Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

SEN

Software Engineering

Software ENgineering

A coinductive calculus of component connectors

F. Arbab, J.J.M.M. Rutten

REPORT SEN-R0216 SEPTEMBER 30, 2002

CWI is the National Research Institute for Mathematics and Computer Science. It is sponsored by the Netherlands Organization 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)

P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 1386-369X

A Coinductive Calculus of Component Connectors

F. Arbab and J.J.M.M. Rutten*
CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
Email: janr@cwi.nl, URL: www.cwi.nl/∼janr

ABSTRACT
Reo is a recently introduced channel-based coordination model, wherein complex coordinators, called connectors, are compositionally built out of simpler ones. Using a more liberal notion of a channel, Reo generalises existing dataflow networks. In this paper, we present a simple and transparent semantical model for Reo, in which connectors are relations on timed data streams. Timed data streams constitute a characteristic of our model and consist of twin pairs of separate data and time streams. Furthermore, coinduction is our main reasoning principle and we use it to prove properties such as connector equivalence.

2000 ACM Computing Classification System: C.2.4, D.1.3, D.3.2, D.3.3, F.1, F.3
Keywords & Phrases: Coinduction, coalgebra, component, stream, coordination.

1 Introduction

Reo (from the Greek word $\rho\epsilon\omega$ which means "*[I] flow*") is a recently introduced [Arb02, AM02] channel-based coordination model, wherein complex coordinators, called connectors, are compositionally built out of simpler ones. Reo is intended as a "glue language" for construction of connectors that orchestrate component instances in a component-based system. The emphasis in Reo is on connectors and their composition only, not on the components that are being connected. In this paper, we present a simple and transparent semantical model of connectors and connector composition, which can be used as a compositional *calculus*, in which properties such as connector equivalence, optimization, and realization can be expressed and proved.

The basic connectors are channels, each of which is a point-to-point communication medium with two distinct ends. Channels can be used as the only communication constructs in communication models for concurrent systems, because the primitives of other communication models (e.g., message passing or remote procedure calls) can be easily defined using channels. In contrast to other channel-based models, Reo uses a generalised concept of channel. In addition to the common channel types of synchronous and asynchronous, with bounded or unbounded buffers, and with fifo and other ordering schemes, Reo allows an open ended set of channels, each with its own, sometimes exotic, behaviour. For instance, a channel in Reo need not have both an input and an output end; it can have two input ends or two output ends instead. In addition to channels, Reo has one more basic connector, called the merge operator. More complex connectors can then be constructed from the basic connectors (channels and merge) through an operation of connector composition.

Because Reo is not concerned with the internal activity of the components that it connects, we represent components by their interfaces only. Therefore, we model the input ends and output ends of connectors as *streams* (infinite sequences) of abstract (uninterpreted) data items. Moreover, we associate with every such data stream an infinite sequence of (natural or non-negative real)

*Rutten is also a member of the Faculty of Mathematics and Computer Science, Vrije Universiteit, De Boelelaan 1081a, 1081 HV Amsterdam.

numbers. These numbers stand for the respective moments in time at which their corresponding data items are being input or output. This allows us to describe and reason about the precise timing constraints of connectors (such as synchronous versus asynchronous, and bounded versus unbounded delay). Thus, we model the potential behaviour of connector ends as *timed data streams*, which are pairs consisting of a data stream and a time stream. Note that we use pairs of streams rather than streams of pairs (of timed data elements), since this enables us to reason about time explicitly, which turns out to be particularly useful.

The main mathematical ingredients of our model are sets A^ω of streams over some set A (of data items or time moments). These sets A^ω carry a so-called *final coalgebra* structure, consisting of the well-known operations of head and tail (here called initial value and derivative). As a consequence, we can benefit from some basic but very general facts from the discipline of coalgebra, which over the last decade has been developed as a general behavioral theory for dynamical systems (see [JR97, Rut00] for an overview). In particular, the final coalgebra A^ω satisfies principles of *coinduction*, both for definitions and for proofs. The latter are formulated in terms of so-called *stream bisimulations*, an elementary variation on Park's and Milner's original notion of bisimulation for parallel processes [Mil80, Par81]. As we shall see, these coinduction principles are surprisingly powerful. They will be applied to both data streams and time streams.

Having modelled connector ends as timed data streams, we then model connectors as *relations* on timed data streams, expressing which combinations of timed data streams are mutually consistent. This relational model is, in spite of its simplicity, already sufficiently expressive to study a number of notions and questions about component connectors, such as equivalence (when do two connectors have the same behavior?), expressiveness (which connectors can I build out of a given set of basic connectors?), optimization (given a connector, can I build an equivalent connector out of a smaller number of basic connectors?), verification (given the specification of a certain connector behavior and given a connector, does the connector meet the specification?), realization (given the specification of a certain connector behavior, can I actually build a connector with precisely that behavior out of a given set of basic connectors?), and the like. In the present paper, we shall mainly focus on connector equivalence and, to a lesser extent, the expressiveness of our connector calculus.

Reo is more general than dataflow models, Kahn-networks, and Petri-nets, which can be viewed as specialised channel-based models that incorporate certain basic constructs for primitive coordination. While Reo is designed to deal with the flow of data, it, more specifically, differs fundamentally from classical dataflow models in four important aspects:

1. Although not treated here, the topology of connections in 'full' Reo is inherently dynamic and accomodates mobility.
2. Reo supports a much more general notion of channels.
3. The model of Reo is based on a clear separation of data and time.
4. Coinduction is the main reasoning principle.

Of all related work, Broy and Stølen's book [BS01] deserves special mention, since it is also based on (timed) data streams. However, the points mentioned above distinguish our model also from theirs. In particular, the separation of data and time, in our model, in combination with the use of coinduction, leads to simpler specifications (definitions) and proofs. See Section 9 for a concrete example. Finally, coalgebra and coinduction have been used in models of component-based systems in [Bar01] and [Dob02]. Also these models are distinguished from ours by the (first three) points above. Moreover, our model is far more concrete, and therefore allows actual equivalence proofs.

2 Streams and coinduction

Let A be any set and let A^ω be the set of all streams (infinite sequences) over A:

$$A^\omega = \{\alpha \mid \alpha : \{0, 1, 2, \ldots\} \to A\}$$

We present some basic facts on A^ω, notably how to give definitions and proofs by coinduction. In this section, the set A is arbitrary but later, we shall look in particular at streams over some data set D and streams over the time domain $\mathbb{R}_+$.

Individual streams will be denoted as $\alpha = (\alpha(0), \alpha(1), \alpha(2), \ldots)$ (or $a = (a(0), a(1), a(2), \ldots)$). We call $\alpha(0)$ the *initial value* of α. The *(stream) derivative* α' of a stream α is defined as

$$\alpha' = (\alpha(1), \alpha(2), \alpha(3), \ldots)$$

Note that $\alpha'(n) = \alpha(n+1)$, for all $n \geq 0$. Later we shall also need 'higher-order' derivatives $\alpha^{(k)}$, for any $k \geq 0$, defined as $\alpha^{(0)} = \alpha$ and $\alpha^{(k+1)} = (\alpha^{(k)})'$. These satisfy $\alpha^{(k)}(n) = \alpha(n+k)$, for any $n \geq 0$.

Stream initial values and derivatives will be used both in definitions of (operations on) streams and in proofs of properties of streams. In this manner, a *calculus* of streams is obtained, in close analogy to classical analytical analysis. More specifically, we formulate definitions using the so-called *behavioural differential equations*, which specify both the initial value and the derivative of the stream being defined. Such definitions are also called *coinductive*. We illustrate this type of definition through a few basic examples. Let the operations *even*, *odd*, and *zip* be defined by the following system of equations (one for each $\alpha, \beta \in A^\omega$):

behavioural differential equation	initial value
$even(\alpha)' = even(\alpha'')$	$even(\alpha)(0) = \alpha(0)$
$odd(\alpha)' = odd(\alpha'')$	$odd(\alpha)(0) = \alpha'(0)$
$zip(\alpha, \beta)' = zip(\beta, \alpha')$	$zip(\alpha, \beta)(0) = \alpha(0)$

The reader should have no trouble convincing himself that these operations satisfy the following identities:

$$\begin{aligned} even(\alpha) &= (\alpha(0), \alpha(2), \alpha(4), \ldots) \\ odd(\alpha) &= (\alpha(1), \alpha(3), \alpha(5), \ldots) \\ zip(\alpha, \beta) &= (\alpha(0), \beta(0), \alpha(1), \beta(1), \ldots) \end{aligned}$$

These equalities could in fact have been taken as definitions, but we prefer the coinductive definitions instead, because they allow the use the coinduction proof principle, as we shall see shortly.

