A specification language for Reo connectors

Alexandra Silva[1]

Centrum Wiskunde & Informatica

Abstract. Recent approaches to component-based software engineering employ coordinating connectors to compose components into software systems. Reo is a model of component coordination, wherein complex connectors are constructed by composing various types of primitive channels. Reo automata are a simple and intuitive formal model of context- dependent connectors, which provided a compositional semantics for Reo.
In this paper, we study Reo automata from a coalgebraic perspective. This enables us to apply the recently developed coalgebraic theory of generalized regular expressions in order to derive a specification language, tailor-made for Reo automata, and sound and complete axiomatizations with respect to three distinct notions of equivalence: (coalgebraic) bisimilarity, the bisimulation notion studied in the original papers on Reo automata and trace equivalence. The obtained language is simple, but nonetheless expressive enough to specify all possible finite Reo automata. Moreover, it comes equipped with a Kleene-like theorem: we provide algorithms to translate expressions to Reo automata and, conversely, to compute an expression equivalent to a state in a Reo automaton.

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 [24]. Research on software composition plays a key role in this quest, as it offers flexible ways of plugging together components. Channel based-languages, where 'channels' or 'connectors' are used to compose components into a system [5, 13, 1, 12], play a prominent in the world of software composition. These 'languages' express various coordination patterns exhibiting combinations of synchronisation, mutual exclusion, non-deterministic choice, context-dependent and state-dependent behaviour. A number of component connector models exist, including Reo [1], Ptolemy [17, 18], MoCha [13], Manifold [2], BIP [6] and an algebra of stateless connectors [10].

In this paper, we focus on the coordination language Reo and in a particular semantic model thereof: Reo automata [8, 9]. We present a specification language for Reo automata, together with a Kleene-like theorem and sound and complete axiomatizations with respect to three notions of equivalence which enable algebraic reasoning on specifications. In order to achieve this, we make use of the coalgebraic view on systems.

In the last decades, coalgebra has arisen as a prominent candidate for a mathematical framework to specify and reason about computer systems. Coalgebraic modeling works, on the surface, as follows: the basic features of a system, such as non-determinism or probability, are collected and combined in the appropriate way, determining the type of the system. This type (formally, a functor) is then used to derive a suitable equivalence relation and a universal domain of behaviors, which allow to reason about equivalence of systems. The strength of coalgebraic modeling lies in the fact that many important notions are parameterized by the type of the system. Recently, in [23] the coalgebraic view on systems enabled the development of a framework wherein specification languages and axiomatizations can uniformly be derived for a large class of systems.

In this paper, we apply the general coalgebraic framework of [23] to Reo automata. The main contributions of the paper are the following:

1. A coalgebraic characterization of Reo automata and of the bisimulation considered in [8].
2. A tailor-made language to specify Reo automata.
3. An analogue of Kleene's theorem for Reo automata, yielding algorithms to convert expressions to equivalent automata and vice-versa.
4. A sound and complete axiomatization of the language with respect to three different types of equivalence (bisimilarity, trace semantics and the bisimulation considered in [8]).

The items $2. - 4.$ partially stem from the general framework of [23]. However, the only axiomatization derived from the general framework of [23] is that of bisimilarity. The other two are completely new.

The paper is organized as follows. In Section 2 we present Reo, Reo automata and basic notions of coalgebraic modeling. We conclude the section by showing how to recast Reo automata in coalgebraic terms. We proceed in Section 3 to presenting a language of regular expressions for Reo automata and in Section 4 an analogue of Kleene theorem in this setting. Section 5 contains sound and complete axiomatizations with respect to bisimilarity and trace equivalence. In Section 5.1, we tie the whole story with the notion of equivalence considered in the papers on Reo automata [8, 9], which we characterize coalgebraically, and we present a sound and complete axiomatization thereof. Section 6 contains an extensive discussion on future and related work.

2 Preliminaries

Reo Reo is a channel-based coordination model wherein so-called *connectors* are used to coordinate (i.e., control the communication among) components or services exogenously (from outside of those components and services). In Reo, complex connectors are compositionally built out of primitive channels. Channels are atomic connectors with exactly two ends, which can be either *source* or *sink* ends. Source ends accept data into, and sink ends dispense data out of their respective channels. Reo allows channels to be undirected, i.e., to have respectively two source or two sink ends.

Fig. 1. Some basic Reo channels

Figure 1 shows the graphical representations of some basic channel types. The Sync channel is a directed, unbuffered channel that synchronously reads data from its source end and writes it to its sink end. The LossySync channel behaves similarly, except that it does not block if the party at the sink end is not ready to receive data. Instead, it just loses the data item. FIFO1 is an asynchronous channel with a buffer of size one. The SyncDrain channel differs from the other channels in that it has two source ends (and no sink end). If there is data at both ends, this channel consumes (and loses) both items synchronously.

Channels can be joined together using nodes. A node can have one of three types: source, sink or mixed node, depending on whether all ends that coincide on the node are source, sink or a combination of both. Source and sink nodes, called *boundary nodes*, form the boundary of a connector, allowing interaction with its environment. Source nodes act as synchronous replicators, and sink nodes as mergers. A mixed node combines both behaviors by atomically consuming a data item from one sink end and replicating it to all of its source ends.

An example connector is depicted in Figure 2. It reads a data item from a, buffers it in a FIFO1 and writes it to d. The connector loses data items from a if and only if the FIFO1 buffer is already full. This construct, therefore, behaves as a connector called (overflow) LossyFIFO1.

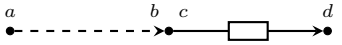


Fig. 2. LossyFIFO1

Semantics: Reo Automata In this section, we recall Reo Automata [9], an automata model that provides a compositional operational semantics for Reo connectors. Intuitively, a Reo Automaton is a non-deterministic automaton whose transitions have labels of the form $g|f$, where g is a *guard* (boolean condition) and f a set of nodes that fire synchronously. A transition can be taken only when its guard g is true.

We recall some facts about Boolean algebras. Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ be a set of symbols that denote names of connector ports, $\overline{\sigma}$ be the negation of σ, 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 $\leq$ as $g_1 \leq g_2 \iff g_1 \wedge g_2 = g_1$. The intended interpretation of $\leq$ is logical implication: g_1 implies g_2. An *atom* of $\mathcal{B}_\Sigma$ is a guard $a_1 \ldots a_k$ such that $a_i \in \Sigma \cup \overline{\Sigma}$ with $\overline{\Sigma} = \{\overline{\sigma}_i \mid \sigma_i \in \Sigma\}$, $1 \leq i \leq k$. 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 $\mathcal{B}_\Sigma$ by $\mathbf{At}_\Sigma$. Given $S \subseteq \Sigma$, we define $\widehat{S} \in \mathcal{B}_\Sigma$ as the conjunction of all elements of S. For instance, for $S = \{a, b, c\}$ we have $\widehat{S} = abc$.

