Tiles for Reo[⋆]

Farhad Arbab[1], Roberto Bruni[2], Dave Clarke[3], Ivan Lanese[4], and Ugo Montanari[2]

[1] CWI, Amsterdam, The Netherlands
farhad@cwi.nl
[2] Dipartimento di Informatica, Università di Pisa, Italy
{bruni,ugo}@di.unipi.it
[3] Department of Computer Science, Katholieke Universiteit Leuven, Belgium
Dave.Clarke@cs.kuleuven.be
[4] Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy
lanese@cs.unibo.it

Abstract. Reo is an exogenous coordination model for software components. The informal semantics of Reo has been matched by several proposals of formalization, exploiting co-algebraic techniques, constraint-automata, and coloring tables. We aim to show that the Tile Model offers a flexible and adequate semantic setting for Reo, such that: (i) it is able to capture context-aware behavior; (ii) it is equipped with a natural notion of behavioral equivalence which is compositional; (iii) it offers a uniform setting for representing not only the ordinary execution of Reo systems but also dynamic reconfiguration strategies.

1 Introduction

Reo [1,7,8] is an exogenous coordination model for software components. It is based on channel-like connectors that mediate the flow of data and signals among components. Notably, a small set of point-to-point primitive connectors is sufficient to express a large variety of interesting constraints over the behavior of connected components, including various forms of mutual exclusion, synchronization, alternation, and context-dependency. In fact, components and primitive connectors can be composed in a circuit fashion via suitable attach points, called Reo nodes. Typical primitive connectors are the synchronous / asynchronous / lossy channels and the asynchronous one-place buffer. The informal semantics of Reo has been formalized in several ways, exploiting co-algebraic techniques [2], constraint-automata [3], and coloring tables [5]. However all the formalizations in the literature that we are aware of are unsatisfactory from some points of view. In fact, both [2] and [3] provide detailed characterizations of the behavior of connectors, allowing to exploit coinductive techniques, but they do not support context-awareness, and, in particular, they are not able to faithfully model the LossySync connector. Up to now, the only approach that takes context-awareness into account is the 3-color semantics presented in [5]. This semantics, however, describes only a single computational step, thus it does not describe the evolution of the state

⋆ Research supported by the project FET-GC II IST-2005-16004 SENSORIA, by the Italian FIRB project TOCAI, by the Dutch NWO project n. 612.000.316 C-Quattro, and by the bilateral German-Dutch DFG-NWO project n. 600.643.000.05N12 SYANCO.

of a connector. Also, none of these semantics allows reconfiguration, which then, for instance as in [12], has to be added on top of them. The interplay between the dataflow semantics of Reo circuits and their reconfiguration has been considered in [11] and [10] using graph transformations triggered by the 3-color semantics.

We aim to show that the Tile Model [9] offers a flexible and adequate semantic setting for Reo. The name 'tile' is due to the graphical representation of such rules (see Fig. 5 in Section 3). The tile α states that the *initial configuration s* can be triggered by the event a to reach the *final configuration t*, producing the *effect b*. Tiles resemble Gordon Plotkin's SOS inference rules [17], but they can be composed in three different ways to generate larger proof steps: (i) horizontally (synchronization), when the effect of one tile matches the trigger for another tile; (ii) vertically (composition in time), when the final configuration of one tile matches the initial configuration of another tile; and (iii) in parallel (concurrency). Tiles take inspiration from Andrea Corradini and Ugo Montanari's Structured Transition Systems [6] and generalise Kim Larsen and Liu Xinxin's context systems [13], by allowing for more general rule formats. The Tile Model also extends José Meseguer's rewriting logic [15] (in the non-conditional case) by taking into account rewrite with side effects and rewrite synchronization. As rewriting logic, the Tile Model admits a purely logical formulation, where tiles are seen as sequents subject to certain inference rules.

Roughly, in our tile encoding, Reo nodes and primitive connectors are represented as hyper-edges (with typed incoming and outgoing tentacles) that can be composed by connecting their tentacles. The one-step semantics of each primitive connector C is defined by suitable basic tiles whose initial configuration is the hyper-edge C (we use the same notation for primitive connectors and corresponding hyper-edges) and whose triggers and effects define how the data can flow through C.

A mapping of a fragment of Reo into the Tile Model has been already presented in [4]. There the emphasis was on exploiting for Reo connectors the normalization and axiomatization techniques developed therein for the used algebra of tile connectors. For this reason the mapping concentrated only on the synchronization connectors, i.e., data values were abstracted away, and data-sensitive connectors such as filters or stateful connectors such as buffers were not considered. The reason was that axiomatization for those more complex connectors was not available. The induced semantics corresponded to the data-insensitive 2-color semantics of Reo [5].

In this paper we extend the mapping in [4] to deal with all Reo connectors, and we concentrate on the 3-color semantics [5], the only one which captures context-awareness. The 3-color semantics for Reo that we propose in Section 6 recovers the good properties of the semantics in the literature, and provides also some additional benefits:

- it allows to model context dependency, and models faithfully the 3-color semantics of [5] as far as a single computational step is concerned;
- it is data-sensitive, describing the actual data that flow inside the connector;
- it can model whole computations, keeping into account the evolution of the state;
- it has a natural notion of behavioral equivalence, tile bisimilarity, that allows to exploit coinductive techniques similar to the ones in [2,3];
- the provided notion of bisimilarity is a congruence, i.e. the behavioral semantics is compositional;

– the congruence property can be easily proved by exploiting standard meta-theoretical results;
– it can be smoothly extended to deal with some form of reconfiguration (Section 7), and the extension also specifies in a formal way the interplay between computation and reconfiguration.

To clarify the approach we first model the simpler 2-color semantics and then show how to handle the 3-color case. In both cases we consider a data-sensitive semantics. Interestingly, the two semantics can be expressed in the same setting (and in a very similar way). Also, they give rise in a natural way to a notion of behavioral equivalence called tile bisimilarity, which is compositional. Finally, we hint at how the same setting can be exploited to model Reo reconfigurations, an aspect that is not considered by the standard Reo semantics. A more detailed treatment of this complex task is left for future work.

Structure of the paper. In Sections 2 and 3 we give some minimal background on Reo and Tile Logic. In Section 4 we define the representation of Reo graphs of connectors in terms of tile configurations. Sections 5 and 6 are dedicated respectively to the modeling of the 2-color and the 3-color semantics. Section 7 outlines the modeling of Reo reconfiguration. Concluding remarks are given in Section 8, together with some hints on future work we have in mind.

2 Reo Connectors

Reo [1,7,8] allows compositional construction of complex connectors with arbitrary behavior out of simpler ones. The simplest (atomic) connectors in Reo consist of a user defined set of *channels*. A channel is a binary connector: a medium of communication with exactly two directed ends. There are two types of channel ends: source and sink. A source channel end accepts data into its channel. A sink channel end dispenses data out of its channel. Every channel (type) specifies its own particular behavior as *constraints* on the flow of data through its ends. These constraints relate, for example, the content, the conditions for loss and/or creation of data that pass through the ends of a channel, as well as the atomicity, exclusion, order, and/or timing of their passage.

Although all channels used in Reo are user-defined and users can indeed define channels with any complex behavior (expressible in the semantic model) that they wish, a very small set of channels, each with very simple behavior, suffices to construct useful Reo connectors with significantly complex behavior [8]. Figure 1 shows a common set of primitive channels often used to build Reo connectors.