As in analysis, whether a differential equation has a (unique) solution or not depends in general on the shape of the equation. For the three elementary behavioural differential equations above (and in fact all other equations that we shall encounter in the present paper), the existence of a unique solution can be easily established by some elementary reasoning. (For the general case, see the remark at the end of this section.)

Proofs about streams will be given in terms of the following elementary notion. A *(stream) bisimulation* is a relation $R \subseteq A^\omega \times A^\omega$ such that, for all α and β in A^ω:

$$\text{if } \alpha \, R \, \beta \text{ then } \begin{cases} (1) & \alpha(0) = \beta(0) \quad \text{and} \\ (2) & \alpha' \, R \, \beta' \end{cases}$$

(The union of all bisimulation relations is itself a bisimulation, called *bisimilarity*.) Bisimulations are used in the formulation of the following *coinduction proof principle*. For all $\alpha, \beta \in A^\omega$:

$$\text{if } \alpha \, R \, \beta, \text{ for some bisimulation } R, \quad \text{then} \quad \alpha = \beta \tag{1}$$

In other words, in order to prove $\alpha = \beta$, it is sufficient to establish the existence of a bisimulation relation $R \subseteq A^\omega \times A^\omega$ such that $\alpha \, R \, \beta$.

Consider for instance the following three identities on streams, for all $\alpha, \beta \in A^\omega$,

1. $even(zip(\alpha, \beta)) = \alpha$
2. $odd(zip(\alpha, \beta)) = \beta$
3. $zip(even(\alpha), odd(\alpha)) = \alpha$

Since the following three relations on streams:

1. $\{\langle even(zip(\alpha,\beta)),\, \alpha\rangle \mid \alpha,\beta \in A^\omega\}$
2. $\{\langle odd(zip(\alpha,\beta)),\, \beta\rangle \mid \alpha,\beta \in A^\omega\}$
3. $\{\langle zip(even(\alpha),\, odd(\alpha)),\, \alpha\rangle \mid \alpha \in A^\omega\} \;\cup\; \{\langle zip(odd(\alpha),\, even(\alpha'')),\, \alpha'\rangle \mid \alpha \in A^\omega\}$

are bisimulations, the above three identities follow, respectively, by coinduction.

The validity of the proof principle itself can be easily established by proving $\alpha(n) = \beta(n)$ for all $n \geq 0$, by induction on n. More abstractly, both the coinduction proof and definition principle are ultimately based on the fact that the set A^ω carries a *final coalgebra* structure, which is given by the combination of the operations of initial value and stream derivative:

$$A^\omega \to A \times A^\omega, \quad \alpha \mapsto \langle \alpha(0), \alpha' \rangle$$

See [JR97, Rut00] for general references on coalgebra. For a detailed treatment of the final coalgebra of streams, see [Rut01]. The latter paper contains in particular detailed results about behavioural differential equations (for streams over the set $A = \mathbb{R}$ of real numbers).

3 Coinduction and greatest fixed points

There is yet another, in fact more classical, way of understanding bisimulations and the coinduction proof principle. Consider the set $\mathcal{P}(A^\omega \times A^\omega) = \{R \mid R \subseteq A^\omega \times A^\omega\}$ of binary relations on A^ω, and the function $\Phi : \mathcal{P}(A^\omega \times A^\omega) \to \mathcal{P}(A^\omega \times A^\omega)$ defined, for any $R \subseteq A^\omega \times A^\omega$, by

$$\Phi(R) = \{\langle \alpha, \beta\rangle \mid \alpha(0) = \beta(0) \;\; \wedge \;\; \langle \alpha', \beta'\rangle \in R\,\}$$

As an immediate consequence of the definition of bisimulation, we have

$$R \text{ is a bisimulation} \quad \Leftrightarrow \quad R \subseteq \Phi(R)$$

Bisimulation relations are, in other words, *post-fixed points* of Φ. (The characterisation of bisimulations as post-fixed points goes back, in the context of nondeterministic transition systems, to [Par81, Mil80].) Consequently, the coinduction proof principle (1) is equivalent to the following equality, where $id_{A^\omega} = \{\langle \alpha, \alpha\rangle \mid \alpha \in A^\omega\}$:

$$id_{A^\omega} \quad = \quad \bigcup\{R \mid R \subseteq \Phi(R)\,\}$$

Since id_{A^ω} is itself a (bisimulation and thus a) post-fixed point, it is in fact the greatest fixed point of Φ. Therefore the above equality is an instance of the following well-known greatest fixed point theorem [Tar55]. Let X be any set and let $\mathcal{P}(X) = \{V \mid V \subseteq X\}$ be the set of all its subsets. If $\Psi : \mathcal{P}(X) \to \mathcal{P}(X)$ is a monotone operator, that is, $R \subseteq S$ implies $\Psi(R) \subseteq \Psi(S)$ for all $R \subseteq X$ and $S \subseteq X$, then Ψ has a greatest fixed point $P = \Psi(P)$ satisfying

$$P \quad = \quad \bigcup\{R \mid R \subseteq \Psi(R)\,\} \tag{2}$$

This equality can be used as a proof principle in the same way as (1): in order to prove that $R \subseteq P$, for any $R \subseteq X$, it suffices to show that R is a post-fixed point of Ψ, that is, $R \subseteq \Psi(R)$. We shall see further instances of this theorem (and as many related proof principles) in Section 6.

4 Timed data streams

We model connectors as relations on timed data streams, which we introduce in this section.

For the remainder of this paper, let D be an (arbitrary) set, the elements of which will be called *data* elements. The set DS of *data streams* is defined as

$$DS = D^\omega$$

that is, the set of all streams $\alpha = (\alpha(0), \alpha(1), \alpha(2), \ldots)$ over D. Let $\mathbb{R}_+$ be the set of non-negative real numbers, which play in the present context the role of *time* moments. Let $\mathbb{R}_+^\omega$ be the set of all streams $a = (a(0), a(1), a(2), \ldots)$ over $\mathbb{R}_+$. Let $<$ and $\leq$ be the relations on $\mathbb{R}_+^\omega$ that are obtained as the pointwise extensions of the corresponding ('strictly smaller' and 'smaller than') relations on $\mathbb{R}_+$. That is, for all $a = (a(0), a(1), a(2), \ldots)$ and $b = (b(0), b(1), b(2), \ldots)$ in $\mathbb{R}_+^\omega$,

$$a < b \;\equiv\; \forall n \geq 0,\ a(n) < b(n), \quad a \leq b \;\equiv\; \forall n \geq 0,\ a(n) \leq b(n)$$

The set TS of *time streams* is defined by the following subset of $\mathbb{R}_+^\omega$:

$$TS = \{a \in \mathbb{R}_+^\omega \mid a < a'\}$$

Note that time streams $a \in TS$ satisfy, for all $n \geq 0$,

$$a(n) < a'(n) = a(n+1)$$

and thus consist of increasing time moments $a(0) < a(1) < a(2) < \cdots$.

Finally, the set TDS of *timed data streams* is defined by

$$TDS = DS \times TS$$

and contains pairs $\langle \alpha, a \rangle$ consisting of a data stream $\alpha = (\alpha(0), \alpha(1), \alpha(2), \ldots)$ in DS, and a time stream $a = (a(0), a(1), a(2), \ldots)$ in TS.

As we shall see shortly, connectors will be modelled as relations on timed data streams. Each of the arguments of such a relation will be viewed as an input or as an output end of the connector that is modelled by the relation. Thus the following operational interpretation of a timed data stream $\langle \alpha, a \rangle$ can be given: the time stream a specifies for each $n \geq 0$ the time moment $a(n)$ at which the nth data element $\alpha(n)$ is being input or output:

α:	$\alpha(0)$	$\alpha(1)$	$\alpha(2)$	$\cdots$	$\alpha(n)$	$\cdots$
a:	$a(0)$	$a(1)$	$a(2)$	$\cdots$	$a(n)$	$\cdots$

Connectors being relations, there are typically many timings a possible at a specific connector's end, which together with a given data stream α form admissible timed data streams $\langle \alpha, a \rangle$, that is, satisfying the connector's relation. A timed data stream $\langle \alpha, a \rangle$ could therefore also be viewed as a *scenario*, one out of many, for the behaviour of a connector end. Connectors, then, relate various such scenarios that together are mutually consistent.