Definition 1 (Reo Automaton [9]). *A* Reo Automaton *is a triple* (Σ, Q, δ) *where* Σ *is the set of nodes,* Q *is the set of states,* $\delta \subseteq Q \times \mathcal{B}_\Sigma \times 2^\Sigma \times Q$ *is the transition relation such that for each* $q \xrightarrow{g|f} q' \in \delta$*:*

(i) $g \leq \widehat{f}$ *(reactivity)*

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

In Reo Automata, for simplicity we abstract data constraints [4] and assume they are *true*. We use arrows $q \xrightarrow{g|f} q'$ for $\langle q, g, f, q' \rangle \in \delta$. If there is more than one transition from a state q to the same state q' we often just draw one arrow and separate their labels by commas. In Figure 3 we depict the Reo Automata for the basic channel types listed in Figure 1.

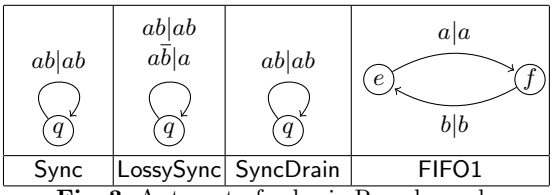


Fig. 3. Automata for basic Reo channels

Intuitively, a transition $q \xrightarrow{g|f} q'$ in an automaton corresponding to a Reo connector conveys the following notion: if the connector is in state q and the boundary requests present at the moment, encoded by an atom α, are such that $\alpha \leq g$, then the nodes f fires and the connector evolves to state q'. Each transition labeled by $g|f$ satisfies two criteria: (i) *reactivity*—data flow only through those nodes where a request is pending, capturing Reo's interaction model; and (ii) *uniformity*—which captures two properties: (a) the request set corresponding precisely to the firing set is sufficient to cause firing, and (b) removing additional unfired requests from a transition will not affect the (firing) behavior of the connector [9].

Composing Reo connectors We now model at the automata level the composition of Reo connectors. We define two operations: product, which puts two connectors in parallel, and synchronization, which models the plugging of two nodes. These two operations can be used to obtain the automaton of a Reo connector by composing the automata of its primitive connectors.

We first define the product operation for Reo Automata. This definition differs from the classical definition of (synchronous) product for automata: our automata have disjoint alphabets and they can either take steps together or independently. In the latter case the composite transition in the product automaton explicitly encodes that one of the two automata cannot perform a step in the current state, using the following notion:

Definition 2. *Given a Reo 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 condition under which $\mathcal{A}$ cannot fire in state q.

Definition 3 (Product). *Given two Reo 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* δ *consists of:*

$$\begin{aligned}
&\{(q,p) \xrightarrow{gg'|ff'} (q',p') \mid q \xrightarrow{g|f} q' \in \delta_1 \wedge p \xrightarrow{g'|f'} p' \in \delta_2\} \\
&\cup \{(q,p) \xrightarrow{gp^\sharp|f} (q',p) \mid q \xrightarrow{g|f} q' \in \delta_1 \wedge p \in Q_2\} \\
&\cup \{(q,p) \xrightarrow{gq^\sharp|f} (q,p') \mid p \xrightarrow{g|f} p' \in \delta_2 \wedge q \in Q_1\}
\end{aligned}$$

Here and throughout, we use ff' as a shorthand for $f \cup f'$. The first term in the union, above, applies when both automata fire in parallel. The other terms

apply when one automaton fires and the other is unable to (indicated by $p^\sharp$ and $q^\sharp$, respectively). Note that the product operation is closed for Reo Automata, since it preserves reactivity and uniformity [9]. Figure 4 shows an example of the product of two automata.

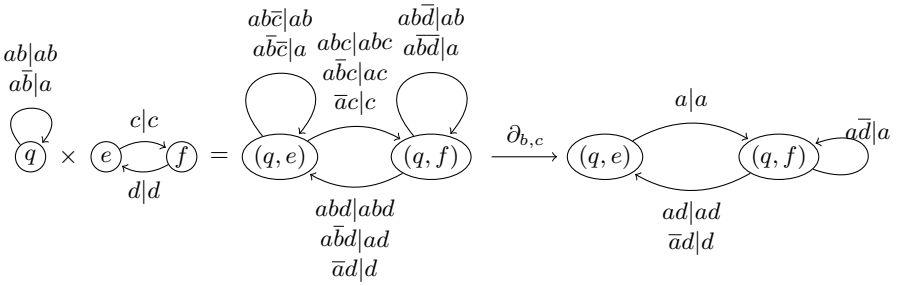


Fig. 4. Product of LossySync and FIFO1 and the synchronization of nodes b and c

We now define a synchronization operation that corresponds to joining two nodes in a Reo connector. In order for this operation to be well-defined we need that every guard in a transition label in the automata is a conjunction of literals. Note that in the automata presented in Figure 3 for basic Reo channels this is already the case. It is always possible to transform any guard g into this form, by taking its disjunctive normal form (DNF) $g_1 \vee \ldots \vee g_k$ and splitting the transition $g|f$ into the several $g_i|f$, for $i = 1, \ldots, k$. Given a transition relation δ we call $norm(\delta)$ the normalized transition relation obtained from δ by putting all of its guards in DNF and splitting the transitions as explained above.

When synchronizing two nodes a and b (which are then made internal), in the resulting automaton, only the transitions where either both a and b or neither a nor b fire are kept — this is what it means for a and b to synchronize. In order to propagate context information (pending requests), we require that every guard contains either a or b, expressed by the condition $g \not\leq \overline{a}\overline{b}$ below. This condition roughly corresponds to the notion of an internal node acting like a *self-contained pumping station* [1], which implies that an internal node cannot store data nor actively block behavior.

Definition 4 (Synchronization). *Given a Reo Automaton $\mathcal{A} = (\Sigma, Q, \delta)$, we define the* synchronization *for $a, b \in \Sigma$ as $\partial_{a,b}\mathcal{A} = (\Sigma, Q, \delta')$ where*

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

Here and throughout, $g\backslash_{ab}$ is the guard obtained from g by deleting all occurrences of a and b. It is worth noting that synchronization preserves reactivity and uniformity.

Figure 4 depicts the product of LossySync and FIFO1, together with the result of synchronizing nodes b and c. This synchronized result provides the semantics for the LossyFIFO1 example in Figure 2.

Compositionality Given two Reo Automata $\mathcal{A}_1$ and $\mathcal{A}_2$ over the 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 node a_i of the first connector is connected to node b_i of the second connector, for all $i \in \{1, \ldots, k\}$. Note that the 'plugging' order does not matter because ∂ is commutative and it interacts well with product. These properties are captured in the following lemma.

Lemma 1 (Compositionality). *For Reo Automata $\mathcal{A}_1 = (\Sigma_1, Q_1, \delta_1)$ and $\mathcal{A}_2 = (\Sigma_2, Q_2, \delta_2)$:*

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. $(\partial_{a,b}\mathcal{A}_1) \times \mathcal{A}_2 \sim_R \partial_{a,b}(\mathcal{A}_1 \times \mathcal{A}_2)$, *if* $a, b \notin \Sigma_2$

The notion of equivalence $\sim_R$ used above, presented in [9], is defined as follows.

Definition 5. *Given the Reo 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 $\sim_R$-*bisimulation *iff for all $(q_1, q_2) \in R$:*

If $q_1 \xrightarrow{g|f} q_1' \in \delta_1$ and $\alpha \in \mathbf{At}_\Sigma$, $\alpha \leq g$, then there exists a transition $q_2 \xrightarrow{g'|f} q_2' \in \delta_2$ such that $\alpha \leq g'$ and $(q_1', q_2') \in R$ and vice-versa.

