', p') \mid q \xrightarrow{g} q' \in \delta_1 \text{ and } p \xrightarrow{f} p' \in \delta_2\} \quad (1)
\cup \{(q, p) \xrightarrow{gp} (q, p') \mid q \xrightarrow{g} q' \in \delta_1 \text{ and } p \in Q_2\} \quad (2)
\cup \{(q, p) \xrightarrow{gp} (q, p') \mid p \xrightarrow{g} p' \in \delta_2 \text{ and } q \in Q_1\} \quad (3)
$$

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 $p^f$ and $q^f$, respectively).

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

\[
\begin{array}{c}
\begin{array}{c}
ab\ab
\end{array}
\times
\begin{array}{c}
\begin{array}{c}
\text{cd|cd, cd|c}
\end{array}
\end{array}
\end{array}
\Rightarrow
\begin{array}{c}
\begin{array}{c}
\text{abcd|abcd}
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\text{cd|cd, cd|c}
\end{array}
\end{array}
\end{array}
\]

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 $(q_1, q_2)^t = q_1^t \land q_2^t$.

4.6 Theorem. 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 $(q_1, q_2) \in Q_1 \times Q_2$,

$$(q_1, q_2)^t = q_1^t \land q_2^t$$

Proof. Let $G_1 = \{q_1 \mid q_1 \xrightarrow{g_1} \}$ and $G_2 = \{q_2 \mid q_2 \xrightarrow{g_2} \}$. Note that $q_1^t = \neg \forall G_1$ and $q_2^t = \neg \forall G_2$. The result follows by formulae manipulation.
such that not uniform, because Reo automaton (in fact, it models a FIFO1 channel). The first automaton is guarded automaton \((\Sigma, Q, \delta)\) such that for each \(q \xrightarrow{g|f} q'\) 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 \(\alpha\), 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.

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 \(\alpha\), 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.

4.7 Definition (Reo automaton). A Reo automaton over an alphabet \(\Sigma\) is a guarded automaton \((\Sigma, Q, \delta)\) such that for each \(q \xrightarrow{g|f} q'\) \(q' \in \delta\):

- \(g \leq \hat{f}\) (reactivity)

- \(\forall g \leq g' \leq \hat{f} \cdot \forall \alpha \leq g' \cdot \exists q \xrightarrow{g''|f} q' \in \delta \cdot \alpha \leq g''\) (uniformity)

Among the guarded automata depicted in Figure 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 \(ab \leq g\). The second automaton in not reactive: \(a \bar{b} \not\leq ab\).

In Figure 3 we depict the guarded automata for the basic channel types listed in Figure 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.

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

Proof. Given \( A_1 = (\Sigma_1, Q_1, \delta_1) \) and \( A_2 = (\Sigma_2, Q_2, \delta_2) \) Reo automata, we want to show that the automaton \( A_1 \times A_2 = (\Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, \delta) \) is also reactive and uniform, that is for every transition \( q \xrightarrow{g} q' \in \delta \), \( g \leq \hat{f} \) and for all \( g \leq g' \leq \hat{f} \) and \( \alpha \leq g' \) there exists \( \alpha \leq g'' \). 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 \( (q_1, q_2) \xrightarrow{g_1, g_2/f_1, f_2} (q_1', q_2') \in \delta \). Because both \( A_1 \) and \( A_2 \) are reactive we know that \( q_1 \leq \hat{f}_1 \) and \( q_2 \leq \hat{f}_2 \). Thus, \( g_1, g_2 \leq \hat{f}_1, \hat{f}_2 = \hat{f}_1, \hat{f}_2 \).