As we already observed in the introduction, one could have, alternatively and equivalently, defined timed data streams as (a subset of) $(D \times \mathbb{R}_+)^\omega$, because of the existence of an isomorphism

$$D^\omega \times \mathbb{R}_+^\omega \cong (D \times \mathbb{R}_+)^\omega, \quad \langle \alpha, a \rangle \mapsto (\langle \alpha(0), a(0) \rangle, \langle \alpha(1), a(1) \rangle, \langle \alpha(2), a(2) \rangle, \ldots)$$

We prefer to work with pairs of streams rather than streams of pairs, because this will allow us to reason about the data streams and time streams separately, which turns out to be of crucial importance for much of what follows.

We could also have used streams of *natural* numbers $0, 1, 2, \ldots$ for our timings, rather than (positive) real numbers. This difference would leave most of our model unaffected. Our model with 'continuous time', however, is more abstract than the model with 'discrete time' would be, in the sense that more connector equivalences can be proved. (In the world of temporal logic, this observation goes back to at least [BRP86].) An example is the equivalence of a fifo_2 buffer (with capacity 2) with the composition of two fifo_1 buffers in Section 7.

Finally, it is often useful to require time streams a to be not only increasing: $a < a'$, but also *progressive*: for every $N \geq 0$ there exists $n \geq 0$ with $a(n) > N$. This assumption prevents 'Zeno' paradoxes, where infinitely many actions take place in a bounded time interval. In most of what follows, the progressive time assumption is not used, but whenever it is, we shall mention it explicitly.

5 Basic connectors: channels

The most basic connectors are *channels*, which are formally defined as binary relations

$$R \subseteq TDS \times TDS$$

on timed data streams. For such relations, we distinguish between input and output argument positions, called *input ends* and *output ends*, respectively. This information will be relevant for the definition of connector composition in Section 7. In the pictures that we draw of channels and connectors, input and output ends are denoted by the following arrow shaft and head:

input: $\vdash\!\!-\!\!-\cdots$ output: $\cdots\!\!-\!\!-\!\!\rightarrow$

Here are the (for our purposes) most important examples of channels:

1. The *synchronous channel* $\vdash\!\!-\!\!-\!\!\rightarrow$ is defined, for all timed data streams $\langle \alpha, a \rangle$ and $\langle \beta, b \rangle$, by

   $$\langle \alpha, a \rangle \vdash\!\!-\!\!-\!\!\rightarrow \langle \beta, b \rangle \quad \equiv \quad \alpha = \beta \ \wedge \ a = b$$

   This channel inputs the data (elements in the) stream α at times a, and outputs the data stream β at times b. All data elements that come in, come out again (in the same order): $\alpha = \beta$. Moreover, each element enters and exits the channel at the very same time moment: $a = b$.

2. The *synchronous drain* $\overset{syndr}{\vdash\!\!-\!\!\dashv}$ is defined, for all timed data streams $\langle \alpha, a \rangle$ and $\langle \beta, b \rangle$, by

   $$\langle \alpha, a \rangle \overset{syndr}{\vdash\!\!-\!\!\dashv} \langle \beta, b \rangle \quad \equiv \quad a = b$$

   The corresponding data elements in the streams α and β enter the two input ends of this channel simultaneously: $a = b$. No relation on the data streams is specified (the data elements enter and 'disappear').

3. The *fifo buffer* $\overset{fifo}{\vdash\!\!-\!\!\rightarrow}$ is defined, for all timed data streams $\langle \alpha, a \rangle$ and $\langle \beta, b \rangle$, by

   $$\langle \alpha, a \rangle \overset{fifo}{\vdash\!\!-\!\!\rightarrow} \langle \beta, b \rangle \quad \equiv \quad \alpha = \beta \ \wedge \ a < b$$

   This is an unbounded fifo (first-in-first-out) buffer. What comes in, comes out (in the same order): $\alpha = \beta$, but later: $a < b$ (which is equivalent to $a(n) < b(n)$, for all $n \geq 0$).

4. The *fifo*$_1$ *buffer* $\overset{fifo_1}{\vdash\!\!-\!\!\rightarrow}$ is defined, for all timed data streams $\langle \alpha, a \rangle$ and $\langle \beta, b \rangle$, by

   $$\langle \alpha, a \rangle \overset{fifo_1}{\vdash\!\!-\!\!\rightarrow} \langle \beta, b \rangle \quad \equiv \quad \alpha = \beta \ \wedge \ a < b < a'$$

   This models a 1-bounded fifo buffer. What comes in, comes out: $\alpha = \beta$, but later: $a < b$. Moreover, at any moment the next data item can be input only after the present data item has been output: $b < a'$, which is equivalent to $b(n) < a(n+1)$, for all $n \geq 0$.

5. The *fifo*$_k$ *buffer* $\overset{fifo_k}{\vdash\!\!-\!\!\rightarrow}$ for any $k \geq 1$, is defined, for all timed data streams $\langle \alpha, a \rangle$ and $\langle \beta, b \rangle$, by

   $$\langle \alpha, a \rangle \overset{fifo_k}{\vdash\!\!-\!\!\rightarrow} \langle \beta, b \rangle \quad \equiv \quad \alpha = \beta \ \wedge \ a < b < a^{(k)}$$

   (Recall from Section 2 that $a^{(k)}$ denotes the k-th derivative of the stream a.) This models a k-bounded fifo buffer, generalizing the fifo$_1$ buffer above. What comes in, comes out: $\alpha = \beta$, but later: $a < b$. Moreover, at any moment the kth-next data item can be input only after the present data item has been output: $b < a^{(k)}$ (which is equivalent to $b(n) < a(n+k)$, for all $n \geq 0$).

6. Let $x \in D$ be any fixed data element. The *fifo*(x) *buffer* $\overset{fifo(x)}{\longmapsto}$ is defined, for all timed data streams $\langle \alpha, a \rangle$ and $\langle \beta, b \rangle$, by

$$\langle \alpha, a \rangle \overset{fifo(x)}{\longmapsto} \langle \beta, b \rangle \quad \equiv \quad \beta(0) = x \ \wedge \ \alpha = \beta' \ \wedge \ a < b'$$

This channel behaves precisely as the unbounded fifo buffer above, but for the fact that, initially, it contains the data element x, which is the first element to come out: $\beta(0) = x$ (at time $b(0)$).

6 More channels and the merge connector

The definitions of the basic (channel) connectors so far have been, mathematically speaking, fairly straightforward. Next, we introduce some further basic connectors, including the merge operator, using greatest fixed point definitions. As we saw in Section 3, each of these definitions will come together with its own proof principle, similar to the coinduction principle in Section 2.

1. The *asynchronous drain* $\overset{asyndr}{\longmapsto}$ inputs any two streams of data items at its two input ends, but never at the same time (in contrast to the synchronous drain of Section 5). It is defined, for all timed data streams $\langle \alpha, a \rangle$ and $\langle \beta, b \rangle$, by

$$\langle \alpha, a \rangle \overset{asyndr}{\longmapsto} \langle \beta, b \rangle \quad \equiv \quad a \bowtie b$$

where $\bowtie \subseteq TS \times TS$ is a relation on time streams, given by

$$a \bowtie b \quad \equiv \quad a(0) \neq b(0) \ \wedge \ \begin{cases} a' \bowtie b & \text{if } a(0) < b(0) \\ a \bowtie b' & \text{if } b(0) < a(0) \end{cases}$$

More precisely, $\bowtie$ is defined as the greatest fixed point of the following monotone operator, $\Phi_{\bowtie} : \mathcal{P}(TS \times TS) \to \mathcal{P}(TS \times TS)$, defined for $R \subseteq TS \times TS$, by

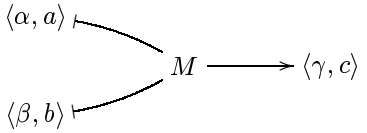
$$\Phi_{\bowtie}(R) \quad = \quad \{\langle a, b \rangle \mid a(0) \neq b(0) \ \wedge \ \begin{cases} \langle a', b \rangle \in R & \text{if } a(0) < b(0) \\ \langle a, b' \rangle \in R & \text{if } b(0) < a(0) \end{cases} \}$$

Thus $\bowtie = gfp(\Phi_{\bowtie})$. A $\bowtie$-*bisimulation* is a relation $R \subseteq TS \times TS$ with $R \subseteq \Phi_{\bowtie}(R)$. There is, as an immediate consequence of (2) in Section 3, the following $\bowtie$-*coinduction* proof principle. For all time streams a and b:

$$\text{if } a \, R \, b, \text{ for some } \bowtie\text{-bisimulation } R, \quad \text{then} \quad \alpha \bowtie \beta \tag{3}$$

For an example of a proof by $\bowtie$-coinduction, see the end of the present section.