We say that two states $q_1 \in Q_1$ and $q_2 \in Q_2$ are equivalent if there exists a relation containing the pair (q_1, q_2) and we write $q_1 \sim_R q_2$. Two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are $\sim_R$-bisimilar, written $\mathcal{A}_1 \sim_R \mathcal{A}_2$, if there exists a $\sim_R$-bisimulation relation such that every state of $\mathcal{A}_1$ is related to some state of $\mathcal{A}_2$.[1]

A coalgebra primer For the purpose of this paper, viewing Reo automata as coalgebras enables us to apply the generic framework presented in [23] in order to derive a tailor-made language and a sound and complete axiomatization with respect to bisimilarity.

We will introduce next the basic notions of coalgebraic modelling: coalgebra, final coalgebra, bisimilarity and generated subcoalgebra. We also present the notion of trace equivalence for non-deterministic automata which we will use later in order to derive a trace semantics for Reo automata.

A G-coalgebra is a pair (S, f) consisting of a set of *states* S together with a function $f\colon S \to GS$, where G is a functor. The functor G, together with the function f, determines the *transition structure* or dynamics of the G-coalgebra [21]. Classical examples of coalgebras are deterministic automata, infinite streams and non-deterministic automata, which are, respectively, coalgebras for the functors $D(X) = 2 \times X^A$, $St(X) = \mathbb{R} \times X$ and $N = 2 \times \mathcal{P}(X)^A$, where $\mathcal{P}$ denotes the finite powerset functor.

A *G-homomorphism* from a G-coalgebra (S, f) to a G-coalgebra (T, g) is a function $h\colon S \to T$ preserving transitions, *i.e.*, such that $g \circ h = Gh \circ f$.

A G-coalgebra (Ω, ω) is said to be *final* if for any G-coalgebra (S, f) there exists a unique G-homomorphism $\mathbf{beh}_S\colon S \to \Omega$.

Let (X, f) and (Y, g) be two G-coalgebras. We say that the states $x \in X$ and $y \in Y$ are *bisimilar*, written $x \sim_G y$, if and only if they are mapped into the same element in the final coalgebra, that is $\mathbf{beh}_X(x) = \mathbf{beh}_Y(y)$.

[1] In [9], the authors used the terminology bisimulation for what we here call $\sim_R$-bisimulation. We make this distinction, which will be further discussed in Section 5.1, because the notion of bisimulation, which is defined in coalgebraic terms below, does not immediately coincide with the notion of bisimulation considered in [8, 9].

In certain contexts, other equivalences different than bisimilarity, such as trace equivalence, are interesting to consider. Trace equivalence is of particular interest when the system under consideration has some form of non-determinism. Coalgebraically, trace semantics has been studied in [14] and, more recently, in [22]. The latter contains a generalization of the classical powerset construction to a more general class of systems. We recall on the right coalgebraically the powerset construction and below the definition of trace equivalence for non-deterministic automata. In the diagram above: (X, f) is a non-deterministic automaton; $(\mathcal{P}(X), f^\sharp)$ is the determinization of the latter, where $f^\sharp$ is uniquely defined from f as $f^\sharp(S) = \langle b, \lambda a.S_a \rangle$, with $b = 1$ if $f(s) = 1$ for some $s \in S$ ($b = 0$, otherwise) and $S_a = \bigcup_{s \in S} f(s)(a)$; and L is the unique homomorphism into the final coalgebra of the functor $D(X) = 2 \times X^A$ which computes the language recognized by a state of a deterministic automaton.

$$\begin{array}{ccccc} X & \xrightarrow{\{\cdot\}} & \mathcal{P}(X) & \xrightarrow{L} & 2^{A^*} \\ \downarrow{\scriptstyle f} & \swarrow{\scriptstyle f^\sharp} & & & \downarrow \\ 2 \times \mathcal{P}(X)^A & & \xrightarrow[2 \times L^A]{} & & 2 \times (2^{A^*})^A \end{array}$$

We then say that two states x and y in a non-deterministic automaton are trace equivalent, and we write $x \sim_{tr} y$, iff $L(\{x\}) = L(\{y\})$.

Given a G-coalgebra (S, f) and a subset V of S with inclusion map $i \colon V \to S$ we say that V is a subcoalgebra of S if there exists $g \colon V \to GV$ such that i is a homomorphism. Given $s \in S$, $\langle s \rangle \subseteq S$ denotes the subcoalgebra generated by s, *i.e.* the set of states that are reachable from s.

A relation $R \subseteq A \times B$ can be seen as a function $f \colon B \to \mathcal{P}(B)$. Hence, the transition relation of each Reo automaton can be seen as a function $f \colon X \to \mathcal{P}(X \times \mathcal{B}_\Sigma \times 2^\Sigma)$, or equivalently, $f \colon X \to \mathcal{P}(X)^{\mathcal{B}_\Sigma \times 2^\Sigma}$.

A Reo automaton can then be conveniently characterized as a coalgebra for the functor $\mathcal{R}(X) = \mathcal{P}(X)^{\mathcal{B}_\Sigma \times 2^\Sigma}$. It should be noted that we are not modeling the (unifomity) and (reactivity) conditions. In other words, every Reo automaton is an $\mathcal{R}$-coalgebra but not every $\mathcal{R}$-coalgebra is a Reo automaton. Only those coalgebras which satisfy the aforementioned conditions.

Also, as remarked in [9], to obtain a trace (language) semantics for Reo automata we need to consider that all states are accepting. For this purpose, we can also see Reo automata as coalgebras of the functor $\mathcal{R}(X) = 2 \times \mathcal{P}(X)^{\mathcal{B}_\Sigma \times 2^\Sigma}$, where every state is set to be final.

3 A specification language for Reo

In this section, we instantiate the generic framework presented in [23] yielding a language to specify and to reason about Reo automata.

Definition 6 (Expressions for Reo automata). *Given sets of ports Σ and variables X, the set* Exp *of expressions for Reo automata is given by the closed expressions contained in the following BNF, for $g \in \mathcal{B}_\Sigma$, $f \in 2^\Sigma$ and $x \in X$:*

$$\begin{aligned} \varepsilon &::= \underline{\emptyset} \mid \varepsilon \oplus \varepsilon \mid \mu x.\gamma \mid x \mid g{\uparrow}f(\sigma) \\ \gamma &::= \underline{\emptyset} \mid \gamma \oplus \gamma \mid \mu x.\gamma \mid g{\uparrow}f(\sigma) \\ \sigma &::= \underline{\emptyset} \mid \sigma \cup \sigma \mid \{\varepsilon\} \end{aligned}$$

The operator μ in the expression $\mu x.\gamma$ functions as a binder for all the occurrences of the variable x in γ. Note that the only difference between γ and ε is the occurrence of x (γ is an expression where x occurs guarded*, that is only inside an expression of the shape $g{\uparrow}f(-)$). An expression ε is* closed *if all variables $x \in X$ occurring in ε are bounded.*

Intuitively, the expressions $\underline{\emptyset}$, $\oplus$ and $\mu x.\gamma$ are the counterpart of the empty expression, $+$ and star expressions in classical regular expressions, where they denoted the empty language, language union and iteration. In our context, the reader can think of $\underline{\emptyset}$ as the specification of a deadlocked channel, of $\oplus$ as putting the specifications of two channels in parallel and of $\mu x.\gamma$ as the specification of a channel with recursive behavior (or in other words, a persistent channel).