For uniformity, the most interesting cases are (2) and (3) in the definition of product. We illustrate case (2). Let \( (q_1, q_2) \xrightarrow{g_1, g_2/f_1} (q_1', q_2') \in \delta \). Now take any \( g' \) such that \( g_1, g_2 \leq g' \leq \hat{f} \) and \( \alpha \leq g' \). The guard \( g' \) can be divided into two guards \( g'_1 \in \mathcal{L}_{\Sigma_1} \) and \( g'_2 \in \mathcal{L}_{\Sigma_2} \) such that \( g' = g'_1 g'_2 \), with \( g_1 \leq g'_1 \leq \hat{f}_1 \) and \( g_2 \leq g'_2 \). Because \( A_1 \) is uniform, we know that for all \( \alpha \leq g'_1 \) there exists \( q_1 \xrightarrow{g'_1} q_1' \in \delta_1 \) such that \( \alpha_1 \leq g''_1 \) (and \( \alpha \leq g''_1 \)). We also know that either \( \alpha \leq g'_2 \) or \( \alpha \leq g_2 \) for some \( g_2 \xrightarrow{g_2/f_2} q'_2 \). In the first case, there exists a transition \( (q_1, q_2) \xrightarrow{g_1, g'_2/f_1} (q_1', q_2') \in \delta \) with \( \alpha \leq g_1' g'_2 \). In the second case, there exists a transition \( (q_1, q_2) \xrightarrow{g_2} (q_1', q_2') \in \delta \) with \( \alpha \leq g_1' g_2' \). \( \Box \)
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 Figure 3 for basic Reo channels this is already the case.

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

\[
\text{norm}(\delta) = \{ q \xrightarrow{g_f} q' \mid q \xrightarrow{g_f} q' \in \delta \text{ and } g' \in \text{norm}(g) \}
\]

4.10 Lemma. 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 \( q \xrightarrow{g_f} q' \in \text{norm}(\delta) \) and for all \( g \leq g' \leq \hat{f} \) and \( \alpha \leq g' \) there exists \( q \xrightarrow{g''_f} q'' \in \text{norm}(\delta) \) such that \( \alpha \leq g'' \). Let \( q \xrightarrow{g''_f} q'' \in \text{norm}(\delta) \). We know that there exists \( q \xrightarrow{g''_f} q'' \) such that \( g'' \in \text{norm}(g) \). Because the original automaton is uniform we know that

\[
\forall g \leq g_1 \leq \hat{f} \forall \alpha \leq g_1 \exists q \xrightarrow{g''_f} q'' \in \delta \cdot \alpha \leq g''
\]

We have

\[
g \leq g_1 \leq \hat{f} \Leftrightarrow \forall g' \in \text{norm}(g) g' \leq g_1 \leq \hat{f} \quad \text{and} \quad \alpha \leq g'' \Leftrightarrow \exists g_2 \in \text{norm}(g'') \cdot \alpha \leq g_2.
\]

Therefore,

\[
\forall g' \leq g_1 \leq \hat{f} \forall \alpha \leq g_1 \exists q \xrightarrow{g''_f} q'' \in \text{norm}(\delta) \cdot \alpha \leq g_2
\]

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

First, take any \( q \xrightarrow{g_f} q' \in \delta \) and \( \alpha \leq g \). Now note that

\[
\alpha \leq g \Leftrightarrow \alpha \leq \text{norm}(g) \Leftrightarrow \exists g' \in \text{norm}(g) \cdot \alpha \leq g'
\]

Thus, by the definition of \( \text{norm}(\delta) \), there exists a transition \( q \xrightarrow{g'_{f_1}} q' \in \text{norm}(\delta) \), such that \( \alpha \leq g' \) and \( \langle q', q'' \rangle \in R \).

Conversely, take any transition \( q \xrightarrow{g'_{f_1}} q' \in \text{norm}(\delta) \) 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 \( q \xrightarrow{g_f} q' \in \delta \) such that \( \alpha \leq g \) and \( \langle q', q'' \rangle \in R \). \( \square \)
Now we are ready to define the synchronisation 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 $b$ synchronise. In order to propagate context information (requests), we require that the guard contains either $a$ or $b$, expressed by the condition $g \not\leq \overline{a\,b}$, 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 6.5).