2. The connector *merge* is a ternary relation M with two input ends and one output end, and is defined, for timed data streams $\langle \alpha, a \rangle, \langle \beta, b \rangle, \langle \gamma, c \rangle$, by



$$\begin{aligned} & \equiv \quad M(\langle \alpha, a \rangle, \langle \beta, b \rangle, \langle \gamma, c \rangle) \\ & \equiv \quad a(0) \neq b(0) \ \wedge \\ & \qquad \begin{cases} \alpha(0) = \gamma(0) \ \wedge \ a(0) = c(0) \ \wedge \ M(\langle \alpha', a' \rangle, \langle \beta, b \rangle, \langle \gamma', c' \rangle) & \text{if } a(0) < b(0) \\ \beta(0) = \gamma(0) \ \wedge \ b(0) = c(0) \ \wedge \ M(\langle \alpha, a \rangle, \langle \beta', b' \rangle, \langle \gamma', c' \rangle) & \text{if } b(0) < a(0) \end{cases} \end{aligned}$$

This connector merges the two data streams α and β on its input ends into a stream γ on its output end, on a 'first come first served' basis. It inputs one data element at a time: $a(0) \neq b(0)$. The data element that is handled first, say $\alpha(0)$ at time $a(0)<b(0)$, is the first element to come out: $\alpha(0) = \gamma(0)$, at exactly the same moment: $a(0) = c(0)$. After that, the connector handles the remainder of the streams in the same manner again: $M(\langle \alpha', a' \rangle, \langle \beta, b \rangle, \langle \gamma', c' \rangle)$. (Similarly for the case that $\beta(0)$ is handled first, at time $b(0) < a(0)$.) The relation M can be formally defined as the greatest fixed point of a monotone operator Φ_M defined, for any $R \subseteq TDS \times TDS \times TDS$, by

$$\begin{array}{l} \Phi_M(R)(\langle \alpha, a \rangle, \langle \beta, b \rangle, \langle \gamma, c \rangle) \\ \quad \Leftrightarrow \quad a(0) \neq b(0) \quad \wedge \\ \qquad \left\{ \begin{array}{lllll} \alpha(0) = \gamma(0) & \wedge & a(0) = c(0) & \wedge & R(\langle \alpha', a' \rangle, \langle \beta, b \rangle, \langle \gamma', c' \rangle) \quad \text{if } a(0) < b(0) \\ \beta(0) = \gamma(0) & \wedge & b(0) = c(0) & \wedge & R(\langle \alpha, a \rangle, \langle \beta', b' \rangle, \langle \gamma', c' \rangle) \quad \text{if } b(0) < a(0) \end{array} \right. \end{array}$$

An *M-bisimulation* is a relation R with $R \subseteq \Phi_M(R)$ and we have a *M-coinduction* proof principle: if $R(\langle \alpha, a \rangle, \langle \beta, b \rangle, \langle \gamma, c \rangle)$, for some M-bisimulation R, then $M(\langle \alpha, a \rangle, \langle \beta, b \rangle, \langle \gamma, c \rangle)$.

Under the assumption that our time streams are progressive (defined at the end of Section 4), the merge operator is *fair*: from both input ends, infinitely many data elements will be input.

3. The *lossy synchronous channel* $\stackrel{lsyn}{\longmapsto}$ is defined, for all $\langle \alpha, a \rangle$ and $\langle \beta, b \rangle$, by

$$\langle \alpha, a \rangle \stackrel{lsyn}{\longmapsto} \langle \beta, b \rangle \quad \equiv$$

$$a(0) \leq b(0) \quad \wedge \quad \left\{ \begin{array}{lll} \alpha(0) = \beta(0) \quad \wedge \quad \langle \alpha', a' \rangle \stackrel{lsyn}{\longmapsto} \langle \beta', b' \rangle & & \text{if } a(0) = b(0) \\ \langle \alpha', a' \rangle \stackrel{lsyn}{\longmapsto} \langle \beta, b \rangle & & \text{if } a(0) < b(0) \end{array} \right.$$

This channel passes an input data element instantaneously on as an output element: $\alpha(0) = \beta(0)$, in case $a(0) = b(0)$; after that, it continues with the remainder of the streams as before. If $a(0) < b(0)$, that is, if it is too early for the output end of the channel to be active, the input data element $\alpha(0)$ is simply discarded (lost), and the channel proceeds with $\langle \alpha', a' \rangle$ on its input and $\langle \beta, b \rangle$ on its output end. As before, this channel can be formally defined as the greatest fixed point of a monotone operator on the set of binary relations on timed data streams.

Next, we illustrate the use of the coinduction proof principles that were introduced above. A look at the definition of M and one moment's thought suffice to see that, for all timed data streams $\langle \alpha, a \rangle$, $\langle \beta, b \rangle$, $\langle \gamma, c \rangle$,

$$\text{if } M(\langle \alpha, a \rangle, \langle \beta, b \rangle, \langle \gamma, c \rangle) \quad \text{then} \quad a \bowtie b \tag{4}$$

since the merge connector never inputs two data items at its two input ends at the same time. But how to prove it formally? The answer is provided by what we have called $\bowtie$-coinduction above. Consider the following relation:

$$R = \{ \langle k, l \rangle \mid \exists \kappa, \lambda, \mu, m : \ M(\langle \kappa, k \rangle, \langle \lambda, l \rangle, \langle \mu, m \rangle) \}$$

Using the definition of M, it is straightforward to prove that this is a $\bowtie$-bisimulation. As a consequence of (3), $R \subseteq \bowtie$, which implies (4). For a second example, consider the following equivalence, for all timed data streams $\langle \alpha, a \rangle, \langle \beta, b \rangle, \langle \gamma, c \rangle$:

$$M(\langle \alpha, a \rangle, \langle \beta, b \rangle, \langle \gamma, c \rangle) \quad \Leftrightarrow \quad M(\langle \beta, b \rangle, \langle \alpha, a \rangle, \langle \gamma, c \rangle) \tag{5}$$

The implication from left to right (and thus the equivalence) follows by M-coinduction from the trivial observation that

$$S = \{(\langle\alpha, a\rangle, \langle\beta, b\rangle, \langle\gamma, c\rangle) \mid M(\langle\beta, b\rangle, \langle\alpha, a\rangle, \langle\gamma, c\rangle)\}$$

is an M-bisimulation, which implies $S \subseteq M$.

7 Composing connectors

Connectors are relations and their composition can therefore be naturally modelled by relational composition. For instance, the composition of two copies of the synchronous channel yields the following binary relation, defined for all timed data streams $\langle\alpha, a\rangle$ and $\langle\beta, b\rangle$, by

$$\begin{aligned}
&\langle\alpha, a\rangle \longmapsto \circ \longmapsto \langle\beta, b\rangle \\
&\quad\equiv\quad \exists\langle\gamma, c\rangle :\ \ \langle\alpha, a\rangle \longmapsto \langle\gamma, c\rangle \quad\wedge\quad \langle\gamma, c\rangle \longmapsto \langle\beta, b\rangle \\
&\quad\equiv\quad \exists\langle\gamma, c\rangle :\ (\alpha = \gamma \ \wedge\ a = c) \ \wedge\ (\gamma = \beta \ \wedge\ c = b)
\end{aligned}$$

(which happens to be equivalent to $\langle\alpha, a\rangle \longmapsto \langle\beta, b\rangle \quad\equiv\quad \alpha = \beta \ \wedge\ a = b$). Composition essentially does two things at the same time: the output end (argument) of the first connector is identified with the input end of the second, and the resulting 'mixed' end is moreover hidden (encapsulated) by the existential quantification. We shall also use the following picture to denote connector composition:

$$\langle\alpha, a\rangle \longmapsto \exists\langle\gamma, c\rangle \longmapsto \langle\beta, b\rangle$$

Here $\exists\langle\gamma, c\rangle$ is used to indicate that this is an internal end of the connector, which is no longer accessible (for further compositions) from outside. The two aspects of connector composition: identification and hiding, could, and for the full version of Reo actually should, be separated. But for the basic examples we shall be dealing with, this type of composition is sufficient.