Remark. As mentioned above, the conditions of (*reactivity*) and (*uniformity*) are not modeled in the functor determining the type of Reo automata. Hence, they also do not appear in the definition of the expressions. However, they can easily be taken into account by defining a notion of well-formed expressions where only expressions specifying reactive and uniform Reo automata are allowed. For simplicity, we abstract away from this in this paper and we will work it in full detail in a future extended version.

Example 1. Even before providing semantics to the expressions above, in order to give the reader a feeling for which expressions specify Reo channels, we include in Figure 5 the expressions equivalent to the Reo automata of Figure 3.

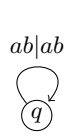
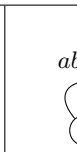
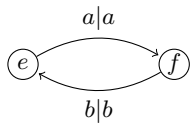
			
Sync	LossySync	SyncDrain	FIFO1
$\mu x.ab{\uparrow}ab(\{x\})$	$\mu x.ab{\uparrow}ab(\{x\}) \oplus a\bar{b}{\uparrow}a(\{x\}))$	$\mu x.ab{\uparrow}ab(\{x\})$	$e = \mu x.a{\uparrow}a(b{\uparrow}b(\{x\}))$ $f = \mu x.b{\uparrow}b(a{\uparrow}a(\{x\}))$

Fig. 5. Expressions corresponding to the automata for basic Reo channels

We now proceed to provide the set of expressions with a coalgebraic structure, which will provide operational semantics to the expressions. More precisely, we will define below a function $\delta\colon \mathsf{Exp} \to \mathcal{P}(\mathsf{Exp})^{\mathcal{B}\Sigma \times 2^{\Sigma}}$. This will allow us to determine when a state s of a system and an expression ε are bisimilar, $s \sim \varepsilon$, or trace equivalent $s \sim_{tr} \varepsilon$.

Definition 7 (Operational semantics). *We define $\delta\colon \mathsf{Exp} \to \mathcal{P}(\mathsf{Exp})^{\mathcal{B}\Sigma \times 2^{\Sigma}}$ by induction on the number of nested ocurrences of μ (and structural induction) as follows:*

$$
\begin{aligned}
\delta(\underline{\emptyset})(\langle g, f\rangle) &= \emptyset \\
\delta(\varepsilon_1 \oplus \varepsilon_2)(\langle g, f\rangle) &= \delta(\varepsilon_1)(g, f) \cup \delta(\varepsilon_2)(g, f) \\
\delta(\mu x.\gamma) &= \delta(\gamma[\mu x.\gamma/x]) \\
\delta(g{\uparrow}f(\sigma))(\langle g', f'\rangle) &= \begin{cases} \overline{\delta}(\sigma) & f = f' \& g = g' \\ \emptyset & otherwise \end{cases}
\end{aligned}
$$

$$
\overline{\delta}(\underline{\emptyset}) = \emptyset \qquad \overline{\delta}(\sigma_1 \cup \sigma_2) = \overline{\delta}(\sigma_1) \cup \overline{\delta}(\sigma_2) \qquad \overline{\delta}(\{\varepsilon\}) = \{\varepsilon\}
$$

Note that $\overline{\delta}$ simply interprets each σ, a syntactical representation of a set of specifications, as the corresponding set.

Having a coalgebra structure on the set of expressions has two advantages: it provides immediately a natural semantics, using the unique homomorphism into the final coalgebra (which can be thought of as the universe of behaviors), and it enables an easy definition on when a state s of a Reo automaton and an expression ε are bisimilar, $s \sim \varepsilon$, or trace equivalent $s \sim_{tr} \varepsilon$.

4 A Kleene theorem for Reo automata

In this section, we present the analogue of Kleene's theorem for Reo automata. More precisely, we show how to convert each expression into a Reo automaton and, conversely, how to compute an expression equivalent to a state of a Reo automaton.

From Reo automata to expressions We start by proving that for each state of a Reo automaton it is possible to compute a bisimilar expression. The expression is built in a similar way as in the classical case of regular expressions and deterministic automata, by solving a system of equations describing the transition structure of each state. In the proof of the theorem we formalize such system of equations and we present an example below in which the similarities with the classical case are more evident.

Theorem 1 (Kleene's theorem for Reo automata (part I)). *For every Reo automaton (S, ξ), if S is* finite *then there exists for any $s \in S$ an expression $\varepsilon_s \in \mathsf{Exp}$ such that $s \sim \varepsilon_s$.*

Proof. Let $S = \{s_1, \ldots, s_n\}$. We construct for a given $s \in S$ an expression ε_s with $s \sim \varepsilon_s$. To this end, we associate with every state $s_i \in S$ a variable $x_i \in X$ and an expression $A_i = \mu x_i.\psi_i$, with ψ_i defined by

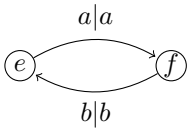
$$\psi_i = \bigoplus_{\substack{\langle g, f\rangle \in \mathcal{B}_\Sigma \times 2^\Sigma \\ \xi(s_i)(\langle g, f\rangle) \neq \emptyset}} \left(g \uparrow f \left(\bigcup_{s_j \in \xi(s_i)(\langle g,f\rangle)} \{x_{s_j}\} \right) \right)$$

Then we define $A_i^0 = A_i$, $A_i^{k+1} = A_i^k\{A_{k+1}^k/x_{k+1}\}$ (for $k = 0, \ldots, n-1$) and we set $\varepsilon_i = A_i^n$. Here, $A\{A'/x\}$ denotes syntactic replacement (that is, substitution without renaming of bound variables in A which are free in A'). This seemingly complicated definition is the analogue of computing the regular expression denoting the language recognized by a state of a deterministic automaton from a system of equations. Below, in an example, the similarities between the system of equations we solve here (using fixed points) in the Reo automaton case and the one for deterministic automata will become more evident. It should be remarked that above we are implicitly considering the argument set of $\bigoplus$ and $\bigcup$ to be ordered. The ordering is not important, since $\oplus$ and $\cup$ can be proved to be an associative, commutative and idempotent operator.

Note that the term $A_i^n = (\mu x_i.\psi_i)\{A_1^0/x_1\}\ldots\{A_n^{n-1}/x_n\}$ is closed, due to the fact that, for every $j = 1, \ldots, n$, the term A_j^{j-1} contains at most $n - j$

free variables in the set $\{x_{j+1}, \ldots, x_n\}$. Moreover, for any $\langle g, f\rangle \in \mathcal{B}_\Sigma \times 2^\Sigma$ $\delta(A_i^n)(\langle g, f\rangle) = \{A_j^n \mid s_j \in \xi(s_i)(\langle g, f\rangle)\}$ (proof in appendix).