The Sync channel takes a data item from its source end and synchronously makes it available at its sink end. This transfer can succeed only if both ends are ready to communicate. The LossySync has the same behavior, except that it does not block its

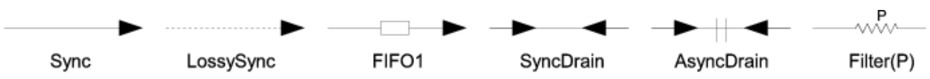


Fig. 1. A typical set of Reo channels

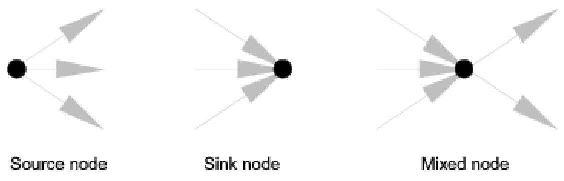
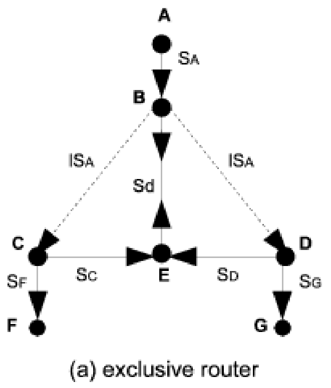


Fig. 2. Reo nodes

writer if its reader end cannot accept data. In this and only this case, the channel accepts the written data item and loses it. The FIFO1 is an asynchronous channel that has a buffer of size one. Unlike the prior channels, FIFO1 is a stateful channel: its behavior depends on whether its buffer is empty or full. The SyncDrain channel has two source ends (and no sink end) through which it can only consume data. It behaves as follows: if and only if there are data items available at both ends, it consumes (and loses) both of them atomically. The AsyncDrain is the asynchronous counterpart of the SyncDrain: it consumes and loses data items from either of its two ends only one at a time, but never from both ends together at the same time. Filter(P) is a synchronous channel with a data-sensitive behavior: it accepts through its source end and loses any data items that do *not* match its filter pattern *P*; it accepts a data item that matches P only if it can synchronously dispose of it through its sink end (exactly as if it were a Sync channel).

A channel end can be composed with other channel ends into Reo *nodes* to build more complex connectors. Reo nodes are logical places where channel ends coincide and coordinate their dataflows as prescribed by node types. Figure 2 shows the three possible node types in Reo. A node with only source channel ends is a *source node*; a node with only sink channel ends is a *sink node*; and a node with both source and sink channel ends is a *mixed node*. The term *boundary nodes* is also sometimes used to collectively refer to source and sink nodes. Boundary nodes define the interface of a connector. Components connect to the boundary nodes of a connector and interact anonymously with each other through this interface by performing I/O operations on the boundary nodes of the connector: *take* operations on sink nodes, and *write* operations on source nodes.

Reo fixes the semantics of (i.e., the constraints on the dataflow through) Reo nodes. Data flow through a source node only if a write operation *offers* a data item on this node and every one of its source channel ends can *accept* a copy of this data item. A source node, thus, behaves as a synchronized replicator. Data flow through a sink node only if at least one of its sink channel ends *offers* a data item and an input operation pending on this node can *accept* this data item. If more than one sink channel end offers data, the node picks one non-deterministically and excludes the offers of all the rest. A sink node, thus, behaves as a non-deterministic merger. The behavior of a mixed node is a combination of that of the other two: data flow through a mixed node only if at least one of its sink channel ends *offers* a data item and every one of its source channel ends can *accept* a copy of this data item. If more than one sink channel end offers data, the node picks one non-deterministically and excludes the offers of all the rest. Because a node has no buffer, data cannot be stored in a node. Hence, nodes instigate the propagation of synchrony and exclusion constraints on dataflow throughout a connector.



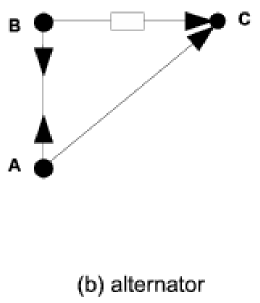


Fig. 3. Reo circuit for (a) exclusive router (from A to either F or G) and (b) Alternator

The simplest formalization of this behavior is the 2-color semantics presented in [5]. The two colors ■/□ model the flow/absence-of-flow of data at each node respectively (this is the so-called data-insensitive semantics; instead if different colors are used to distinguish the kind of data one obtains a data-sensitive semantics). This coloring must satisfy the constraint conditions imposed by connectors. Each connector determines the possible color combinations on its ends. For instance, both ends of Sync must have the same color (i.e. either the datum flows through the whole connector or no data flow at all), while AsyncDrain allows any coloring but (■,■), which would represent data flowing synchronously at both of its ends. All channel ends connected to a □ node must be colored by □, while for ■ nodes, exactly one of the incoming channel ends, and all the outgoing channel ends, must have the ■ color. Deriving the semantics of a Reo connector amounts to resolving the composition of the constraints of its constituent channels and nodes. Given a connector C a coloring c for C is a function associating a color to each node in C. The 2-color semantics of C is given by its coloring table T_C, which contains all of its allowed colorings. For instance the coloring table of a connector with two nodes A and B connected by a Sync connector is $T = \{[A \mapsto ■, B \mapsto ■], [A \mapsto □, B \mapsto □]\}$.

In Fig. 3 we present two examples of Reo connectors that illustrate how non-trivial dataflow behavior emerges from composing simple channels using Reo nodes. The local constraints of individual channels propagate through (the synchronous regions of) a connector to its boundary nodes. This propagation also induces a certain context-awareness in connectors. See [5] for a detailed discussion of this.

The connector shown in Fig. 3(a) is an *exclusive router*: it routes data from A to either F or G (but not both). This connector can accept data only if there is a write operation at the source node A, and there is at least one taker at the sink nodes F and G. If both F and G can dispense data, the choice of routing to F or G follows from the non-deterministic decision by the mixed node E: E can accept data only from one of its sink ends, excluding the flow of data through the other, which forces the latter's respective LossySync to lose the data it obtains from A, while the other LossySync passes its

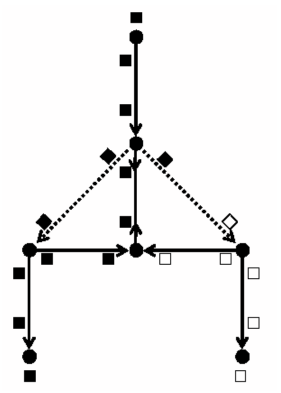


Fig. 4. A 2-coloring example for the exclusive router

data as if it were a Sync. A valid coloring of the exclusive router is shown in Fig. 4. The case shown in Fig. 4 corresponds to the forwarding of the data available on node A to the node F but not to G. There are two other possible 2-colorings for the exclusive router: one representing the case where the flow goes from A to G and not to F (i.e. the mirrored diagram w.r.t. Fig. 4) and one representing no dataflow (all the boxes are empty).

The connector shown in Fig. 3(b) is an *alternator* that imposes an ordering on the flow of the data from its input nodes A and B to its output node C. The SyncDrain enforces that data flow through A and B only synchronously. The empty buffer together with the SyncDrain guarantee that the data item obtained from A is delivered to C while the data item obtained from B is stored in the FIFO1 buffer. After this, the buffer of the FIFO1 is full and data cannot flow in through either A or B, but C can dispense the data stored in the FIFO1 buffer, which makes it empty again.