Note that the identification of connector ends, which are timed data streams, includes the identification of the respective time streams, thus synchronising the timings of the two connectors.

The general definition of the composition of an arbitrary n-ary connector R and an m-ary connector T, is essentially the same. One has to select a number of (distinct) output ends and input ends from R, and equal numbers of input ends and output ends from T, which then are connected in pairs in precisely the same manner as in the example above. To describe this in full generality, one would have to be slightly more formal and explicit about the (input and output) types of argument positions. Although not very difficult, such a formalisation would not be very interesting. Moreover, it will not be necessary for the instances of connector composition that will be presented here. In all cases, the relevant typing information will be contained in the pictorial representations with which connector compositions will be introduced.

We shall also allow an output end to be connected to *several* input ends at the same time, to each of which the output is copied. Here is an example, in which the output end of a synchronous channel is connected to the input ends of two other synchronous channels:

$$\begin{aligned}
&\langle\alpha, a\rangle \longmapsto \exists\langle\delta, d\rangle \longmapsto \langle\beta, b\rangle \\
&\qquad\qquad\quad\ \big\downarrow \\
&\qquad\qquad\quad\ \langle\gamma, c\rangle \\
&\quad\equiv\quad \exists\langle\delta, d\rangle :\ (\alpha = \delta \ \wedge\ a = d) \ \wedge\ (\delta = \beta \ \wedge\ d = b) \ \wedge\ (\delta = \gamma \ \wedge\ d = c) \\
&\quad\equiv\quad \alpha = \beta = \gamma \ \wedge\ a = b = c
\end{aligned}$$

The connection of several *output ends* to one and the same input end, can be modelled by means of the merge operator introduced in Section 6.

Finally, it is relevant to note that nothing in the above prevents us from connecting an output end to an input end of one and the same connector, simply by connecting them to the input end and the output end of a synchronous channel. In other words, we have in passing included the possibility of *feedback* loops into our calculus.

Here are various examples of composite connectors (including examples of feedback) built out of a number of basic connectors. As usual, $\langle \alpha, a \rangle$, $\langle \beta, b \rangle$, $\langle \gamma, c \rangle$, ... are arbitrary timed data streams.

1. The composition of two unbounded fifo buffers yields again an unbounded fifo buffer:

   $$\overset{fifo}{\longmapsto} \circ \overset{fifo}{\longmapsto} = \overset{fifo}{\longmapsto}$$

   because of the following equivalences:

   $$\begin{aligned}
   & \langle \alpha, a \rangle \overset{fifo}{\longmapsto} \circ \overset{fifo}{\longmapsto} \langle \beta, b \rangle \\
   \equiv \quad & \langle \alpha, a \rangle \overset{fifo}{\longmapsto} \exists \langle \gamma, c \rangle \overset{fifo}{\longmapsto} \langle \beta, b \rangle \\
   \equiv \quad & \exists \langle \gamma, c \rangle : \ \alpha = \gamma \ \wedge \ a < c \ \wedge \ \gamma = \beta \ \wedge \ c < b \\
   \equiv \quad & \alpha = \beta \ \wedge \ a < b \\
   \equiv \quad & \langle \alpha, a \rangle \overset{fifo}{\longmapsto} \langle \beta, b \rangle
   \end{aligned}$$

2. The composition of two fifo_1 buffers yields a fifo_2 buffer:

   $$\overset{fifo_1}{\longmapsto} \circ \overset{fifo_1}{\longmapsto} = \overset{fifo_2}{\longmapsto}$$

   because

   $$\begin{aligned}
   & \langle \alpha, a \rangle \overset{fifo_1}{\longmapsto} \circ \overset{fifo_1}{\longmapsto} \langle \beta, b \rangle \\
   \equiv \quad & \langle \alpha, a \rangle \overset{fifo_1}{\longmapsto} \exists \langle \gamma, c \rangle \overset{fifo_1}{\longmapsto} \langle \beta, b \rangle \\
   \equiv \quad & \exists \langle \gamma, c \rangle : \ \alpha = \gamma \ \wedge \ a < c < a' \ \wedge \ \gamma = \beta \ \wedge \ c < b < c' \\
   \Rightarrow \quad & \alpha = \beta \ \wedge \ a < b < a'' = a^{(2)} \quad [\text{since } c < a' \text{ implies } c' < a''] \\
   \equiv \quad & \langle \alpha, a \rangle \overset{fifo_2}{\longmapsto} \langle \beta, b \rangle
   \end{aligned}$$

   Given $\alpha = \beta$ and $a < b < a''$, the converse of the above implication can be proved by defining $\gamma = \alpha$ and

   $$c(n) = 1/2 \times (\max\{a(n), b(n-1)\} + \min\{a(n+1), b(n)\})$$

   for all $n \geq 0$ (where $b(n-1) = 0$ for $n = 0$).

3. Consider the following composition of three synchronous channels:

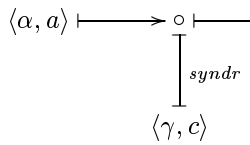
   $$\langle \alpha, a \rangle \longmapsto \circ \longmapsto \langle \beta, b \rangle \quad \equiv \quad \alpha = \beta = \gamma \ \wedge \ a = b = c$$

   $$\Big\downarrow$$

   $$\langle \gamma, c \rangle$$

   This connector can be viewed as a 'take-cue' regulator: any time a data item is taken from γ (by some future context), that same data item is allowed to flow from left to right. This constitutes one of the most basic examples of what could be called *exogenous coordination*, that is, coordination from outside.

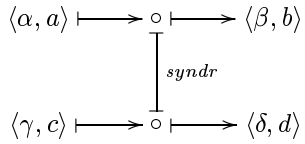
4. The following connector is a variation on the previous one, in that the lower channel now is a synchronous drain:



$\equiv \quad \alpha = \beta \ \wedge \ a = b = c$

It is a 'write-cue' regulator that regulates the flow of data items from left to right by inputs or writes on the lower channel end. Note that what is being input there is irrelevant. What matters is that such inputs are synchronised, through the synchronous drain, with both the channels above: $a = b = c$.

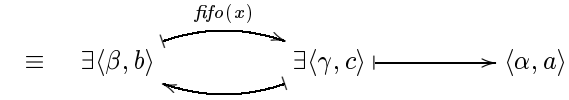
5. With four synchronous channels and one synchronous drain, the following barrier synchroniser can be constructed:



$\equiv \quad \alpha = \beta \ \wedge \ \gamma = \delta \ \wedge \ a = b = c = d$

The synchronous drain in the middle ensures that data items pass through the upper and lower channels simultaneously.