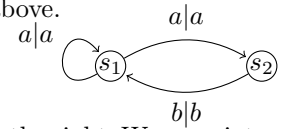
As a consequence of the above we have that the relation $R = \{\langle s, \varepsilon_s\rangle \mid s \in S\}$ is a bisimulation and thus $s \sim \varepsilon_s$, for all $s \in S$.



Let us illustrate the construction above. We recall on the left one of the Reo automata presented in Figure 3. We associate with e and f the variables x_1 and x_2, respectively, and we define the expressions $A_1 = \mu x_1.\psi_1$ and $A_2 = \mu x_2.\psi_2$, where $\psi_1 = a{\uparrow}a(\{x_2\})$ and $\psi_2 = b{\uparrow}b(\{x_1\})$.Then, we compute $A_1^1 = A_1^0 = A_1$, $A_1^2 = A_1^1\{A_2^1/x_2\}$, $A_2^1 = A_2^0\{A_1^0/x_1\} = A_2\{A_1/x_1\}$ and $A_2^2 = A_2^1$. This yields the expressions

$$\varepsilon_1 = A_1^2 = \mu x_1.a{\uparrow}a(\{\mu x_2.b{\uparrow}b(\{\mu x_1.a{\uparrow}a(\{x_2\})\})\})$$
$$\varepsilon_2 = A_2^2 = \mu x_2.b{\uparrow}b(\{\mu x_1.a{\uparrow}a(\{x_2\})\})$$

By construction we have $e \sim \varepsilon_1$ and $f \sim \varepsilon_2$. Note that the expression computed here is slightly different than the one presented in Figure 5. They are however equivalent as can be proved using the axioms we shall introduce later or by just directly constructing a bisimulation. Moreover, we note that computing all the A_j^i is not really needed. In general, one can solve the system of equations by eliminating variables in a more convenient way, but we decided in this example to follow exactly the formalization which we presented above.



All the Reo automata we have seen so far were deterministic. For the reader to get the intuition of what happens in the truly non-deterministic case expressionwise we show the construction above for the automaton on the right. We associate with s_1 and s_2 the variables x_1 and x_2, respectively, and we define $A_1 = \mu x_1.\psi_1$ and $A_2 = \mu x_2.\psi_2$, where ψ_1 and ψ_2 are given by

$$\psi_1 = a{\uparrow}a(\{x_1\} \cup \{x_2\}) \qquad \psi_2 = b{\uparrow}b(\{x_1\})$$

Then, we compute $A_1^1 = A_1^0 = A_1$, $A_1^2 = A_1^1\{A_2^1/x_2\}$, $A_2^1 = A_2^0\{A_1^0/x_1\} = A_2\{A_1/x_1\}$ and $A_2^2 = A_2^1$. This yields the expressions

$$\varepsilon_1 = A_1^2 = \mu x_1.a{\uparrow}a(\{x_1\} \cup \{\varepsilon_2\})\}) \quad \varepsilon_2 = A_2^2 = \mu x_2.b{\uparrow}b(\{\mu x_1.a{\uparrow}a(\{x_1\} \cup \{x_2\})\})$$

As before, we have, by construction, $s_1 \sim \varepsilon_1$ and $s_2 \sim \varepsilon_2$.

From expressions to Reo automata The coalgebra structure (Exp, δ) also provides us with a way of constructing a Reo automaton from an expression $\varepsilon \in \mathsf{Exp}$, by considering the subcoalgebra $\langle\varepsilon\rangle$ (intuitively, $\langle\varepsilon\rangle$ denotes the unraveling of the automaton generated starting in ε by applying δ). The synthesis of a Reo automaton from an expression $\varepsilon \in \mathsf{Exp}$ is what we need to be able to state and prove the second half of Kleene's theorem for Reo automata.[2]

Theorem 2 (Kleene's theorem for Reo automata (part II)). *For every expression $\varepsilon \in \mathsf{Exp}$, there exists a Reo automaton (S, ξ) with S finite and $s \in S$ such that $s \sim \varepsilon$.*

[2] For those readers familiar with the analogue situation for regular expressions we remark the following. For a regular expression r, it is the case that $\langle r\rangle$ will in general

Proof. Let $\varepsilon \in \mathsf{Exp}$. We define $(S, \xi) = \langle \varepsilon \rangle$, where $\langle \varepsilon \rangle$ denotes the smallest subcoalgebra generated by ε. We just need to prove that S is finite, since we already know that there exists a state in S, namely ε, which is trivially bisimilar to ε. We present the proof of finiteness in appendix.

Let us illustrate the construction of the theorem above. Consider the expression $\varepsilon_1 = \mu x.ab{\uparrow}ab(\{x\} \cup \{\mu y.ab{\uparrow}ab(\{y\})\})$. Applying δ we obtain the following:

$$\begin{aligned}\delta(\varepsilon_1)(\langle g, f\rangle) &= \delta(ab{\uparrow}ab(\{\varepsilon_1\} \cup \{\mu y.ab{\uparrow}ab(\{y\})\}))(\langle g, f\rangle)\\ &= \overline{\delta}(\{\varepsilon_1\} \cup \{\mu y.ab{\uparrow}ab(\{y\})\}))\\ &= \{\varepsilon_1, \mu y.ab{\uparrow}ab(\{y\})\}\end{aligned}$$

The first step of the unraveling then yields the automaton on the left below, where $\varepsilon_2 = \mu y.ab{\uparrow}ab(\{y\})$. Applying δ to the new state $\varepsilon_2 = \mu y.ab{\uparrow}ab(\{y\})$ then completes the automaton, which we depict below on the right.

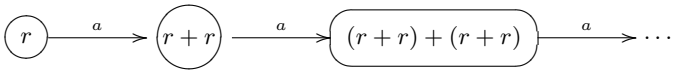
5 Sound and complete axiomatizations

We present next an equational system for expressions in Exp. We define the relation $\equiv \subseteq \mathsf{Exp} \times \mathsf{Exp}$, written infix, as the least reflexive and transitive relation containing the following identities:

1. $(\mathsf{Exp}, \oplus, \underline{\emptyset})$ is a join-semilattice
   $\varepsilon \oplus \varepsilon \equiv \varepsilon$ $(Idemp)$ $\quad$ $\varepsilon_1 \oplus \varepsilon_2 \equiv \varepsilon_2 \oplus \varepsilon_1$ $(Commut)$
   $\varepsilon_1 \oplus (\varepsilon_2 \oplus \varepsilon_3) \equiv (\varepsilon_1 \oplus \varepsilon_2) \oplus \varepsilon_3$ $(Assoc)$ $\quad$ $\underline{\emptyset} \oplus \varepsilon \equiv \varepsilon$ $(Empty)$
2. μ is the unique fixed-point.
   $\gamma[\mu x.\gamma/x] \equiv \mu x.\gamma$ (FP) $\quad$ $\gamma[\varepsilon/x] \equiv \varepsilon \Rightarrow \mu x.\gamma \equiv \varepsilon$ $(Unique)$
3. The join-semilattice structure propagates through the expressions.
   $g{\uparrow}f(\underline{\emptyset}) \equiv \underline{\emptyset}$ $(Zero)$ $\quad$ $g{\uparrow}f(\sigma_1 \cup \sigma_2) \equiv g{\uparrow}f(\sigma_1) \oplus g{\uparrow}f(\sigma_2)$ $(Dist)$
4. $\equiv$ is a congruence.
   $\varepsilon_1 \equiv \varepsilon_2 \Rightarrow \varepsilon[\varepsilon_1/x] \equiv \varepsilon[\varepsilon_2/x]$ if x is free in ε $(Cong)$
5. α-equivalence
   $\mu x.\gamma \equiv \mu y.\gamma[y/x]$ if y is not free in γ $(\alpha - equiv)$

---

be infinite, yielding infinite subcoalgebras like:



In the general framework of [23] this situation also occurred for some of the functors considered and it was solved in a similar fashion to what happens in classical regular expressions. In the functor we are considering in this paper, for Reo automata, this difficulty does not show because of the semilattice structure of $\mathcal{P}$ which is taken into account in the definition of δ for the expression $\varepsilon_1 \oplus \varepsilon_2$.