3 Tile Logic

Reo connectors are naturally represented as graphs. The advantage of using (freely generated) symmetric monoidal categories for representing configuration graphs is two-fold. First, it introduces a suitable notion of (observable) interfaces for configurations. Second, the natural isomorphism defined by symmetries allows to take graphs up to interface-preserving graph isomorphisms.

We recall that a *(strict) monoidal category* [14] $(\mathcal{C},\otimes,e)$ is a category $\mathcal{C}$ together with a functor $\otimes\colon \mathcal{C}\times\mathcal{C}\to\mathcal{C}$ called the *tensor product* and an object e called the *unit*, such that for any arrows $\alpha_1,\alpha_2,\alpha_3\in\mathcal{C}$ we have $(\alpha_1\otimes\alpha_2)\otimes\alpha_3=\alpha_1\otimes(\alpha_2\otimes\alpha_3)$ and $\alpha_1\otimes id_e=\alpha_1=id_e\otimes\alpha_1$. The tensor product has higher precedence than the categorical composition ;. Note that we focus only on "strict" monoidal categories, where the monoidal axioms hold as equalities and not just up to natural isomorphisms. By functoriality of $\otimes$ we have, e.g., $\alpha_1\otimes\alpha_2=\alpha_1\otimes id_{a_2};id_{b_1}\otimes\alpha_2=id_{a_1}\otimes\alpha_2;\alpha_1\otimes id_{b_2}$ for any $\alpha_i\colon a_i\to b_i, i\in\{1,2\}$.

Definition 1 (symmetric monoidal categories). *A* symmetric (strict) monoidal category $(\mathcal{C},\otimes,e,\gamma)$ *is a (strict) monoidal category* $(\mathcal{C},\otimes,e)$ *together with a family of arrows* $\{\gamma_{a,b}\colon a\otimes b \to b\otimes a\}_{a,b}$*, called* symmetries*, indexed by pairs of objects in* $\mathcal{C}$ *such that for any two arrows* $\alpha_1,\alpha_2 \in \mathcal{C}$ *with* $\alpha_i\colon a_i \to b_i$*, we have* $\alpha_1\otimes\alpha_2;\gamma_{b_1,b_2} = \gamma_{a_1,a_2};\alpha_2\otimes\alpha_1$ *(that is,* γ *is a natural isomorphism) that satisfies the coherence equalities (for any objects* a,b,c*):*

$$\gamma_{a,b};\gamma_{b,a} = id_{a\otimes b} \qquad\qquad \gamma_{a\otimes b,c} = id_a\otimes\gamma_{b,c};\gamma_{a,c}\otimes id_b.$$

The categories we are interested in are those freely generated from a sorted (hyper)signature Σ, i.e., from a sorted family of operators $f\colon \tau_i \to \tau_f$. The objects are words on some alphabet S expressing the sorts of interfaces (we use ε to denote the empty word). Consider, e.g., $S = \{\bullet,\circ\}$. Then $f\colon \bullet\circ \to \bullet\bullet$ means that f has two "attach points" on both the interfaces, with types $\bullet\circ$ for the initial one and $\bullet\bullet$ for the final one. The operators $\sigma\in\Sigma$ are seen as basic arrows with source and target defined according to the sort of σ. Symmetries can always be expressed in terms of the basic sorted symmetries $\gamma_{x,y}\colon x\otimes y \to y\otimes x$. Intuitively, symmetries can be used to rearrange the input-output interfaces of graph-like configurations.

In this paper, we choose the Tile Model [9] for defining the operational and observational semantics of Reo connectors. In fact, tile configurations are particularly suitable to represent the above concept of connector, which includes input and output interfaces where actions can be observed and that can be used to compose configurations and also to coordinate their local behaviors.

A tile $\alpha : s \xrightarrow[b]{a} t$ is a rewrite rule stating that the *initial configuration s* can evolve to the *final configuration t* via α, producing the *effect b*; but the step is allowed only if the 'arguments' of s can contribute by producing a, which acts as the *trigger* of α (see Fig. 5(i)). Triggers and effects are called *observations* and tile vertices are called *interfaces*.

Tiles can be composed horizontally, in parallel, or vertically to generate larger steps (see Fig. 5). Horizontal composition $\alpha;\beta$ coordinates the evolution of the initial configuration of α with that of β, yielding the 'synchronization' of the two rewrites. Horizontal composition is possible only if the initial configurations of α and β interact cooperatively: the effect of α must provide the trigger for β. Vertical composition $\alpha * \beta$ is sequential composition of computations. The parallel composition $\alpha\otimes\beta$ builds concurrent steps.

The operational semantics of concurrent systems can be expressed via tiles if system configurations form a monoidal category $\mathcal{H}$, and observations form a monoidal category

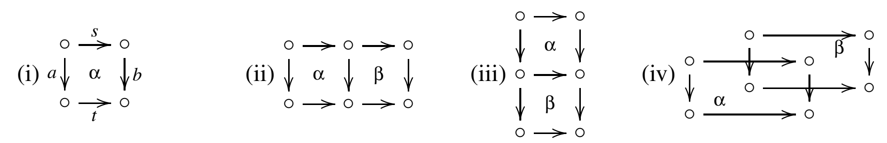


Fig. 5. Examples of tiles and their composition

$\mathcal{V}$ with the same underlying set of objects as $\mathcal{H}$. Abusing the notation, we denote by $\_\otimes\_$ both monoidal functors of $\mathcal{H}$ and $\mathcal{V}$ and by $\_;\_$ both sequential compositions in $\mathcal{H}$ and $\mathcal{V}$.

Definition 2 (tile system). *A* tile system *is a tuple $\mathcal{R} = (\mathcal{H},\mathcal{V},N,R)$ where $\mathcal{H}$ and $\mathcal{V}$ are monoidal categories with the same set of objects $O_{\mathcal{H}} = O_{\mathcal{V}}$, N is the set of rule names and $R\colon N \to \mathcal{H}\times\mathcal{V}\times\mathcal{V}\times\mathcal{H}$ is a function such that for all $A \in N$, if $R(A) = \langle s,a,b,t\rangle$, then the arrows s,a,b,t can form a tile like in Fig. 5(i).*

Like rewrite rules in rewriting logic, tiles can be seen as sequents of *tile logic*: the sequent $s \xrightarrow[b]{a} t$ is *entailed* by the tile logic associated with $\mathcal{R}$, written $\mathcal{R} \vdash s \xrightarrow[b]{a} t$, if it can be obtained by horizontal, parallel, and/or vertical composition of some basic tiles in R, plus possibly some auxiliary tiles such as identities $id \xrightarrow[a]{a} id$ which propagate observations, and horizontal symmetries $\gamma \xrightarrow[b\otimes a]{a\otimes b} \gamma$ which swap the order in which concurrent observations are attached to the left and right interfaces. The "borders" of composed sequents are defined in Fig. 6.

The main feature of tiles is their double labeling with triggers and effects, allowing to observe the input-output behavior of configurations. By taking $\langle\text{trigger},\text{effect}\rangle$ pairs as labels one can see tiles as a labeled transition system. In this context, the usual notion of bisimilarity is called *tile bisimilarity*.