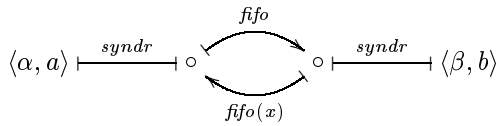
6. Here is a simple example of a feedback loop, consisting of one (unbounded) fifo buffer containing an initial data element $x \in D$, and two synchronous channels:



$$
\begin{aligned}
\equiv \quad & \exists \langle \beta, b \rangle \, \exists \langle \gamma, c \rangle : \ (\gamma(0) = x \ \wedge \ \beta = \gamma' \ \wedge \ b < c') \ \wedge \ (\gamma = \beta \ \wedge \ c = b) \\
& \wedge \ (\gamma = \alpha \ \wedge \ c = a) \\
\equiv \quad & \exists \langle \beta, b \rangle \, \exists \langle \gamma, c \rangle : \ \alpha = \beta = \gamma \ \wedge \ a = b = c \ \wedge \ \gamma(0) = x \ \wedge \ \gamma = \gamma' \ \wedge \ c < c' \\
\equiv \quad & \exists \langle \beta, b \rangle \, \exists \langle \gamma, c \rangle : \ \alpha = \beta = \gamma \ \wedge \ a = b = c \ \wedge \ \gamma = (x, x, x, \ldots) \\
\equiv \quad & \alpha = (x, x, x, \ldots)
\end{aligned}
$$

For the last but one equivalence, note that $(\gamma(0) = x \wedge \gamma = \gamma')$ is equivalent to $\gamma = (x, x, x, \ldots)$; moreover, the inequality $c < c'$ is redundant, because c is by assumption a time sequence and hence satisfies this inequality by definition. The behaviour of this connector is thus pretty much what it should be. It outputs perpetuously the data element x. Note that there are no constraints on the time stream a $(= b = c)$. The only requirement is that it is indeed a time stream, that is, satisfies $a < a'$.

7. Let $x \in D$ be again some fixed data element. The following connector acts as a *sequencer* on its two connector ends:



$$\equiv \quad \langle \alpha, a \rangle \overset{syndr}{\longmapsto\!\!\!\!\dashv} \exists \langle \gamma, c \rangle \underset{fifo(x)}{\overset{fifo}{\rightleftarrows}} \exists \langle \delta, d \rangle \overset{syndr}{\longmapsto\!\!\!\!\dashv} \langle \beta, b \rangle$$

$$\begin{aligned}
&\equiv \quad \exists \langle \gamma, c \rangle \, \exists \langle \delta, d \rangle : \; a = c \;\; \wedge \;\; d = b \;\; \wedge \\
&\qquad (\gamma = \delta \;\; \wedge \;\; c < d) \;\; \wedge \;\; (\gamma(0) = x \;\; \wedge \;\; \gamma' = \delta \;\; \wedge \;\; d < c') \\
&\equiv \quad \exists \langle \gamma, c \rangle \, \exists \langle \delta, d \rangle : \; \gamma = \delta = (x, x, x, \ldots) \;\; \wedge \;\; c = a \;\; \wedge \;\; d = b \;\; \wedge \;\; (c < d < c') \\
&\equiv \quad a < b < a'
\end{aligned}$$

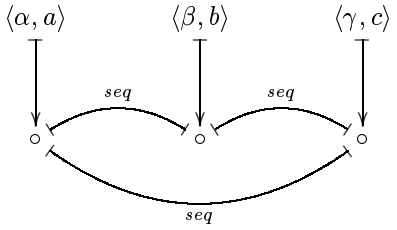
Thus, arbitrary data elements can be input alternatingly from the left and the right channel ends, at times

$$a(0) < b(0) < a(1) < b(1) < \cdots$$

For future reference, we introduce the following notation for the sequencer connector:

$$\langle \alpha, a \rangle \overset{seq}{\longmapsto\!\!\!\!\dashv} \langle \beta, b \rangle \quad \equiv \quad a < b < a' \tag{6}$$

8. One can construct a connector that serialises any number k of channel ends by combining $k + 1$ sequencers. For instance,



$$\begin{aligned}
&\equiv \quad (a < b < a') \;\; \wedge \;\; (a < c < a') \;\; \wedge \;\; (b < c < b') \\
&\equiv \quad a < b < c < a'
\end{aligned}$$

8 Connector equivalence

In Section 7, we already saw some elementary examples of connector equivalence, such as:

$$\longmapsto \circ \longmapsto \;=\; \longmapsto$$

$$\overset{fifo}{\longmapsto} \circ \overset{fifo}{\longmapsto} \;=\; \overset{fifo}{\longmapsto}$$

$$\overset{fifo_1}{\longmapsto} \circ \overset{fifo_1}{\longmapsto} \;=\; \overset{fifo_2}{\longmapsto}$$

Below we present some further, slightly less elementary examples.

1. Recall the definition of the sequencer in Section 7:

$$\langle \alpha, a \rangle \overset{seq}{\longmapsto\!\!\!\!\dashv} \langle \beta, b \rangle \quad \equiv \quad \langle \alpha, a \rangle \overset{syndr}{\longmapsto\!\!\!\!\dashv} \circ \underset{fifo(x)}{\overset{fifo}{\rightleftarrows}} \circ \overset{syndr}{\longmapsto\!\!\!\!\dashv} \langle \beta, b \rangle$$

$$\equiv \quad a < b < a'$$

(where $x \in D$ is some fixed data item). Here is an alternative way of constructing the sequencer, now with a 1-bounded fifo buffer and a synchronous drain:

$$\langle \alpha, a \rangle \overset{fifo_1}{\longmapsto} \circ \overset{syndr}{\longmapsto} \langle \beta, b \rangle$$

$$\begin{aligned}
&\equiv \quad \langle \alpha, a \rangle \overset{fifo_1}{\longmapsto} \exists \langle \gamma, c \rangle \overset{syndr}{\longmapsto} \langle \beta, b \rangle \\
&\equiv \quad \exists \langle \gamma, c \rangle : \ (\alpha = \gamma \ \ \wedge \ \ a < c < a') \ \ \wedge \ \ c = b \\
&\equiv \quad a < b < a' \\
&\equiv \quad \langle \alpha, a \rangle \overset{seq}{\longmapsto} \langle \beta, b \rangle
\end{aligned}$$

2. Conversely, a 1-bounded fifo buffer can be constructed using two synchronous channels, a sequencer, and an unbounded fifo buffer:

$$\langle \alpha, a \rangle \overset{fifo_1}{\longmapsto} \langle \beta, b \rangle \quad \equiv \quad \langle \alpha, a \rangle \longmapsto \circ \underset{seq}{\overset{fifo}{\rightleftarrows}} \circ \longmapsto \langle \beta, b \rangle$$

because we have the following equivalences:

$$\langle \alpha, a \rangle \longmapsto \circ \underset{seq}{\overset{fifo}{\rightleftarrows}} \circ \longmapsto \langle \beta, b \rangle$$

$$\equiv \quad \langle \alpha, a \rangle \longmapsto \exists \langle \gamma, c \rangle \underset{seq}{\overset{fifo}{\rightleftarrows}} \exists \langle \delta, d \rangle \longmapsto \langle \beta, b \rangle$$

$$\begin{aligned}
&\equiv \quad \exists \langle \gamma, c \rangle, \exists \langle \delta, d \rangle : \ \langle \alpha, a \rangle = \langle \gamma, c \rangle \ \ \wedge \ \ \langle \delta, d \rangle = \langle \beta, b \rangle \ \ \wedge \\
&\qquad (\gamma = \delta \ \ \wedge \ \ c < d) \ \ \wedge \ \ c < d < c') \\
&\equiv \quad \alpha = \beta \ \ \wedge \ \ a < b < a' \\
&\equiv \quad \langle \alpha, a \rangle \overset{fifo_1}{\longmapsto} \langle \beta, b \rangle
\end{aligned}$$

3. Recall the definition of asynchronous drain in Section 6:

$$\langle \alpha, a \rangle \overset{asyndr}{\longmapsto} \langle \beta, b \rangle \quad \equiv \quad a \bowtie b$$

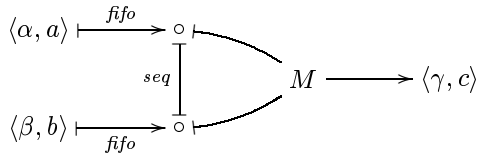
Here is another way of constructing the asynchronous drain, using the merge connector and a synchronous drain:

$$\begin{matrix} \langle \alpha, a \rangle \longmapsto \\ \qquad\qquad M \longrightarrow \circ \overset{syndr}{\circlearrowright} \\ \langle \beta, b \rangle \longmapsto \end{matrix}$$

$$\begin{aligned}
&\equiv \quad \exists \langle \gamma, c \rangle : \ M(\langle \alpha, a \rangle, \langle \beta, b \rangle, \langle \gamma, c \rangle) \ \ \wedge \ \ c = c \\
&\equiv \quad a \bowtie b \\
&\equiv \quad \langle \alpha, a \rangle \overset{asyndr}{\longmapsto} \langle \beta, b \rangle
\end{aligned}$$

For the middle equivalence, the implication from left to right follows from (3) in Section 6, which was proved by $\bowtie$-coinduction. The converse implication can be proved in a similar fashion, using M-coinduction.