4.11 Definition (Synchronisation). Given a guarded automaton $A = (\Sigma, Q, \delta)$. We define the synchronisation of $a$ and $b$ ($a, b \in \Sigma$) as $\partial_{a:b}A = (\Sigma, Q, \delta')$ where

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

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

4.12 Lemma. 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 $q \xrightarrow{gf} q' \in \delta'$ and for all $g \leq g' \leq \hat{f}$ and $\alpha \leq g'$ there exists $q \xrightarrow{\alpha\,f}$, $q' \in \delta$ such that $\alpha \leq g''$.

Take a transition $q \xrightarrow{g\setminus ab[f\setminus\{a,b\}]} q'$ in $\delta'$. We know that $g \not\leq \overline{a\,b}$ and that $a \in f \Leftrightarrow b \in f$. Thus,

$$g\setminus ab \leq g' \leq f \setminus \{a, b\} \iff g \leq g' \leq \hat{f} \text{ or } g \leq g'ab \leq \hat{f}$$

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

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

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 $\partial$ 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.
4.13 Lemma. 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. $\partial_{a,c}(A_1 \times \text{Sync}(a, b)) \sim A_1[b/c]$, if $a, b \notin \Sigma_1$ and $c \in \Sigma_1$.

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

**Proof.** Property 1 follows easily from the definition. For 2., first, observe that

$$\partial_{a,b}A_1 = \{ (q, g') \mapsto (q', p') | q \xrightarrow{g} g' \in \text{norm}(\delta_1), a \in f \Leftrightarrow b \in f \text{ and } g \not\in \overline{p} \}$$

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

$$\delta = \{ (q, p) \xrightarrow{g|a|g|f|a|b|f'} (q', p') | q \xrightarrow{g|f} g' \in \text{norm}(\delta_1), p \xrightarrow{g'|f'} p' \in \delta_2, a \in f \Leftrightarrow b \in f \text{ and } g \not\in \overline{p} \}$$

(1)

$$\cup \{ (q, p) \xrightarrow{g|a|p|f|a|b|f} (q', p) | q \xrightarrow{g|f} g' \in \text{norm}(\delta_1), p \in Q_2, a \in f \Leftrightarrow b \in f \text{ and } g \not\in \overline{p} \}$$

(2)

$$\cup \{ (q, p) \xrightarrow{g|b|f} (q, p') | p \xrightarrow{g|f} p' \in \delta_2, q \in Q_1, q \notin \overline{p} \}$$

(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

$$\delta = \{ (q, p) \xrightarrow{g|a|g|f|a|b|f'} (q', p') | q \xrightarrow{g|f} g' \in \text{norm}(\delta_1), p \xrightarrow{g'|f'} p' \in \delta_2, a \in f \Leftrightarrow b \in f \text{ and } g \not\in \overline{p} \}$$

(1)

$$\cup \{ (q, p) \xrightarrow{g|a|p|f|a|b|f} (q', p) | q \xrightarrow{g|f} g' \in \text{norm}(\delta_1), p \in Q_2, g' \in \text{norm}(p'), a \in f \Leftrightarrow b \in f \text{ and } g \not\in \overline{p} \}$$

(2)

$$\cup \{ (q, p) \xrightarrow{g|b|f} (q, p') | p \xrightarrow{g|f} p' \in \delta_2, q \in Q_1, q \notin \overline{p} \}$$

(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:

$$(\partial_{a,b}A_1) \times A_2 \sim \partial_{a,b}(A_1 \times A_2)$$

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

$$\delta' = \{ (q, \cdot) \xrightarrow{g|b|f} (q', \cdot) | q \xrightarrow{g|f} q' \in \delta, c \in f, f = f'c, g = g'c \}$$