Definition 3 (tile bisimilarity). *Let $\mathcal{R} = (\mathcal{H},\mathcal{V},N,R)$ be a tile system. A symmetric relation $\sim_{\mathrm{t}}$ on configurations is called a* tile bisimulation *if whenever $s \sim_{\mathrm{t}} t$ and $\mathcal{R} \vdash s \xrightarrow[b]{a} s'$, then t' exists such that $\mathcal{R} \vdash t \xrightarrow[b]{a} t'$ and $s' \sim_{\mathrm{t}} t'$.*

The maximal tile bisimulation is called tile bisimilarity *and it is denoted by $\simeq_{\mathrm{t}}$.*

$$\frac{s \xrightarrow[b]{a} t \quad h \xrightarrow[c]{b} f}{s;h \xrightarrow[c]{a} t;f}\ (\texttt{hor}) \qquad \frac{s \xrightarrow[b]{a} t \quad h \xrightarrow[d]{c} f}{s\otimes h \xrightarrow[b\otimes d]{a\otimes c} t\otimes f}\ (\texttt{par}) \qquad \frac{s \xrightarrow[b]{a} t \quad t \xrightarrow[d]{c} h}{s \xrightarrow[b;d]{a;c} h}\ (\texttt{ver})$$

Fig. 6. Inference rules for tile logic

Note that $s \simeq_{\mathrm{t}} t$ only if s and t have the same input-output interfaces.

The basic source property is a syntactic criterion ensuring that tile bisimilarity is a congruence.

Definition 4 (basic source property). *A tile system $\mathcal{R} = (\mathcal{H},\mathcal{V},N,R)$ enjoys the basic source property if for each $A \in N$ if $R(A) = \langle s,a,b,t\rangle$, then s is an operator in Σ.*

The following result from [9] can be used to ensure that tile bisimilarity is a congruence.

Lemma 1. *If a tile system $\mathcal{R}$ enjoys the basic source property, then tile bisimilarity is a congruence for $\mathcal{R}$.*

4 From Reo Connectors to Tile Configurations

In order to give semantics to Reo connectors using tile logic, we first need to map them into tile configurations. The basic entities of Reo connectors are nodes and channels, which are then composed by plugging channels into nodes. Here we consider Reo nodes as composed out of replicators, mergers, and basic nodes, as in [5], since this will simplify our mapping. A replicator is a ternary atomic connector with one source and two sink ends. A merger is a ternary atomic connector with two source and one sink ends. A basic node is one that has at most one source and at most one sink ends. Essentially, a node N with $n > 1$ incoming and $m > 1$ outgoing channel ends will be represented by a basic node with one incoming tree of $n-1$ mergers and one outgoing tree of $m-1$ replicators. Incoming channel ends of N will be connected to the leaves of the tree of mergers, and outgoing channel ends of N will be connected to the leaves of the tree of replicators.

The horizontal signature of the tile system for modeling Reo connectors, thus, includes operators for basic nodes, mergers, replicators, and channels. As usual, when modeling graphs with tiles (see, e.g., [16]), nodes are on the left, with their interfaces heading toward right, and channels are on the right with their interfaces toward left. The two interfaces are joined using symmetries, mergers and replicators. Notice that this technique for representing graphs is fully general, i.e. any graph can be represented in this style. Since we do not model components explicitly, boundary nodes are nodes with a non-connected element of their right interface (the sink for source nodes, the source for sink nodes). Interfaces are typed according to the direction of flow of data: $\bullet$ for data going from left to right (from nodes to channels) and $\circ$ for data going from right to left (from channels to nodes). Thus, e.g., the Merger operator has sort $\mathsf{Merger} : \circ \to \circ\circ$. This denotes the fact that data flow in the Merger operator from right to left. Similarly the Sync channel has sort $\mathsf{Sync} : \bullet\circ \to \varepsilon$, with an empty right interface as for all channels. Note that the order of elements in the interface matters. However, symmetries can be used to reorder the elements in an interface as necessary. A basic node, with one sink end and one source end has sort $\mathsf{Node} : \varepsilon \to \circ\bullet$. The full horizontal signature of our tile system (for a sample set of basic connectors) is presented in Fig. 7: on the left-hand side in textual notation, and on the right-hand side in graphical notation, where for simplicity we abbreviate the names of operators using their initials.

The tile model for a general node, with n sink and m source ends is obtained by composing the tiles of a basic node, $n-1$ mergers, and $m-1$ replicators, as explained above. For instance, a node with 2 sinks and 3 sources is: $\mathsf{Node}; \mathsf{Merger} \otimes \mathsf{Replicator}; id_{\circ\circ\bullet} \otimes \mathsf{Replicator} : \varepsilon \to \circ\circ\bullet\bullet\bullet$ (see Fig. 8).

We can now define the mapping $[\![\cdot]\!]$ from Reo connectors to tile configurations. If a connector C has n boundary nodes, then $[\![C]\!] : \varepsilon \to \omega$ where $\omega \in \{\bullet, \circ\}^n$ is a word of length n. The mapping is parametric with respect to an interface function I_n associating to each boundary node in C an element in ω, i.e. I is a bijection between nodes of C and $\{1, \ldots, n\}$.

Definition 5 (from Reo to tile configurations). *Given a Reo connector C with n boundary nodes and an interface function I_n, the tile configuration $[\![C]\!]_{I_n}$ is defined as follows:*

Replicator	$\bullet \to \bullet\bullet$	Merger	$\circ \to \circ\circ$
Node	$\varepsilon \to \circ\bullet$	Sync	$\bullet\circ \to \varepsilon$
SyncDrain	$\bullet\bullet \to \varepsilon$	LossySync	$\bullet\circ \to \varepsilon$
SyncSpout	$\circ\circ \to \varepsilon$	AsyncDrain	$\bullet\bullet \to \varepsilon$
AsyncSpout	$\circ\circ \to \varepsilon$	Filter(p)	$\bullet\circ \to \varepsilon$
FIFO1	$\bullet\circ \to \varepsilon$	FIFO1(x)	$\bullet\circ \to \varepsilon$

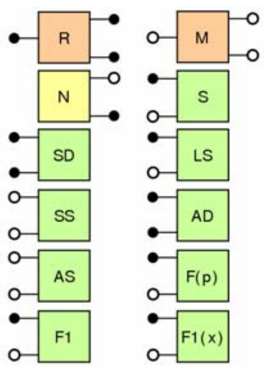


Fig. 7. Signature for Reo configurations

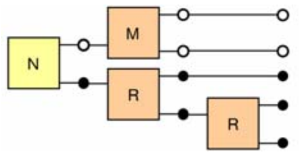


Fig. 8. A tile configuration representing a mixed node

- *on the left, it has a parallel composition of* Node *operators, one for each node in C, with the two ends connected to trees composed by $n-1$ mergers and $m-1$ replicators respectively, if the Reo node has n sources and m sinks (the trees may be empty); for boundary nodes one of the two attach points has no connected tree, and will be connected to the outside interface;*
- *on the right, it has a parallel composition of channel operators, one for each channel in C;*
- *the two parts are connected via identities and symmetries, so that each incoming channel is connected to the* Merger *tree of the corresponding node, and similarly for outgoing channels and* Replicator *trees;*
- *for each boundary node A, its free attach point is connected to the interface element $I_n(A)$ via identities and symmetries.*

The tile configurations corresponding to the Reo connectors that define the exclusive router and the alternator are presented in Fig. 9. The corresponding textual notation for the alternator is below (where *Perm* is a composition of identities and symmetries):

$$\mathsf{Node} \otimes \mathsf{Node} \otimes \mathsf{Node}; id_\circ \otimes \mathsf{Replicator} \otimes id_\circ \otimes \mathsf{Replicator} \otimes \mathsf{Merger} \otimes id_\bullet; Perm;$$
$$id_{\circ\otimes\circ} \otimes \mathsf{SyncDrain} \otimes \mathsf{Sync} \otimes \mathsf{FIFO1} \otimes id_\bullet : \varepsilon \to \circ\circ\bullet$$

Now we can give semantics to Reo connectors via tiles.