Theorem 3 (Soundness and Completeness (bisimilarity)). *The axiomatization presented above is sound and complete with respect to bisimilarity, that is:*

$$\varepsilon_1 \sim \varepsilon_2 \Leftrightarrow \varepsilon_1 \equiv \varepsilon_2$$

It is interesting to remark that in the axiomatization above one cannot derive $g{\uparrow}f(\varepsilon_1 \oplus \varepsilon_2) \equiv g{\uparrow}f(\varepsilon_1) \oplus g{\uparrow}f(\varepsilon_2)$. This is similarly to what happens in, for instance, CCS, where the axiom $a.(P+Q) = a.P + a.Q$ is not valid. It is also the key point in order to distinguish bisimilarity from trace equivalence.

An interesting observation, which was not at all considered in the general framework of [23], is that the axiomatization above can be extended with the axiom above and yield a sound and complete axiomatization for trace semantics. This is reminiscent of what Rabinovich [20] showed for a fragment of CCS, where adding to the axiomatization of Milner for bisimilarity the axiom $a.(P+Q) = a.P + a.Q$ resulted in a sound and complete axiomatization for trace semantics. The proof of the theorem below follows a similar structure to that of Rabinovich's and, for space reasons, we omit it here.

Theorem 4 (Soundness and Completeness (trace semantics)). *The axiomatization presented above, augmented with the axiom*

$$g{\uparrow}f(\{\varepsilon_1 \oplus \varepsilon_2\}) \equiv g{\uparrow}f(\{\varepsilon_1\}) \oplus g{\uparrow}f(\{\varepsilon_2\}) \qquad (D1)$$

is sound and complete with respect to trace semantics, that is:

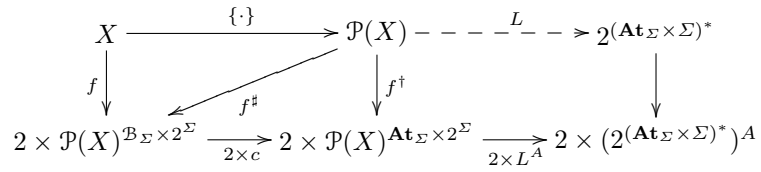
$$\varepsilon_1 \sim_{tr} \varepsilon_2 \Leftrightarrow \varepsilon_1 \equiv \varepsilon_2$$

An interesting feature of the axiomatization(s) above is that $\oplus$ enables the definition of a natural order on expressions: $\varepsilon_1 \leq \varepsilon_2 \Leftrightarrow \varepsilon_1 \oplus \varepsilon_2 \equiv \varepsilon_2$. This opens the door to study refinement of specifications of Reo automata (or, at the automaton level notions of simulation).

5.1 Coalgebraic characterization of $\sim_R$

The definition of bisimulation, which we denote $\sim_R$, considered in [8, 9] (Definition 5) is different than the notion of coalgebraic bisimilarity which one obtains from the functor of Reo automata. The definition of $\sim_R$ involved atoms and, in fact, in [9] they showed a two step construction, where in the first step the automaton is determinized (using the powerset construction, which we recalled in the preliminaries) and in the second step each transition labeled by $g|f$ in the automaton is replaced by n transitions labeled by $\alpha_i|f$, using the fact that each guard g is always equivalent to a disjunction of atoms $\alpha_1 \vee \ldots \vee \alpha_n$. The construction described above had as goal to show that the set $2^{(\mathbf{At}_\Sigma \times \Sigma)^*}$ of guarded strings is the counterpart of formal languages for Reo automata.

It is the aim of this section to show that the definition of $\sim_R$ can be recovered coalgebraically and that the axiomatization above (the one for trace semantics) can be augmented with two axioms yielding a sound and complete axiomatization with respect to the bisimulation of [8, 9]. The key observation is that the bisimulation of [8, 9] can be characterized coalgebraically by the following diagram



where c performs the replacement of $g|f$ by $\alpha_i|f$ as explained above. It is easy to show now that the bisimulation of [8, 9] which we recalled in Definition 5 and denoted by $\sim_R$ can be recovered from the above diagram

$$q_1 \sim_R q_2 \Leftrightarrow L(\{q_1\}) = L(\{q_2\})$$

Moreover, by analyzing the construction above we discovered which axioms we have to add to our previous axiomatization.

Theorem 5 (Soundness and Completeness). *The axiomatization presented in the previous section for trace semantics plus the axioms*

$$(b_1 \vee b_2)\uparrow f(\sigma) \equiv b_1\uparrow f(\sigma) \oplus b_2\uparrow f(\sigma) \quad (\vee) \qquad (\bot\uparrow f)(\sigma) \equiv \underline{\emptyset} \quad (\bot)$$

is sound and complete with respect to $\sim_R$, that is $\varepsilon_1 \sim_R \varepsilon_2 \Leftrightarrow \varepsilon_1 \equiv \varepsilon_2$.

To wrap up this section, we observe that the three equivalences considered in this paper are related by an inclusion: $\sim \subseteq \sim_{tr} \subseteq \sim_R$. This means that the Kleene theorem we presented above for bisimilarity is also valid for the other two equivalences.

6 Discussion

We have presented a framework to reason about Reo automata, a simple and compositional model of the coordination language Reo. The framework consists of (i) a specification language, together with (ii) a Kleene theorem or, more precisely, algorithms to translate expressions to automata and vice-versa and (iii) axiomatizations which enable equational reasoning on expressions. We considered three axiomatizations which are sound and complete with respect to, respectively, bisimilarity, trace equivalence and the bisimulation of [8, 9].

The framework presented in this paper is still in its early stages: there are improvements needed to turn it into a practical language. However, we believe it sets the base of an interesting framework for Reo, which will allow the use of powerful existing tools, in order to perform verification, synthesis and model-checking of Reo circuits. For instance, the general framework of [23] was recently implemented in the automatic theorem prover `Circ` [19, 7]. In [7], the authors proved that it is always possible to automatically decide if two expressions are bisimilar. This enables automatic reasoning on the language presented in this paper. We would like to (i) integrate the framework of [7] in the Eclipe tool-suite of Reo; (ii) extend the `Circ` framework of [7] in order to also automatically prove different equivalences of expressions, such as trace equivalence.