(1)

$$\cup \{ (q, \cdot) \xrightarrow{g|c|f} (q', \cdot) | q \xrightarrow{g|f} q' \in \delta, c \not\in f \text{ and } g \not\in \tau \}$$

(2)

$$\cup \{ (q, \cdot) \xrightarrow{g|c|f} (q', \cdot) | q \xrightarrow{g|f} q' \in \delta, c \not\in f \}$$

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

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

Conversely, take any transition \((q, \cdot) \xrightarrow{g/f} q' \in \delta[b/c]\) and \(\alpha \leq g\). If \(b \in f\), there exists a transition \((q, \cdot) \xrightarrow{g'b/f} (q', \cdot) \in \delta'\) and \(\alpha \leq g' b = g\). If \(b \notin f\) and \(g \leq \overline{b}\), there exists a transition \((q, \cdot) \xrightarrow{(g \setminus b)/f} (q', \cdot) \in \delta'\) and \(\alpha \leq g \setminus b = b\). If \(b \notin f\) and \(g \notin \overline{b}\), there exists a transition \((q, \cdot) \xrightarrow{(g \setminus c)/f} (q', \cdot) \in \delta'\) and \(\alpha \leq g \setminus c\).

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

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 [25] 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 [25] 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 \(\text{LossySync}(a, b)\) channel:

\[
\begin{array}{c}
\{a, b\} \\
\circ \\
\{a\}
\end{array}
\]
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, \( \mathcal{P}(X) \), where \( X \) is the set of ports of the connector, and \( \land = \cup, \lor = \cap, \tau = X \setminus \cdot \), as usual. Each transition of such an automaton has the form \( q \xrightarrow{A|B} q' \), where \( A \subseteq \mathcal{P}(X) \) and \( B \in \mathcal{P}(X) \). 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:

\[
\{\{a, b\}\}|\{a, b\} \xrightarrow{q_1} \{\{a\}\}|\{a\}
\]

On the other hand, a non-context-dependent LossySync channel, as represented by the port automaton above, would be modelled as:

\[
\{\{a, b\}\}|\{a, b\} \xrightarrow{q_1} \{\{a, b\}, \{a\}\}|\{a\}
\]

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

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

**uniformity** for all \( C \in \mathcal{P}(X) \) such that \( B \subseteq C \subseteq A_0 \in A \), there is also a transition \( q \xrightarrow{A'|B} q' \) in the automaton such that \( C \in A' \).

### 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 describes both the ports of a connector that synchronise along with data flow at those ports. Specifically, each transition includes 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 FIFO1 buffer can be represented by the following automata, where there is a state \( \text{full}(d) \) and pair of transitions for each \( d \in \text{Data} \):

\[
\{a\}, d_a = d
\]

\[
\text{empty} \xrightarrow{\text{full}(d)} \text{full}(d), \{b\}, d_b = d
\]
This automaton states, firstly, that in state 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 \(d\), and the automaton goes into state full \((d)\). Similarly, in state 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 \(d\). Note that there will be one transition for each \(d\) with start state empty, whereas there is only one transition with start state full \((d)\). So even though the constraint on both transitions are of the same shape, this automaton does capture the behaviour of a FIFO1 buffer, as when the channel is full in state 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 \(Data\), let \(X ⇀ Data\) denote the partial functions from \(X\) to \(Data\). This models the flow of data on the ports of the connector: if \(f:X \rightarrow Data\) and \(f(x) = d\), where \(x \in X\) and \(d \in Data\), 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 \(q \xrightarrow{N,δ} q'\) where \(N \subseteq X\) and \(δ\) is a constraint over \(N\), describing the data flow on ports \(N\). We assume that \(δ\) is specified by the following grammar:

\[
δ = ⊤ | ⊥ | δ_1 \land δ_2 | δ \lor δ | ¬δ | d_a = d | d_a = d_b | P(d_a)
\]

where \(d_a\) represents the datum at port \(a\), \(d \in Data\), and \(P(−)\) is some monadic predicate over \(Data\), corresponding to a set \(P_I \subseteq Data\).