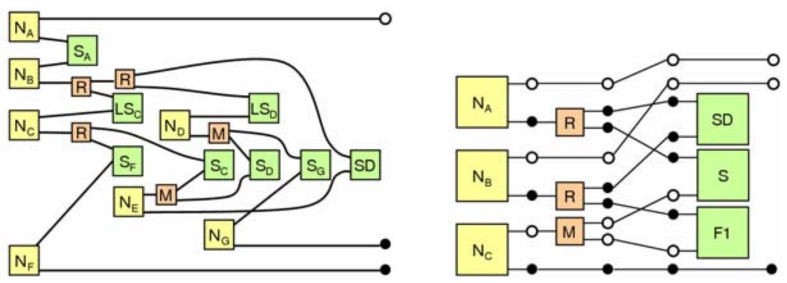


Fig. 9. Exclusive router and alternator as tile configurations

5 Modeling the 2-Color Semantics of Reo

The one-step tile semantics of a connector C is the set of all tiles that have C as their starting configuration. In order to give semantics to Reo connectors we need to provide the basic tiles defining the semantics of each operator, and then tile composition operations allow the derivation of the semantics of general connectors. We begin by presenting the 2-color data-sensitive semantics, which cannot express context-dependent behavior, but which is simpler than the corresponding 3-color semantics that we will introduce in the next section.

We choose as basic observations the data communicated at the interfaces of connectors, to model data-sensitive semantics, and we consider also a special observation untick to denote no data communication. For instance, the tile $\mathsf{Merger} \xrightarrow[a\otimes\mathsf{untick}]{a} \mathsf{Merger}$ allows a Merger connector to get an action a from the first element in its right interface and propagate it to its left interface, provided that there is no piece of data on the other element of the right interface.

The basic tiles are described in Fig. 10, assuming an alphabet **Act** for basic actions. We also assume that x and y range over $\mathbf{Act} \cup \{\mathsf{untick}\}$ and a and b range over **Act**. A graphical representation of the tile that models the filling of a FIFO1 buffer is in Fig. 11. Note that observations on the interface are drawn along the vertical dimension.

These tiles define an LTS semantics for Reo, where states are tile configurations and observations are $\langle$trigger, effect$\rangle$ pairs. This semantics recovers all the information in the 2-color tile semantics for Reo described in [5]. Furthermore it adds to it: (i) the possibility of observing the actual data flowing in the connector, allowing to model data-sensitive primitive connectors such as filters, (ii) the possibility to consider full computations instead of single steps, keeping track also of how the state evolves (particularly, whether buffers get full or become empty). The theorem below shows how the information provided by the 2-color semantics can be recovered from the tile semantics. We call a connector data-insensitive if its behavior (i.e., whether or not it allows data to flow) does not depend on data values. Specifically, every connector built using any of the primitive connectors described above, excluding filters, is a data-insensitive connector. To formalize the correspondence between our tile model and the 2-color semantics, we must restrict tiles to the one-step semantics of the connectors, and therefore

$$\gamma \xrightarrow[y\otimes x]{x\otimes y} \gamma \qquad \text{Replicator} \xrightarrow[x\otimes x]{x} \text{Replicator} \qquad \text{Node} \xrightarrow[x\otimes x]{id_\varepsilon} \text{Node}$$

$$\text{Merger} \xrightarrow[a\otimes\text{untick}]{a} \text{Merger} \qquad \text{Merger} \xrightarrow[\text{untick}\otimes a]{a} \text{Merger} \qquad \text{Merger} \xrightarrow[\text{untick}\otimes\text{untick}]{\text{untick}} \text{Merger}$$

$$\text{Sync} \xrightarrow[id_\varepsilon]{x\otimes x} \text{Sync} \qquad \text{SyncDrain} \xrightarrow[id_\varepsilon]{a\otimes b} \text{SyncDrain} \qquad \text{SyncDrain} \xrightarrow[id_\varepsilon]{\text{untick}\otimes\text{untick}} \text{SyncDrain}$$

$$\text{SyncSpout} \xrightarrow[id_\varepsilon]{a\otimes b} \text{SyncSpout} \qquad \text{SyncSpout} \xrightarrow[id_\varepsilon]{\text{untick}\otimes\text{untick}} \text{SyncSpout}$$

$$\text{AsyncDrain} \xrightarrow[id_\varepsilon]{x\otimes\text{untick}} \text{AsyncDrain} \qquad \text{AsyncDrain} \xrightarrow[id_\varepsilon]{\text{untick}\otimes x} \text{AsyncDrain}$$

$$\text{AsyncSpout} \xrightarrow[id_\varepsilon]{x\otimes\text{untick}} \text{AsyncSpout} \qquad \text{AsyncSpout} \xrightarrow[id_\varepsilon]{\text{untick}\otimes x} \text{AsyncSpout}$$

$$\text{LossySync} \xrightarrow[id_\varepsilon]{x\otimes x} \text{LossySync} \qquad \text{LossySync} \xrightarrow[id_\varepsilon]{a\otimes\text{untick}} \text{LossySync}$$

$$\text{FIFO1} \xrightarrow[id_\varepsilon]{a\otimes\text{untick}} \text{FIFO1(a)} \qquad \text{FIFO1} \xrightarrow[id_\varepsilon]{\text{untick}\otimes\text{untick}} \text{FIFO1}$$

$$\text{FIFO1(a)} \xrightarrow[id_\varepsilon]{\text{untick}\otimes a} \text{FIFO1} \qquad \text{FIFO1(a)} \xrightarrow[id_\varepsilon]{\text{untick}\otimes\text{untick}} \text{FIFO1(a)}$$

$$\text{Filter(P)} \xrightarrow[id_\varepsilon]{a\otimes a} \text{Filter(P) if } P(a) \qquad \text{Filter(P)} \xrightarrow[id_\varepsilon]{a\otimes\text{untick}} \text{Filter(P) if } \neg P(a)$$

Fig. 10. Tiles for data-sensitive, 2-color semantics

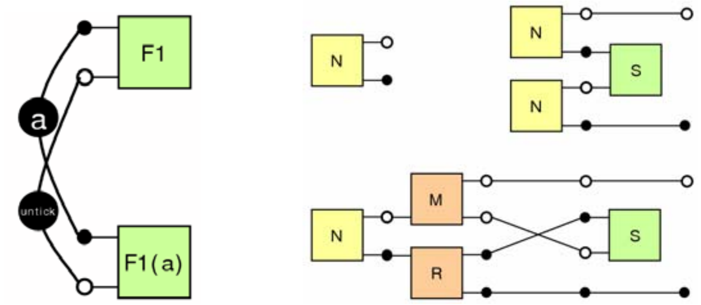


Fig. 11. The tile for filling a FIFO1 buffer (left) and three bisimilar configurations (right)

we do not need vertical composition of tiles. However, including vertical composition does not add any one-step transition either.