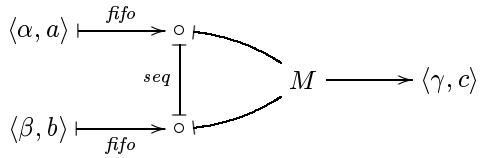
4. Next we look at a connector that is built from two unbounded fifo buffers, the sequencer, and the merge operator:



Using the coinduction proof principle, we shall prove that this connector has the following behaviour (with the operations *zip*, *even* and *odd* as in Section 2):

$$zip(\alpha,\beta) = \gamma \quad \wedge \quad a < even(c) \quad \wedge \quad b < odd(c)$$

The proof consists of the following sequence of equivalences:



$$\equiv \quad \langle\alpha,a\rangle \xrightarrow{fifo} \exists\langle\delta,d\rangle \;\; \langle\beta,b\rangle \xrightarrow{fifo} \exists\langle\epsilon,e\rangle \;\; (seq,\ M) \longrightarrow \langle\gamma,c\rangle$$

$$\begin{aligned}
&\equiv \quad \exists\langle\delta,d\rangle\,\exists\langle\epsilon,e\rangle :\ (\alpha=\delta \quad \wedge \quad a<d) \quad \wedge \quad (\beta=\epsilon \quad \wedge \quad b<e) \quad \wedge \quad (d<e<d') \\
&\qquad \wedge \quad M(\langle\delta,d\rangle,\ \langle\epsilon,e\rangle,\ \langle\gamma,c\rangle) \\
&\equiv \quad \exists d,\ e :\ a<d \quad \wedge \quad b<e \quad \wedge \quad (d<e<d') \quad \wedge \quad M(\langle\alpha,d\rangle,\ \langle\beta,e\rangle,\ \langle\gamma,c\rangle) \\
&\equiv \quad \exists d,\ e :\ a<d \quad \wedge \quad b<e \quad \wedge \quad zip(\alpha,\beta)=\gamma \quad \wedge \quad zip(d,e)=c \quad \text{[using (7) below]} \\
&\equiv \quad \exists d,\ e :\ a<d \quad \wedge \quad b<e \quad \wedge \quad zip(\alpha,\beta)=\gamma \quad \wedge \quad d=even(c) \quad \wedge \quad e=odd(c) \\
&\equiv \quad zip(\alpha,\beta)=\gamma \quad \wedge \quad a<even(c) \quad \wedge \quad b<odd(c)
\end{aligned}$$

We have used the following equivalence, which will be proved by coinduction:

$$(d<e<d') \quad \wedge \quad M(\langle\alpha,d\rangle,\ \langle\beta,e\rangle,\ \langle\gamma,c\rangle) \quad \Leftrightarrow \quad (zip(\alpha,\beta)=\gamma \quad \wedge \quad zip(d,e)=c) \qquad (7)$$

For the implication from left to right, define the following relation on streams:

$$R = \{\langle zip(\kappa,\lambda),\ \mu\rangle \mid \exists k,l,m :\ (k<l<k') \ \wedge \ M(\langle\kappa,k\rangle,\ \langle\lambda,l\rangle,\ \langle\mu,m\rangle)\}$$

We show that R is a bisimulation. Consider a pair $\langle zip(\kappa,\lambda),\ \mu\rangle$ in R, with corresponding time streams k,l,m. Because $k<l$ it follows from $M(\langle\kappa,k\rangle,\ \langle\lambda,l\rangle,\ \langle\mu,m\rangle)$ that $\mu(0) = \kappa(0)$, and since $\kappa(0) = zip(\kappa,\lambda)(0)$, this proves the first of the two bisimulation conditions. Next consider the pair of derivatives $\langle zip(\kappa,\lambda)',\mu'\rangle = \langle zip(\lambda,\kappa'),\mu'\rangle$. It follows from the definition of M that the latter pair is again in R, since

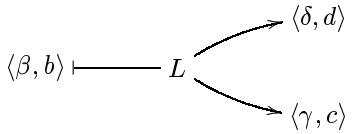
$$\begin{aligned}
&(k<l<k') \quad \wedge \quad M(\langle\kappa,k\rangle,\ \langle\lambda,l\rangle,\ \langle\mu,m\rangle) \\
&\quad \Rightarrow \quad (l<k'<l') \quad \wedge \quad M(\langle\kappa',k'\rangle,\ \langle\lambda,l\rangle,\ \langle\mu',m'\rangle) \\
&\quad \equiv \quad (l<k'<l') \quad \wedge \quad M(\langle\lambda,l\rangle,\ \langle\kappa',k'\rangle,\ \langle\mu',m'\rangle) \quad \text{[by equivalence (5)]}
\end{aligned}$$

This proves that R is a bisimulation. Assuming now $(d<e<d')$ and $M(\langle\alpha,d\rangle,\ \langle\beta,e\rangle,\ \langle\gamma,c\rangle)$, $zip(\alpha,\beta) = \gamma$ follows by coinduction. In the same manner, one shows $zip(d,e) = c$. This proves the implication from left to right of equivalence (7). The implication from right to left can be proved along similar lines, using M-coinduction.

9 Protocol verification

Our calculus of component connectors also allows the formulation and formal verification of communication protocols. We present a simple example, taken from [BS01, pp. 29–36]. It consists of an (unbounded fifo) lossy buffer composed with a driver that corrects the lossiness of the buffer. Below we specify both connectors, and prove that their composition is equivalent to an ordinary (correct) unbounded fifo buffer.

Let $\langle \alpha, a \rangle$, $\langle \beta, b \rangle$, and $\langle \delta, d \rangle$ be timed data streams over an arbitrary data set D and let $\langle \gamma, c \rangle$ be a timed data stream with $\gamma \in \{0,1\}^\omega$. We define the lossy buffer as a ternary relation L on timed data streams with one input end and two output ends as follows:



$$\begin{aligned} &\equiv \quad L(\langle \beta, b \rangle, \langle \gamma, c \rangle, \langle \delta, d \rangle) \\ &\equiv \quad b < c \ \wedge \ b < d \ \wedge \ \left\{ \begin{array}{l} \gamma(0) = 1 \ \wedge \ \delta(0) = \beta(0) \ \wedge \ L(\langle \beta', b' \rangle, \langle \gamma', c' \rangle, \langle \delta', d' \rangle) \\ \vee \\ \gamma(0) = 0 \ \wedge \ L(\langle \beta', b' \rangle, \langle \gamma', c' \rangle, \langle \delta, d \rangle) \end{array} \right. \end{aligned}$$

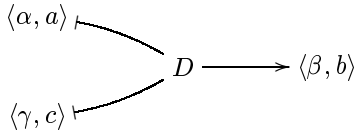
This connector inputs data items at the input end β. For every data item that is input, there are two possible scenarios: (1) the data item is stored successfully and is output (at some later moment) at the upper output end δ together (not necessarily simultaneously) with a success signal 1 along γ, after which the connector proceeds as before with the remainder of all streams involved. *Or*: (2) storage of the data item fails, no data item is output along the end β, a 0 signalling the failure is output along γ, and the connector proceeds as before, now with $\langle \beta', b' \rangle$ and $\langle \gamma', c' \rangle$, but with $\langle \delta, d \rangle$ unchanged.

It follows from the definition of the lossy buffer that eventually some data item gets successfully stored and output. In other words, there exists $n \geq 0$ with $\gamma(n) = 1$. In order to prove this, assume $L(\langle \beta, b \rangle, \langle \gamma, c \rangle, \langle \delta, d \rangle)$ and suppose that $\gamma(n) = 0$, for all $n \geq 0$. Then

$$L(\langle \beta^{(n)}, b^{(n)} \rangle, \langle \gamma^{(n)}, c^{(n)} \rangle, \langle \delta, d \rangle)$$

for all $n \geq 0$ (recall that the superscript (n) stands for the n-th derivative). As a consequence, $b^{(n)}(0) = b(n) < d(0)$, for all n. Under the assumption that our time streams are progressive (cf. the end of Section 4): for any $N \geq 0$ there exists $n \geq 0$ with $b(n) > N$, this is a contradiction. Therefore, there exists $n \geq 0$ with $\gamma(n) = 1$. We see that, somewhat surprisingly, the fact that all our streams are infinite and the assumption that time streams are progressive, together imply here the right type of fairness (or liveness) behaviour.