We interpret each of these in the power set Boolean algebra \(P(X \rightarrow Data)\) as follows:

\[
I(⊤) = P(X \rightarrow Data)
\]
\[
I(⊥) = ∅
\]
\[
I(δ_1 \land δ_2) = I(δ_1) ∪ I(δ_2)
\]
\[
I(δ_1 \lor δ_2) = I(δ_1) \cap I(δ_2)
\]
\[
I(¬δ) = P(X \rightarrow Data) \setminus I(δ)
\]
\[
I(d_a = d) = \{ f \in X \rightarrow Data \mid f(a) = d \}
\]
\[
I(d_a = d_b) = \{ f \in X \rightarrow Data \mid f(a) = f(b) \}
\]
\[
I(P(d_a)) = \{ f \in X \rightarrow Data \mid f(a) \in P_I \}
\]

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

\[
I(\text{just}(N)) = \{ f \in X \rightarrow Data \mid \text{dom}(f) = N \}
\]

The meaning of the label on a transition \(q \xrightarrow{N,δ} q'\) is given by the set \(\text{just}(N) \cap I(δ)\). 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 $q \xrightarrow{g|f} q'$ where $g \in \mathbb{P}(X \rightharpoondown Data)$ (or alternatively, in as an expression in the Boolean algebra), and $f \in X \rightharpoondown Data$ such that $f \in g$ (reactivity). (Uniformity is as above for port automata.)

This, however, is not a particularly compact representation. Instead we can keep $g$ as an element of the Boolean algebra and allow $f$ to be a pair $N, \delta$ as in the original constraint automata model. Reactivity requires that just $(N) \cap I(\delta) \subseteq g$—where $g$ is considered as an element of $\mathbb{P}(X \rightharpoondown Data)$.

In addition, we can also use the notions defined here to better model interaction with the components connected to a connector. These issue a requests to connectors on its ports. In general, two basic kinds of requests are possible, depending upon whether the port is an input or output ports: a request to write a particular datum $d$ to port $x$ is represented as request $\{x \mapsto d\}$; and a request to take some datum is $\{x \mapsto d\}$ ($d \in Data$). Other variants, such as a request to write either $d_1$ or $d_2$, or a request for some datum satisfying predicate $P(-)$, are denoted $\{x \mapsto d_1\}, \{x \mapsto d_2\}$ and $\{x \mapsto d\}$ ($d \in P_I$), respectively.

This example shows how to incorporate data into our framework.

6. 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 synchronisation step because the condition $b \in f \iff c \in f$ fails, and we mark in gray the transitions that are removed because $g \leq \overline{b}c$.

The following definition will come in handy.

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

$$\text{firings}_A(q, \alpha) = \{ (f, q') \mid q \xrightarrow{g|f} q' \in \delta \land \alpha \leq g \}.$$ 

We will drop the subscript $A$ whenever the automaton is clear from the context.

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 a synchronous channel gives an automaton which should be $\text{Loser}(a, d)$ if Sync behaved like the identity:
Loser(a, b) = (q_1 \cdots a|a) \quad \partial_{b,c}(Loser(a, b) \times Sync(c, d)) = (q_1, q_1) \cdots a|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 LossySync(a, b) × Sync(c, d) is:

\[
\begin{array}{c}
\text{ab|ab} \\
\text{a|b|a}
\end{array} \quad \times \quad \begin{array}{c}
\text{cd|cd} \\
\text{c|d|a}
\end{array} = \begin{array}{c}
\text{abcd|abcd} \\
\text{ab|cd|d}
\end{array}
\]

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

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

6.2. 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.