Theorem 1 (correspondence between 2-color coloring tables and tiles). *Let T_C be the 2-color coloring table of a data-insensitive Reo connector C with n boundary nodes. T_C contains a coloring c iff for each interface function I_n there exists a tile obtained without using vertical composition having as initial configuration $[\![C]\!]_{I_n}$ such that, for each node A, $c(A) = \square$ iff the observation at the interface $I_n(A)$ in the tile is* untick.

Proof (Sketch). First notice that there is a bijection between colorings for channels, mergers and replicators and the basic tiles that have the corresponding operators as their starting configurations, i.e. a basic connector allows a coloring c iff there is a basic tile with that operator as its starting configuration and observation untick on an interface iff the corresponding node has color $\square$ in the coloring.

One has to prove that the correspondence is preserved while composing colorings on one side and tiles on the other side. We consider the left-to-right implication, the other being simpler. Colorings can be composed iff they agree on the color of their common nodes (see Definition 3 in [5]). In order to compose the corresponding tiles to derive a tile with the desired starting configuration, observations on matching interfaces have to coincide. Let us consider the case of just one possible data value. Then the possibility of composing the tiles follows from the hypothesis if connectors are connected directly (e.g., channels to mergers and replicators), and from the properties of the auxiliary tiles for identities and symmetries and the basic tiles for nodes if connectors are connected via them.

Let us now consider the general case of an arbitrary set of data values. Note that for data-insensitive connectors, if a tile for a certain data flow exists, then a tile with the same data flow, but where all the data are equal can be built (this can be easily proved by induction on the number of operators in the starting configuration of the tile), thus the case of an arbitrary set of data values can be reduced to the one data value case. Notice that the above property does not hold for data-sensitive connectors. $\square$

As we have seen, all information provided by the coloring tables can be deduced from the tile semantics. Furthermore, the final configuration of a tile represents the state of the connector after the data flow has been performed. This can be used also to recover information provided by the constraint-automata or coalgebraic semantics of Reo. However a detailed comparison with those semantics is left for future work.

The theorem below ensures that the tile semantics is compositional w.r.t. the operators of parallel and sequential composition provided by tiles.

Theorem 2 (2-coloring congruence). *Tile bisimilarity is a congruence for the 2-color semantics of Reo connectors.*

Proof. Straightforward by inspection, using Lemma 1. $\square$

Note that the compositionality is proved w.r.t. the operators of tile composition, however this can be extended also to Reo composition operators. Composition in Reo is obtained by merging boundary nodes. In the tile model this can be obtained by connecting them via Sync channels (this corresponds to compose them in parallel and then sequentially with the Sync channel and some identities). The example below shows that the additional channel does not influence the behavior of the composition.

Example 1. Consider the simple Reo connector C_1 composed out of a mixed node with one source end and one sink end, $\mathsf{Node} : \varepsilon \to \circ\bullet$ (see Fig. 11, top-center). We can show that this is bisimilar to a Reo connector C_2 composed out of two such nodes connected by a Sync channel: $\mathsf{Node} \otimes \mathsf{Node}; id_\circ \otimes \mathsf{Sync} \otimes id_\bullet : \varepsilon \to \circ\bullet$ (see Fig. 11, top-right). First, note that the two connectors have the same interface. Then, observe that for both connectors the only possible tiles are vertical compositions of tiles $C_i \xrightarrow[x]{x} C_i$ (with $i = 1$ or $i = 2$). Thus, from the definition of bisimilarity $C_1 \sim_t C_2$. Therefore, thanks to the congruence theorem, in each connector we can replace the two nodes connected by a Sync channel with a single node without changing the overall behavior. A third bisimilar configuration is in Fig. 11, bottom-right.

6 Modeling the 3-Color Semantics of Reo

As pointed out in [5], the 2-color semantics of Reo fails to fully capture the context-dependent behavior of Reo connectors. Consider in fact the connector in Fig. 12, which is represented by the tile configuration:

$\mathsf{Node} \otimes \mathsf{Node} \otimes \mathsf{Node}; id_\circ \otimes \mathsf{LossySync} \otimes \mathsf{FIFO1} \otimes id_\bullet : \varepsilon \to \circ\bullet$

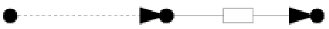


Fig. 12. Overflow-lossy FIFO1

There are two possible tiles with this initial configuration and with the observation $\langle id_\varepsilon, a \otimes \mathsf{untick}\rangle$ modeling data entering in the connector. The first one loses the data item in the LossySync channel and has the final configuration $\mathsf{Node} \otimes \mathsf{Node} \otimes \mathsf{Node}; id_\circ \otimes \mathsf{LossySync} \otimes \mathsf{FIFO1} \otimes id_\bullet$. The second one transports the data item into the buffer of the FIFO1(a) channel and has the final configuration $\mathsf{Node} \otimes \mathsf{Node} \otimes \mathsf{Node}; id_\circ \otimes \mathsf{LossySync} \otimes \mathsf{FIFO1(a)} \otimes id_\bullet$. The expected behavior corresponds to the second one, since there is no reason for the data to be lost. However, both the 2-color semantics and the tile model we presented above generate both alternatives as permissible behavior for this connector.

The 3-color semantics of Reo discussed in [5] solves this problem by tracking 'reasons to prohibit data flow', and allows LossySync to lose data only if there is a reason for the data not to flow out of the channel (e.g., an attached full buffer or an interface that does not accept data at the other end). The 3-color semantics replaces the $\square$ color by two colors corresponding to 'giving a reason for no data flow' and 'requiring a reason for no data flow.' Briefly, 'giving a reason' is used to model either a choice made by the connector or to capture the absence of data flow on a particular channel end. On the other hand, 'requiring a reason' is used to model that the context determines whether a particular choice is made. Consider the two key tiles for LossySync:

$$\mathsf{LossySync} \xrightarrow[id_\varepsilon]{a\otimes a} \mathsf{LossySync} \qquad\qquad \mathsf{LossySync} \xrightarrow[id_\varepsilon]{a\otimes \rhd} \mathsf{LossySync}$$

The first one simply states that data flow through the LossySync. The second states that data will be lost in the LossySync if a reason for no flow can be provided by the context in which the channel is plugged. If a tile with the label $a \otimes \lhd$ was also present, this would say that the LossySync provides a reason for the data to be lost, and thus the LossySync would lose the property that the decision ought to be made by the context.

Composition in the 3-color model includes the additional requirement that at each basic node where there is no data flow, at least one reason for no flow must be present.

We show that tile logic can also easily model this more detailed semantics. To this end, we must refine our untick observation into $\triangleleft$, which models 'requires a reason for no data flow,' and $\triangleright$, which models 'gives a reason for no data flow,' when these symbols occur on the left-hand side of the tile (above the line in the rule format). When these observations occur on the right-hand side of a tile, their meanings are reversed. For instance, one of the rules for Replicator:

$$\mathsf{Replicator} \xrightarrow[\triangleright\otimes\triangleright]{\triangleright} \mathsf{Replicator}$$

means that a reason is required from the channel end on the left of the tile (above the line) and will be given (propagated) to the channel ends on the right of the tile (below the line). This captures that no-input to the Replicator is sufficient to cause no data flow through the Replicator, and that this reason is passed onto the sink ends.