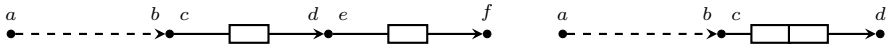
Another research direction is to investigate how to model composition of connectors at the expression level. We have preliminary results on this which suggest that this is not only possible but also not very difficult. Once the composition operator is part of the language it is a natural question whether it is possible to

easily prove (algebraically or coalgebraically) interesting properties such as, for instance, that the Sync channel is an identity element for the composition. Further, casting the framework we presented in this paper in a bialgebraic setting would enable adding new operators, specified by structural operational semantic rules, to the language. Also introducing syntactic sugar would improve the usability of the language (for example, $b \ll a$ could denote b only fires if a also fires and would be translated to a long expression containing all the possible firings containing ab or only a).

Recently, Reo was extended with stochastic information and a quantitative version of Reo automata was proposed as an operational model. Extending the language in order to incorporate stochastic values is an interesting research path, as well as studying if `Circ` can be used to perform quantitative analysis or to model check quantitative Reo.

Encoding translation of other models into the language could also yield useful results. For instance, properties such as the one mentioned above, of Sync being identity in the composition, could then be automatically checked in `Circ` for several semantic models of Reo.

We are also interested in studying and axiomatizing weak (bi)similarity for Reo automata. A prototypical example where this notion of equivalence plays a role is in the two connectors below:



The connector on the right accepts two data items and only then starts loosing items, whereas the connector on the left may loose some data item while the first accepted token moves from one buffer to other. This phenomenon is reminiscent of what happens in process calculi with τ transitions. In the previous papers of Reo automata eliminating τ transitions was not considered in detail. If this elimination is well understood we hope to be able to also axiomatize weak (bi)similarity. We conjecture (and believe) that the connectors above should not be weakly bisimilar but the one on the right weakly simulates the one on the left. This conjecture is based on the fact that weakly bisimilarity is usually not a congruence and having the above connector equivalent indicates otherwise.

Related work There has been work on specification, verification and model checking of Reo circuits [5, 15, 16, 3]. The work presented in [5, 16] have a strong connection with our work in the sense that the languages presented stem from process algebra. However, their focus is different: in both of the papers issues as axiomatizations and Kleene theorem do not arise and they do not consider different equivalence relations, as we do in the present paper. In [16], they take advantage of an existing powerful tool to implement verification of Reo circuits. An eventual integration of `Circ` into the Eclipse tools with certainly benefit from the research carried in the aforementioned paper. A full comparison, discussing the (dis)advantages of each approach, between the approach of [16] and what we present in this paper is left as future work. The main difference with the work presented in [15, 3] is that they consider Büchi like automata (and thus, infinite traces). Also, in [3], only connectors which are not context dependent are taken into account (since they base their results on the constraint automata model).

References

1. F. Arbab. Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.
2. F. Arbab, I. Herman, and P. Spilling. An overview of Manifold and its implementation. *Concurrency - Practice and Experience*, 5(1):23–70, 1993.
3. C. Baier, T. Blechmann, J. Klein, S. Klüppelholz, and W. Leister. Design and verification of systems with exogenous coordination using Vereofy. In *ISoLA (2)*, vol. 6416 of *LNCS*, pages 97–111. Springer, 2010.
4. C. Baier, M. Sirjani, F. Arbab, and J. Rutten. Modeling component connectors in Reo by constraint automata. *SCP*, 61(2):75–113, 2006.
5. M. Barbosa, L. Barbosa, and J. Campos. Towards a coordination model for interactive systems. *ENTCS*, 183:89–103, 2007.
6. S. Bliudze and J. Sifakis. The algebra of connectors - structuring interaction in BIP. *IEEE Trans. Computers*, 57(10):1315–1330, 2008.
7. M. Bonsangue, G. Caltais, E. Goriac, D. Lucanu, J. Rutten, and A. Silva. A decision procedure for bisimilarity of generalized regular expressions. In *Proceedings of SBMF'10*. Springer, 2010. To appear.
8. M. Bonsangue, D. Clarke, and A. Silva. Automata for context-dependent connectors. In *COORDINATION*, vol. 5521 of *LNCS*, pages 184–203. Springer, 2009.
9. M. Bonsangue, D. Clarke, and A. Silva. A model of context-dependent component connectors. *SCP*, 2010. To appear.
10. R. Bruni, I. Lanese, and U. Montanari. A basic algebra of stateless connectors. *TCS*, 366(1-2):98–120, 2006.
11. D. Clarke, D. Costa, and F. Arbab. Connector colouring I: Synchronisation and context dependency. *SCP*, 66(3):205–225, 2007.
12. J. Fiadeiro and A. Lopes. Community on the move: Architectures for distribution and mobility. In *FMCO*, vol. 3188 of *LNCS*, pages 177–196. Springer, 2003.
13. J. Scholten. *Mobile channels for exogenous coordination of distributed systems: semantics, implementation and composition*. PhD thesis, LIACS, Faculty of Mathematics and Natural Sciences, Leiden University, January 2007.
14. I. Hasuo, B. Jacobs, and A. Sokolova. Generic trace semantics via coinduction. *Logical Methods in Computer Science*, 3(4), 2007.
15. M. Izadi, M. Bonsangue, and D. Clarke. Modeling component connectors: Synchronisation and context-dependency. In *SEFM*, pages 303–312. IEEE, 2008.
16. N. Kokash, C. Krause, and E. de Vink. Verification of context-dependent channel-based service models. In *Proceedings of FMCO'09*, 2010.
17. B. Lee and E. Lee. Hierarchical concurrent finite state machines in Ptolemy. In *ACSD*, pages 34–40. IEEE Computer Society, 1998.
18. X. Liu, Y. Xiong, and E. Lee. The Ptolemy II framework for visual languages. In *HCC*, pages 50–51. IEEE Computer Society, 2001.
19. D. Lucanu and G. Rosu. Circ : A circular coinductive prover. In *CALCO*, vol. 4624 of *LNCS*, pages 372–378. Springer, 2007.
20. A. Rabinovich. A complete axiomatisation for trace congruence of finite state behaviors. In *MFPS*, vol. 802 of *LNCS*, pages 530–543. Springer, 1993.
21. J. Rutten. Universal coalgebra: a theory of systems. *TCS*, 249, 2000.
22. A. Silva, F. Bonchi, M. Bonsangue, and J. Rutten. Generalized powerset construction, coalgebraically. In *Proceedings of FSTTCS'10*, 2010. To appear.
23. A. Silva, M. Bonsangue, and J. Rutten. Non-deterministic Kleene coalgebras. *Logical Methods in Computer Science*, 6(3), 2010.
24. C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Professional, 2nd edition, 2002.