6.2 Definition (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 \(\alpha \in \text{At}_\Sigma\), \(\text{firings}(q, \alpha) \neq \emptyset\). ♦

The presentation of connector colouring [11] requires that the colouring tables are total. Unfortunately, composition does not preserve totality. Consider the Rep-AsyncDrain in Figure 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 (Figure 4), which behaves neutrally in the composition (since \((q_1, q_2)^2 = \top\)).

We also find it unnecessary to specify any behaviour that does not result in a firing (though we do permit \(\tau\)-transitions, represented by \(\top|\emptyset\)). The following definition captures a sensible notion, which is weaker than totality. It states that if some request set \(\alpha\) causes a firing, then all larger request sets also cause a firing (though not necessarily the same one).

6.3 Definition (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 \(\alpha \in \mathcal{A}_\Sigma\), if \(\text{firings}(q, \alpha) \neq \emptyset\), then for all \(\alpha_1\) such that \(\alpha^+ \subseteq \alpha_1^+\) we have \(\text{firings}(q, \alpha_1) \neq \emptyset\).

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 \(\partial_{b,b'}|\partial_{c,c'}|\text{PriorityMerger}(ab, c) \times \text{Rep}(c', b'd)\) and its accompanying automaton, where \(a\) is the higher priority port:

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, \( \partial \), which preserved Firing upclosed. Unfortunately, all our attempts failed to satisfy the required equivalence \( \partial_{a,b} \partial_{c,d} A \sim \partial_{c,d} \partial_{a,b} A \). 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.

6.4 Definition (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, \( \text{firings}(q, \cdot) \) is monotonic for all \( q \in Q \).

6.5 Definition (Context Dependent). 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 \( \text{firings}(q, \alpha_1) \nsubseteq \text{firings}(q, \alpha_2) \). 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 \( \text{firings}(q, ab) = \{ (q, a) \} \nsubseteq \{ (q, ab) \} = \text{firings}(q, ab) \), whereas the state \( p \) does not.

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

6.6 Lemma. 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
\[
\alpha_1 \leq g \iff \hat{g} \subseteq \alpha_1^+ \Rightarrow g \subseteq \alpha_2^+ \iff \alpha_2 \leq g \quad (1)
\]
Then, we reason
\[
\text{firings}(q, \alpha_1) = \{ (f, q') \mid q \xrightarrow{g} q' \text{ and } \alpha_1 \leq g \} \quad \text{(def. of firings)}
\subseteq \{ (f, q') \mid q \xrightarrow{g} q' \text{ and } \alpha_2 \leq g \} \quad \text{(by (1))}
= \text{firings}(q, \alpha_2) \quad \text{(def. of firings)}
\]

\( \square \)
It is interesting to remark that firing monotonicity is not preserved by product. As a counterexample consider the product of the automata corresponding to two FIFO channels \(FIFO(a, b) \times FIFO(c, d)\). The original automata are firing monotonic whereas the product automaton is not (the automaton appears in Figure 6).

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:

6.7 Lemma. 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.

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.

6.8 Definition (Enabledness). Let \(A = (\Sigma, Q, \delta)\) be a guarded automaton. A port \(a \in \Sigma\) is enabled in a state \(q\) if for all \(\alpha \in \text{At}_{\Sigma}\) such that \(\alpha \leq a\), (1) \(\text{firings}(q, \alpha) \neq \emptyset\) and (2) for all \((f, \_ \_) \in \text{firings}(q, \alpha)\) 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 \(q^\#\)) preserves enabledness through product.