The main tiles for modeling the 3-color semantics of Reo are in Fig. 13. The others are analogous.

$$\gamma \xrightarrow[y\otimes x]{x\otimes y} \gamma \qquad \mathsf{Replicator} \xrightarrow[a\otimes a]{a} \mathsf{Replicator}$$

$$\mathsf{Replicator} \xrightarrow[\triangleright\otimes\triangleright]{\triangleright} \mathsf{Replicator} \qquad \mathsf{Replicator} \xrightarrow[\triangleright\otimes\triangleleft]{\triangleleft} \mathsf{Replicator} \qquad \mathsf{Replicator} \xrightarrow[\triangleleft\otimes\triangleright]{\triangleleft} \mathsf{Replicator}$$

$$\mathsf{Node} \xrightarrow[a\otimes a]{id_\varepsilon} \mathsf{Node} \qquad \mathsf{Node} \xrightarrow[\triangleleft\otimes\triangleright]{id_\varepsilon} \mathsf{Node} \qquad \mathsf{Node} \xrightarrow[\triangleright\otimes\triangleleft]{id_\varepsilon} \mathsf{Node} \qquad \mathsf{Node} \xrightarrow[\triangleleft\otimes\triangleleft]{id_\varepsilon} \mathsf{Node}$$

$$\mathsf{Merger} \xrightarrow[a\otimes\triangleright]{a} \mathsf{Merger} \qquad \mathsf{Merger} \xrightarrow[\triangleright\otimes a]{a} \mathsf{Merger} \qquad \mathsf{Merger} \xrightarrow[\triangleright\otimes\triangleright]{\triangleright} \mathsf{Merger} \qquad \mathsf{Merger} \xrightarrow[\triangleleft\otimes\triangleleft]{\triangleleft} \mathsf{Merger}$$

$$\mathsf{Sync} \xrightarrow[id_\varepsilon]{a\otimes a} \mathsf{Sync} \qquad \mathsf{Sync} \xrightarrow[id_\varepsilon]{\triangleleft\otimes\triangleright} \mathsf{Sync} \qquad \mathsf{Sync} \xrightarrow[id_\varepsilon]{\triangleright\otimes\triangleleft} \mathsf{Sync} \qquad \mathsf{Sync} \xrightarrow[id_\varepsilon]{\triangleright\otimes\triangleright} \mathsf{Sync}$$

$$\mathsf{LossySync} \xrightarrow[id_\varepsilon]{a\otimes a} \mathsf{LossySync} \qquad \mathsf{LossySync} \xrightarrow[id_\varepsilon]{a\otimes\triangleright} \mathsf{LossySync} \qquad \mathsf{LossySync} \xrightarrow[id_\varepsilon]{\triangleright\otimes\triangleleft} \mathsf{LossySync}$$

$$\mathsf{FIFO1} \xrightarrow[id_\varepsilon]{\triangleright\otimes\triangleleft} \mathsf{FIFO1} \qquad \mathsf{FIFO1} \xrightarrow[id_\varepsilon]{a\otimes\triangleleft} \mathsf{FIFO1(a)} \qquad \mathsf{FIFO1(a)} \xrightarrow[id_\varepsilon]{\triangleleft\otimes a} \mathsf{FIFO1} \qquad \mathsf{FIFO1(a)} \xrightarrow[id_\varepsilon]{\triangleleft\otimes\triangleright} \mathsf{FIFO1(a)}$$

Fig. 13. Tiles for data-sensitive, 3-color semantics

Note that the tile Node includes a behavior that mimics the so-called *flip rule* in connector coloring [5]. The point of the flip rule is to reduce the size of coloring tables using the fact that nodes need no more than one reason. The fact that nodes can also accept multiple reasons is captured by the tile:

$$\mathsf{Node} \xrightarrow[\triangleleft\otimes\triangleleft]{id_\varepsilon} \mathsf{Node}$$

Results analogous to the one in the previous section can be proved, showing that the 3-color tile semantics recovers all the information provided by the standard 3-color semantics of Reo. As for the 2-color semantics, the tile semantics is data-sensitive, and allows to track the state of connectors and model full computations.

Theorem 3 (correspondence between 3-color coloring tables and tiles). *Let T_C be the 3-color coloring table (see [5]) of a data-insensitive Reo connector C with n boundary nodes. T_C contains a coloring c iff for each interface function I_n there exists a tile obtained without using vertical composition with initial configuration $[\![C]\!]_{I_n}$ such that, for each node A:*

- *$c(A)$ is the color for no dataflow with the reason coming into the node (given) iff the observation at the interface element $I_n(A)$ in the tile is $\triangleright$ (this is always below the line);*
- *$c(A)$ is the color for no dataflow with the reason leaving the node (required) iff the observation at the interface element $I_n(A)$ in the tile is $\triangleleft$ (this is always below the line).*

Proof. The proof is similar to the one of Theorem 1. □

As for the 2-color semantics, tile bisimilarity is a congruence.

Theorem 4 (3-coloring congruence). *Tile bisimilarity is a congruence for the 3-color semantics of Reo connectors.*

7 Reconfiguration of Reo Connectors

Since the tile semantics of a Reo connector includes also the state of the connector after each step, one can model inside the Tile Model also the reconfiguration of Reo connectors triggered by dataflow as presented in [11].

The idea is that some connectors, when suitable conditions concerning their state and the ongoing dataflow are met, can automatically be reconfigured to meet the requirements of the environment. We sketch this approach by demonstrating it through the example of an infinite FIFO buffer [11], and leave a more detailed study of reconfiguration for future work. An infinite FIFO buffer is a FIFO buffer that grows when a new datum arrives to be inserted and its buffer is full, and shrinks when a datum is consumed out of the buffer. To model this we require two new channels: FIFO_∞ is the empty infinite buffer, and $\mathsf{FIFOtmp}(a)$ is a temporary buffer, containing value a, that will disappear when the a is consumed.

For simplicity we give semantics to the infinite FIFO buffer using the 2-color semantics, however, the 3-color semantics can be used as well. The necessary basic tiles can be found in Fig. 14. Note that the tile for shrinking the buffer transforms the temporary buffer $\mathsf{FIFOtmp}(a)$ into a Sync channel. Thanks to Example 1, up to bisimilarity, this

$$\mathsf{FIFO}_\infty \xrightarrow[id_\varepsilon]{\mathsf{untick}\otimes\mathsf{untick}} \mathsf{FIFO}_\infty \qquad \mathsf{FIFO}_\infty \xrightarrow[id_\varepsilon]{a\otimes\mathsf{untick}} id_\bullet \otimes \mathsf{Node} \otimes id_\circ ; \mathsf{FIFO}_\infty \otimes \mathsf{FIFOtmp(a)}$$

$$\mathsf{FIFOtmp(a)} \xrightarrow[id_\varepsilon]{\mathsf{untick}\otimes a} \mathsf{Sync} \qquad \mathsf{FIFOtmp(a)} \xrightarrow[id_\varepsilon]{\mathsf{untick}\otimes\mathsf{untick}} \mathsf{FIFOtmp(a)}$$