Next we turn to the driver, which has two input ends and one output end, and is defined as the following ternary relation D:

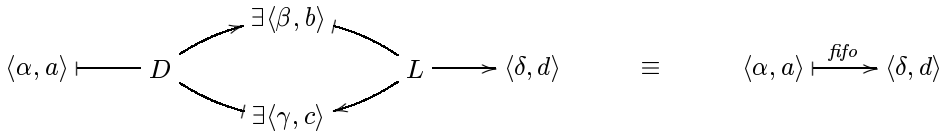


$$\begin{aligned} &\equiv \quad D(\langle \alpha, a \rangle, \langle \beta, b \rangle, \langle \gamma, c \rangle) \\ &\equiv \quad a = b \ \wedge \ a < c < a' \ \wedge \ \beta(0) = \alpha(0) \ \wedge \ \left\{ \begin{array}{l} \gamma(0) = 1 \ \wedge \ D(\langle \alpha', a' \rangle, \langle \beta', b' \rangle, \langle \gamma', c' \rangle) \\ \vee \\ \gamma(0) = 0 \ \wedge \ D(\langle \alpha, a' \rangle, \langle \beta', b' \rangle, \langle \gamma', c' \rangle) \end{array} \right. \end{aligned}$$

The driver inputs data items at α and outputs them at β. Before proceeding with the next data item, it checks its input at γ. If $\gamma(0) = 1$ then the last data item that has been output is considered to have been handled correctly (by the lossy buffer in the composition below), and D proceeds as before with the remainder of all streams involved. If $\gamma(0) = 0$, however, something has gone

wrong (the buffer has lost the data item), and D sends the data item again. This is modelled here by $D(\langle \alpha, a' \rangle, \langle \beta', b' \rangle, \langle \gamma', c' \rangle)$, in which all streams have progressed to their derivatives but for α, which remains unchanged. As a consequence, $\alpha(0)$ is (again) the next data item that D will output (but note that the time stream a *has* changed into a').

Composing the driver and the lossy buffer as below yields a connector that is equivalent with the (non-lossy) unbounded fifo buffer: for all timed data streams $\langle \alpha, a \rangle$ and $\langle \delta, d \rangle$,



For the implication from left to right, we have to show that $\alpha = \delta$ (and $a < d$). To this end, define the following relation on data streams:

$$R = \{\langle \alpha, \delta \rangle \mid \exists a, d, \langle \beta, b \rangle, \langle \gamma, c \rangle : \ D(\langle \alpha, a \rangle, \langle \beta, b \rangle, \langle \gamma, c \rangle) \ \wedge \ L(\langle \beta, b \rangle, \langle \gamma, c \rangle, \langle \delta, d \rangle)\}$$

In order to prove that R is a bisimulation relation, consider a pair $\langle \alpha, \delta \rangle$ in R with 'witnesses' $a, d, \langle \beta, b \rangle, \langle \gamma, c \rangle$ such that $D(\langle \alpha, a \rangle, \langle \beta, b \rangle, \langle \gamma, c \rangle)$ and $L(\langle \beta, b \rangle, \langle \gamma, c \rangle, \langle \delta, d \rangle)$. Let n be the smallest natural number such that $\gamma(n) = 1$ (which exists by the remark above). It follows that

$$D(\langle \alpha, a^{(n)} \rangle, \langle \beta^{(n)}, b^{(n)} \rangle, \langle \gamma^{(n)}, c^{(n)} \rangle) \ \wedge \ L(\langle \beta^{(n)}, b^{(n)} \rangle, \langle \gamma^{(n)}, c^{(n)} \rangle, \langle \delta, d \rangle)$$

Together with $\gamma^{(n)}(0) = \gamma(n) = 1$, this implies $\alpha(0) = \beta^{(n)}(0) = \delta(0)$ and, moreover,

$$D(\langle \alpha', a^{(n+1)} \rangle, \langle \beta^{(n+1)}, b^{(n+1)} \rangle, \langle \gamma^{(n+1)}, c^{(n+1)} \rangle) \ \wedge$$

$$L(\langle \beta^{(n+1)}, b^{(n+1)} \rangle, \langle \gamma^{(n+1)}, c^{(n+1)} \rangle, \langle \delta', d' \rangle)$$

Thus, $\langle \alpha', \delta' \rangle \in R$, which concludes the proof that R is a bisimulation. It now follows by coinduction that $\alpha = \delta$. (A minor variation on this argument proves that $a < d$.)

For the implication from right to left, choose $\langle \beta, b \rangle = \langle \alpha, a \rangle$, $\gamma = (1, 1, 1, \ldots)$, and $c = 1/2 \times (a + a')$ (in a hopefully self explanatory notation). It is now a straightforward proof by (D- and L-)coinduction to show that $D(\langle \alpha, a \rangle, \langle \beta, b \rangle, \langle \gamma, c \rangle)$ and $L(\langle \beta, b \rangle, \langle \gamma, c \rangle, \langle \delta, d \rangle)$.

10 Conclusion

We have provided a simple and transparent semantical model for Reo, in which connectors are defined as relations on timed data streams. We use coinduction to reason about both time streams and data streams, leading to some initial formal results on expressiveness and connector equivalence in Reo. Our work on Reo and this model is on-going. One of the first questions to address is to decide what set(s) of basic channels and connectors to choose as the basis for a connector calculus (or calculi). Another plan is to look at more instances of connector protocol verification. On the basis of the example of Section 9 (and other examples not included in the present paper, such as the alternating bit protocol), we expect that the present model will be competitive with both traditional dataflow networks and with process algebra, by combining the best of those two worlds. Like data flow and unlike process algebra, Reo is channel-based and models the (communication) topology of connectors explicitly. Like process algebra and unlike data flow, (our model of) Reo is a calculus in which complex connectors are compositionally built out of simpler ones. Moreover, unlike data flow, the model for Reo that we presented is both simple and formal enough to allow actual verification. And unlike process algebra, there is no need to use nondeterministic transition systems and computationally complicated notions such as weak or branching bisimulation. Instead, streams and coinduction are all that is needed.

Acknowledgements

Many thanks to Jaco de Bakker, Falk Bartels, Frank de Boer, Clemens Kupke, Alexander Kurz, and the members of the ACG Colloquium, for many stimulating discussions, suggestions, and corrections.

References

[AM02] F. Arbab and F. Mavaddat. Coordination through channel composition. In *Proceedings of Coordination Languages and Models*, volume 2315 of *Lecture Notes in Computer Science*. Springer, 2002.

[Arb02] F. Arbab. A channel-based coordination model for component composition. Report SEN-R0203, CWI, 2002. Available at URL www.cwi.nl.

[Bar01] L. Barbosa. *Components as Coalgebras*. PhD thesis, Universidade do Minho, Braga, Portugal, 2001.

[BRP86] H. Barringer, R.Kuiper, and A. Pnueli. A really abstract concurrent model and its temporal logic. In *Proceedings of the 13th Annual ACM Symposium on Principles of Programming Languages*, pages 173–183. ACM, 1986.

[BS01] M. Broy and K. Stølen. *Specification and development of interactive systems*, volume 62 of *Monographs in Computer Science*. Springer, 2001.

[Dob02] E.-E. Doberkat. Pipes and filters: modelling a software architecture through relations. Report 123, Chair for software technology, University of Dortmund, 2002.

[JR97] Bart Jacobs and Jan Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of the EATCS*, 62:222–259, 1997. Available at URL: www.cwi.nl/∼janr.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1980.

[Par81] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proceedings 5th GI conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

[Rut00] J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000.

[Rut01] J.J.M.M. Rutten. Elements of stream calculus (an extensive exercise in coinduction). In Stephen Brooks and Michael Mislove, editors, *Proceedings of MFPS 2001: Seventeenth Conference on the Mathematical Foundations of Programming Semantics*, volume 45 of *Electronic Notes in Theoretical Computer Science*, pages 1–66. Elsevier Science Publishers, 2001.

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.