6.9 Lemma. Let \(A_1 = (\Sigma_1, Q_1, \delta_1)\) and \(A_2 = (\Sigma_2, Q_2, \delta_2)\) be guarded automata with \(\Sigma_1 \cap \Sigma_2 = \emptyset\). Assume that in \(A_1\) the port \(a \in \Sigma_1\) is enabled in state \(q \in Q_1\). Then in \(A_1 \times A_2\), 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 \((q, q') \xrightarrow{gg' \downarrow f'} (q_1, q'_1) \in \delta_1 \times \delta_2\). For condition 2, the proof follows by case analysis of the product definition. The most interesting case is the third clause, \((q, q') \xrightarrow{g^\# \downarrow f'} (q, q'_1)\). Here, the key observation is that if \(\alpha \leq g\) for some \(q \xrightarrow{g \downarrow} q_1 \in \delta_1\) then \(\alpha \not\leq q^\#\) and thus \((f', (q, q''_1)) \notin \text{firings}((q, q'), \alpha)\). □
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 $q^2$ from the definition of product, thus taking the naive definition of product $(\times)$ following the definition in constraint automata directly, then $a$ is no longer enabled in LossySync$(a, b) \times$ Sync$(c, d)$, because a transition with guard $cd|ed$ is present in the resulting automaton. This transition matches request set $acd$, but $a$ does not fire.

6.5. Justification of the $g \not\leq \alpha_b$ condition in $\partial_{a,b}$

The LossySync-FIFO1 example (Figure 5) alone motivated the research into context-dependent models. When the FIFO1 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.

![Diagram of LossySync-FIFO1](image)

Figure 5: LossySync-FIFO1

6.10 Definition. Let $A = (\Sigma, Q, \delta)$ 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 $(f, \cdot) \in$ firings$(q, \alpha_{R,(a)})$ and firings$(q, \alpha_{R,(a)}) \neq \emptyset$.

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:

6.11 Lemma. Let $A_i = (\Sigma_i, Q_i, \delta_i)$ be Reo automata, for $i \in \{1, 2\}$, with $\Sigma_1 \cap \Sigma_2 = \emptyset$, and $a_i \in \Sigma_i$, $q_i \in Q_i$, $R_i \subseteq \Sigma_i$, such that $a_i \not\in R_i$. If $a_i$ is
$(q_i, R_i)$-sensitive, for $i \in \{1, 2\}$, then
\[
\text{firings}_{\partial a_1 \cup \{a_2\}, \alpha R_2}((q_1, q_2), \alpha R_1 \cup \{a_1, a_2\}) = \\
\{ (f \setminus \{a_1, a_2\}, q') \mid (f, q') \in \text{firings}_{A_1 \times A_2}((q_1, q_2), \alpha R_1 \cup \{a_1, a_2\}) \}
\]

**Proof.** First, note that
\[
\delta_{\partial a_1 \cup \{a_2\}, \alpha R_2}((q_1, q_2)) = \{ (q_1, q_2) \xrightarrow{g_1 f_1} (q_1', q_2') \mid q_1 \xrightarrow{g_1 f_1} q_1' \in \delta_1, q_2 \xrightarrow{g_2 f_2} q_2' \in \delta_2, g_1 g_2 \not\leq \alpha R_2 \}
\]
This is a direct consequence of sensitivity: since $a_i \in f_i$ for $q_i \xrightarrow{g_i f_i} q_i' \in \delta_i$ ($i = 1, 2$), 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 $\delta_{a_1 \cup \{a_2\}}$, by the condition $a_1 \in f \iff a_2 \in f$. Thus, we have:
\[
\text{firings}_{\partial a_1 \cup \{a_2\}, \alpha R_2}((q_1, q_2), \alpha R_1 \cup \{a_1, a_2\}) = \\
\{ ((q_1', q_2), f_1 f_2 \setminus \{a_1, a_2\}) \mid q_1 \xrightarrow{g_1 f_1} q_1' \in \delta_1, q_2 \xrightarrow{g_2 f_2} q_2' \in \delta_2, \alpha R_1 \cup \{a_1, a_2\} \leq g_1 g_2 \}
\]
We then calculate for $\text{firings}_{A_1 \times A_2}((q_1, q_2), \alpha R_1 \cup \{a_1, a_2\})$:
\[
\text{firings}_{A_1 \times A_2}((q_1, q_2), \alpha R_1 \cup \{a_1, a_2\}) = \\
\{ ((q_1', q_2'), f) \mid ((q_1', q_2'), f_1 f_2 \setminus \{a_1, a_2\}) \mid q_1 \xrightarrow{g_1 f_1} q_1' \in \delta_1, q_2 \xrightarrow{g_2 f_2} q_2' \in \delta_2, g = g_1 g_2 \}
\]
This says that if both $a$ and $b$ are mutually enabled in the presence of request set $R_i$ 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, 22, 14].