Fig. 14. Tiles for 2-color semantics of infinite buffer

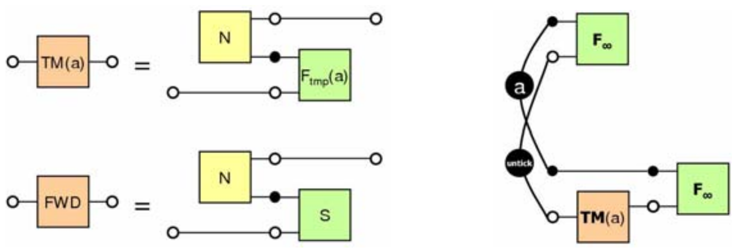


Fig. 15. Some graphical shorthand

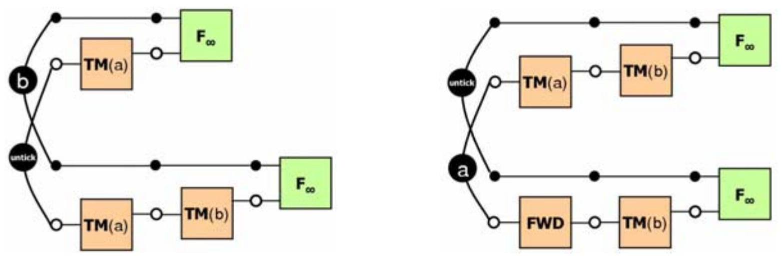


Fig. 16. Datum b arrives (left) and datum a leaves

corresponds to removing the temporary buffer and its nearby node. However, the tile needed to actually do the garbage collection would not satisfy the basic source property, thus we preferr this approach.

To sketch the evolution of infinite buffers, we draw some possible proof steps obtained by horizontal composition of basic tiles. To simplify the graphical notation we introduce some suitable graphical shorthand in Fig. 15 (left) for the composition of a node and a temporary buffer (TM) and for the composition of a node and a synchronous channel (FWD) that basically behaves as a forwarder. Using the shorthand, the tile for inserting a new datum in the infinite buffer can be drawn as in Fig. 15 (right). Figure 16 shows what happens if a new datum b arrives when the buffer already contains a datum a (left) and what happens if a datum is then requested from the buffer (right). Note that it is also allowed for the arrival and departure of data happen at the same time (see Fig. 17).

Proposition 1 (a reconfiguration congruence). *Tile bisimilarity is a congruence for the 2-color semantics of Reo connectors including the infinite FIFO buffer.*

Observe that in this approach reconfiguration and computation are fully integrated (while in [11] and [10,11] the two aspects are dealt with by separate models). Furthermore, reconfigurable connectors and normal connectors can be used together, since reconfiguration is not visible from the outside. However, our tile model currently cannot express more complex reconfigurations that change the interfaces of connectors. Capturing these reconfiguration in such a way as to allow the congruence of bisimilarity to be proved using the basic source property, requires (1) connectors to agree on when and

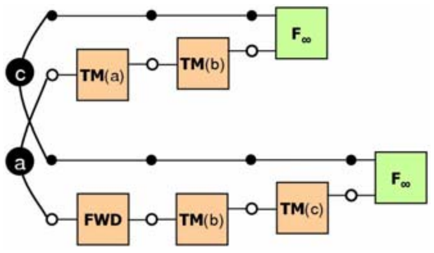


Fig. 17. Datum c arrives while datum a leaves

which reconfiguration to perform, and (2) nodes to propagate this kind of information. We leave an analysis of this approach for future work.

8 Conclusion

We have shown that the Tile Model can be used to describe all main aspects of the semantics of Reo connectors: synchronization, dataflow, context dependency, and reconfiguration. This is the first semantic description of Reo connectors able to present all these aspects natively in a single framework. Furthermore, the semantics is compositional.

As future work we want to consider an alternative approach to the 3-color semantics based on priorities: one can specify that losing data in the LossySync channel has lower priority than data flowing through it. Our goal is to match the expected intuitive semantics of Reo, and solve the problem of causes for data-discard that arises in some cycles in the 3-color semantics, as discussed in [5]. However, further research is necessary to understand how to apply this reasoning to complex connectors. Another long term goal of our work is to understand how to define complex reconfigurations along the lines sketched at the end of Section 7.

References

1. Arbab, F.: Reo: A channel-based coordination model for component composition. Math. Struct. in Comput. Sci. 14(3), 1–38 (2004)
2. Arbab, F., Rutten, J.J.M.M.: A coinductive calculus of component connectors. In: Wirsing, M., Pattinson, D., Hennicker, R. (eds.) WADT 2002. LNCS, vol. 2755, pp. 34–55. Springer, Heidelberg (2003)
3. Baier, C., Sirjani, M., Arbab, F., Rutten, J.J.M.M.: Modeling component connectors in Reo by constraint automata. Sci. Comput. Program 61(2), 75–113 (2006)
4. Bruni, R., Lanese, I., Montanari, U.: A basic algebra of stateless connectors. Theoret. Comput. Sci. 366(1-2), 98–120 (2006)
5. Clarke, D., Costa, D., Arbab, F.: Connector colouring I: Synchronisation and context dependency. Sci. Comput. Program 66(3), 205–225 (2007)
6. Corradini, A., Montanari, U.: An algebraic semantics for structured transition systems and its application to logic programs. Theoret. Comput. Sci. 103, 51–106 (1992)
7. CWI. Reo home page, `http://reo.project.cwi.nl`

8. CWI. A repository of Reo connectors, `http://homepages.cwi.nl/~proenca/webreo/`
9. Gadducci, F., Montanar, U.: The tile model. In: Plotkin, G., Stirling, C., Tofte, M. (eds.) Proof, Language and Interaction: Essays in Honour of Robin Milner, pp. 133–166. MIT Press, Cambridge (2000)
10. Koehler, C., Arbab, F., de Vink, E.: Reconfiguring Distributed Reo Connectors. In: Corradini, A., Montanari, U. (eds.) WADT 2008. LNCS, vol. 5486, pp. 221–235. Springer, Heidelberg (2009)
11. Koehler, C., Costa, D., Proença, J., Arbab, F.: Reconfiguration of Reo connectors triggered by dataflow. In: Ermel, C., Heckel, R., de Lara, J. (eds.) Proceedings of GT-VMT 2008. Elect. Communic. of the European Association of Software Science and Technology, vol. 10, pp. 1–13. EASST (2008)
12. Koehler, C., Lazovik, A., Arbab, F.: Connector rewriting with high-level replacement systems. In: Canal, C., Poizat, P., Viroli, M. (eds.) Proceedings of FOCLASA 2007. Elect. Notes in Th. Comput. Sci. Elsevier Science, Amsterdam (2007)
13. Larsen, K.G., Xinxin, L.: Compositionality through an operational semantics of contexts. In: Paterson, M. (ed.) ICALP 1990. LNCS, vol. 443, pp. 526–539. Springer, Heidelberg (1990)
14. MacLane, S.: Categories for the working mathematician. Springer, Heidelberg (1971)
15. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theoret. Comput. Sci. 96, 73–155 (1992)
16. Montanari, U., Rossi, F.: Graph rewriting, constraint solving and tiles for coordinating distributed systems. Applied Categorical Structures 7(4), 333–370 (1999)
17. Plotkin, G.D.: A structural approach to operational semantics. J. Log. Algebr. Program. 60-61, 17–139 (2004)