A Proofs

Proof of Theorem 1 (cont.) We show here that for any $\langle g, f\rangle \in \mathcal{B}_\Sigma \times 2^\Sigma$ $\delta(A_i^n)(\langle g, f\rangle) = \{A_j^n \mid s_j \in \xi(s_i)(\langle g, f\rangle)\}$.

$$
\begin{aligned}
& \delta(A_i^n)(\langle g, f\rangle) \\
= \;& \delta(\mu x_i.\psi_i)\{A_1^0/x_1\}\ldots\{A_n^{n-1}/x_n\})(\langle g, f\rangle) \\
= \;& \delta(\mu x_i.\psi_i\{A_1^0/x_1\}\ldots\{A_{i-1}^{i-2}/x_{i-1}\}\{A_{i+1}^{i}/x_{i+1}\}\ldots\{A_n^{n-1}/x_n\})(\langle g, f\rangle) \\
\overset{(i)}{=} \;& \delta(\psi_i\{A_1^0/x_1\}\ldots\{A_{i-1}^{i-2}/x_{i-1}\}\{A_{i+1}^{i}/x_{i+1}\}\ldots\{A_n^{n-1}/x_n\}[A_i^n/x_i])(\langle g, f\rangle) \\
\overset{(ii)}{=} \;& \delta(\psi_i\{A_1^0/x_1\}\ldots\{A_{i-1}^{i-2}/x_{i-1}\}\{A_{i+1}^{i}/x_{i+1}\}\ldots\{A_n^{n-1}/x_n\}\{A_i^n/x_i\})(\langle g, f\rangle) \\
\overset{(iii)}{=} \;& \delta(\psi_i\{A_1^0/x_1\}\ldots\{A_{i-1}^{i-2}/x_{i-1}\}\{A_i^n/x_i\}\{A_{i+1}^{i}/x_{i+1}\}\ldots\{A_n^{n-1}/x_n\})(\langle g, f\rangle) \\
\overset{(iv)}{=} \;& \overline{\delta}\left(\left(\bigoplus_{s_j\in\xi(s_i)(\langle g,f\rangle)}\{x_{s_j}\}\right)\{A_1^0/x_1\}\ldots\{A_{i-1}^{i-2}/x_{i-1}\}\{A_i^n/x_i\}\{A_{i+1}^{i}/x_{i+1}\}\ldots\{A_n^{n-1}/x_n\}\right) \\
= \;& \{x_j\{A_j^{j-1}/x_j\}\ldots\{A_n^{n-1}/x_n\} \mid s_j \in \xi(s_i)(\langle g, f\rangle)\} \\
= \;& \{A_j^n \mid s_j \in \xi(s_i)(\langle g, f\rangle)\}
\end{aligned}
$$

Here, note that (i) follows by definition of δ; (ii) because $[A_i^n/x_i] = \{A_i^n/x_i\}$, since A_i^n has no free variables; (iii) follows because x_i is not free in $A_{i+1}^i, \ldots, A_n^{n-1}$; (iv) is a consequence of the definition of $\overline{\delta}$ and:

$$
\begin{aligned}
& \{A_i^n/x_i\}\{A_{i+1}^i/x_{i+1}\}\ldots\{A_n^{n-1}/x_n\} \\
= \;& \{A_i^{i-1}\{A_{i+1}^i/x_{i+1}\}\ldots\{A_n^{n-1}/x_n\}/x_i\}\{A_{i+1}^i/x_{i+1}\}\ldots\{A_n^{n-1}/x_n\} \\
= \;& \{A_i^{i-1}/x_i\}\{A_{i+1}^i/x_{i+1}\}\ldots\{A_n^{n-1}/x_n\} \qquad (1)
\end{aligned}
$$

Equation (1) uses the syntactic identity

$$A\{B\{C/y\}/x\}\{C/y\} = A\{B/x\}\{C/y\} \qquad (2)$$

Proof of Theorem 2 (cont.) Let $\langle\varepsilon\rangle = (S, \xi)$, for some $\varepsilon \in \mathsf{Exp}$. We prove that S is contained in a finite set, namely the set $cl(\varepsilon) \cup \{\underline{\emptyset}\}$, where the closure $cl(\varepsilon)$ of ε, which is the set containing all subexpressions and unfoldings of ε, is defined as the smallest set satisfying

$$
\begin{aligned}
cl(\underline{\emptyset}) &= \{\underline{\emptyset}\} \\
cl(\varepsilon_1 \oplus \varepsilon_2) &= \{\varepsilon_1 \oplus \varepsilon_2\} \cup cl(\varepsilon_1) \cup cl(\varepsilon_2) \\
cl(g\!\uparrow\! f(\sigma)) &= \{g\!\uparrow\! f(\sigma)\} \cup \overline{cl}(\sigma) \\
cl(\mu x.\varepsilon) &= \{\mu x.\varepsilon\} \cup cl(\varepsilon[\mu x.\varepsilon/x]) \\
cl(x) &= \{x\}
\end{aligned}
$$

$$
\begin{aligned}
\overline{cl}(\underline{\emptyset}) &= \{\underline{\emptyset}\} \\
\overline{cl}(\sigma_1 \cup \varepsilon_2) &= \{\sigma_1 \cup \sigma_2\} \cup \overline{cl}(\sigma_1) \cup \overline{cl}(\sigma_2) \\
\overline{cl}(\{\sigma\}) &= \{\{\sigma\}\} \cup cl(\sigma)
\end{aligned}
$$

The set $cl(\varepsilon)$ is finite, because the number of different unfoldings of μ-expressions is finite. We only need to prove that for any $\langle g, f\rangle \in \mathcal{B}_\Sigma \times 2^\Sigma$

$$\delta^\dagger(\varepsilon)(\langle g, f\rangle) \in cl(\varepsilon)$$

and, for any $\varepsilon' \in cl(\varepsilon)$, $\delta^\dagger(\varepsilon')(\langle g, f\rangle) \in cl(\varepsilon)$. Because $\varepsilon \in cl(\varepsilon)$, the first equation is actually a special case of the second which we prove below by induction on $N(\varepsilon')$.

$$N(\underline{\emptyset}) = N(g\uparrow f(\sigma)) = 0 \quad N(\varepsilon_1 \oplus \varepsilon_2) = max\{N(\varepsilon_1), N(\varepsilon_2)\} + 1 \quad N(\mu x.\gamma) = N(\gamma) + 1$$

$$\begin{array}{lll}
\delta(\underline{\emptyset})(\langle g, f\rangle) & = \underline{\emptyset} \in cl(\varepsilon) & \\
\delta(\varepsilon_1 \oplus \varepsilon_2)(\langle g, f\rangle) & = \delta(\varepsilon_1) \cup \delta(\varepsilon_2)(\langle g, f\rangle) \in cl(\varepsilon_1) \cup cl(\varepsilon_2) & \text{(induction hypothesis)} \\
\delta(\mu x.\varepsilon_1)(\langle g, f\rangle) & = \delta(\varepsilon_1[\mu x.\varepsilon_1/x])(\langle g, f\rangle) \in cl(\varepsilon) & \text{(induction hypothesis)} \\
\delta(g'\uparrow f'(\varepsilon_1))(\langle g, f\rangle) & = \underline{\emptyset} & \text{(if } g \neq g' \text{ or } f \neq f'\text{)} \\
\delta(g'\uparrow f'(\varepsilon_1))(\langle g, f\rangle) & = \overline{\delta}(\varepsilon_1) \in cl(\varepsilon) & \text{(if } g = g' \text{ and } f = f'\text{)}
\end{array}$$

For the last step note that $\overline{\delta}(\varepsilon_1) \subseteq cl(\varepsilon_1) \subseteq cl(\varepsilon)$.