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, 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 \( \tau \) 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 [14] 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 labeled with \( \top | f \). 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 Figure 6. It has four states, each representing whether the first or second FIFO is either full (\( f \)) or empty (\( e \)). Clearly, the synchronisation of ports \( f \) and \( g \) will result in a similar Reo automata 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 \( \tau \) 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 \( ((f, e), (f, e)) \) to either \( ((e, f), (f, e)) \) nor \( ((f, e), (e, f)) \). 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 [36]. 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 $\mathcal{L} \subseteq 2^A$ [33]. Thus, we will be providing a trace semantics for the original non-deterministic automata.

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

The guarded automata presented above are acceptors of non-empty, prefix-closed languages $\mathcal{L} \subseteq GS_\Sigma = 2^{(A_\Sigma \times 2^C)^*}$. 

Figure 6: Two FIFO1 buffers plugged together, their automaton, and the result of performing ‘hiding’—a FIFO1 buffer.
Note that the automata presented in this paper have labels in $\mathcal{L}_\Sigma \times 2^\Sigma$. 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 $(\Sigma, Q, \Delta)$ where

$$\Delta : Q \to (1 + Q)^{\mathcal{L}_\Sigma \times 2^\Sigma}.$$  

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

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

where $S' = \{ q' \mid \langle q, g, o, q' \rangle \in \delta, \alpha \leq g, q \in S \}$

Here, $\kappa_1 : 1 \to 1 + Q$ and $\kappa_2 : Q \to 1 + Q$ 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

$$\mathcal{L}_S = \bigcup_{q \in S} \mathcal{L}_q.$$  

As an example of determinisation consider the automaton representing the FIFO1 buffer. Determinisation would yield the following automaton (we do not draw undefined transitions):

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

$$\mathcal{L}_e = ((ab|a + \overline{a}b|a)(ab|b + \overline{b}b)|b)^\ast((ab|a + \overline{a}b|a) + \varepsilon)$$

$$\mathcal{L}_f = ((ab|b + \overline{b}b|b)(ab|a + \overline{a}b|a))^\ast((ab|b + \overline{b}b|b) + \varepsilon)$$

$$\mathcal{L}_{(e,f)} = (ab|a + \overline{a}b|a)\mathcal{L}_f + (ab|b + \overline{b}b|b)\mathcal{L}_e + \varepsilon$$

The equality $\mathcal{L}_{(e,f)} = \mathcal{L}_e + \mathcal{L}_f$ can be easily derived, using the axioms of regular expressions.
8. 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) [26] 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 synthesise 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.

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.


³http://reo.project.cwi.nl/


Centrum Wiskunde & Informatica (CWI) is the national research institute for mathematics and computer science in the Netherlands. The institute's strategy is to concentrate research on four broad, societally relevant themes: earth and life sciences, the data explosion, societal logistics and software as service.

Centrum Wiskunde & Informatica (CWI) is het nationale onderzoeksinstituut op het gebied van wiskunde en informatica. De strategie van het instituut concentreert zich op vier maatschappelijk relevante onderzoeksthema's: aard- en levenswetenschappen, de data-explosie, maatschappelijke logistiek en software als service.

Bezoekadres:
Science Park 123
Amsterdam

Postadres:
Postbus 94079, 1090 GB Amsterdam
Telefoon 020 592 93 33
Fax 020 592 41 99
info@cwi.nl
www.cwi.nl