Encoding Context-Sensitivity in Reo into Non-Context-Sensitive Semantic Models

Sung-Shik T.Q. Jongmans[1,⋆], Christian Krause[2,⋆⋆], Farhad Arbab[1]

[1] Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands
[2] Hasso Plattner Institute (HPI), University of Potsdam, Germany

Abstract. Reo is a coordination language which can be used to model the interactions among a set of components or services in a compositional manner using *connectors*. The language concepts of Reo include synchronization, mutual exclusion, data manipulation, memory and context-dependency. Context-dependency facilitates the precise specification of a connector's possible actions in situations where it would otherwise exhibit nondeterministic behavior. All existing formalizations of context-dependency in Reo are based on extended semantic models that provide constructs for modeling the presence and absence of I/O requests at the ports of a connector.
In this paper, we show that context-dependency in Reo can be encoded in basic semantic models, namely connector coloring with two colors and constraint automata, by introducing additional fictitious ports for Reo's primitives. Both of these models were considered as not expressive enough to handle context-dependency up to now. We demonstrate the usefulness of our approach by incorporating context-dependency into the constraint automata based Vereofy model checker.

1 Introduction

Over the past decades, *coordination languages* have emerged for modeling and implementing interaction protocols between two or more software components. One example is Reo [1], a language for compositional construction of *connectors*. Connectors are software entities that coordinate the communication between components; they constitute the *glue* that holds components together, and become, once considered at a higher level of abstraction, components themselves.

Connectors have several behavioral properties; for instance, they may manipulate data items that pass through them. Another property is *context-dependency* or *context-sensitivity*: whereas the behavior of a context-*in*sensitive connector depends only on its own state, the behavior of a context-sensitive connector depends also on the presence or absence of I/O-requests at its ports—its *context*. To illustrate context-sensitivity, we consider the LossySync connector, which coordinates the interaction between two components: a *writer* and a *taker*. If the

⋆ Corresponding author. E-mail: jongmans@cwi.nl
⋆⋆ Supported by the research school in 'Service-Oriented Systems Engineering' at HPI

taker is prepared to receive data, LossySync properly relays a data item from the writer to the taker. If the taker, however, refuses to receive, LossySync loses the data item sent by the writer. Since LossySync's behavior depends on the taker's willingness to receive data, that is, the presence or absence of a request for input, LossySync exhibits context-dependent behavior.

Several formal models for describing the behavior of Reo connectors exist, but not all of them have constructs for context-dependency. For example, the early models (e.g., an operational model based on *constraint automata* [2]), although attractive because of their simplicity, lack such constructs. These models implement context-sensitivity as non-determinism. In an attempt to mend this deficiency, more recent models incorporate constructs for context-dependency, but at the cost of more complex formalisms (e.g., the *3-coloring model* [3]). As a result, the algorithms for their simulation and verification suffer from a high computational complexity, which makes these models less attractive in practice.

In this contribution, we show that context-dependency in fact *can* be captured in simple semantic models, namely the *2-coloring model* [3] and constraint automata: we define an operator that transforms a connector with 3-coloring semantics to one with 2-coloring semantics, while preserving its context-sensitive behavior. Furthermore, we prove the transformation's correctness, and, to illustrate its merits, we show how our approach enables the verification of context-dependent connectors with the Vereofy model checker (impossible up to now). Other applications of our approach include context-sensitive *connector decomposition* [4], and, as we speculate, an improved implementation of Reo's interpreter.

The paper is organized as follows. In Section 2, we briefly discuss Reo and connector coloring. In Section 3, we present the transformation from 3-coloring models to 2-coloring models. In Section 4, we present an application of our approach to Vereofy. We discuss related work in Section 5. Section 6 concludes the paper.

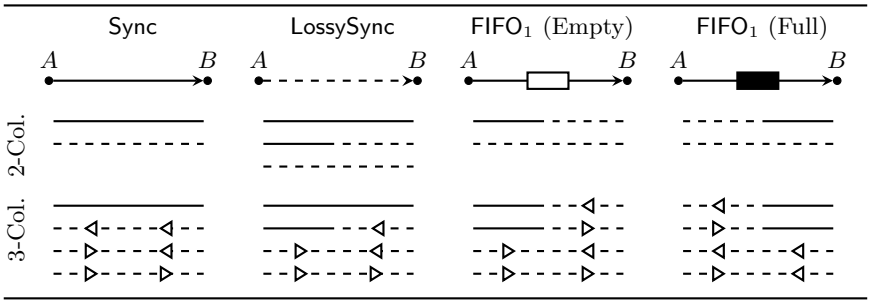
2 Reo Overview

In this section, we discuss connectors in Reo and the coloring models for describing their behavior. A comprehensive overview appears in [1, 5].

The simplest connectors, called *primitives*, consist of a number of input and output *nodes* to which components can connect and at which they can issue *write* and *take* requests for data items. Data items *flow* through a primitive from its input node(s) to its output node(s); if data flow through a node, this node *fires*. The semantics of a primitive specifies its behavior by describing how and when data items flow through the primitive's nodes. To illustrate this, we now present some common primitives and sketch their semantics informally.

The Sync primitive consists of an input node and an output node. Data items flow through this primitive only if these nodes have pending write and take requests. The LossySync primitive behaves similarly, but, as described in Section 1, loses a data item if its input node has a pending write request, while its output node has no pending take request. In contrast to the previous two

Table 1. Common primitives.



memoryless primitives, primitives can have *buffers* to store data items in. Such primitives exhibit different states, while the internal configuration of Sync and LossySync always stays the same. For instance, the FIFO_1 primitive consists of an input node, an output node, and a buffer of size 1. In the EMPTY state, a write request on the input node of FIFO_1 causes a data item to flow into the buffer (i.e., the buffer becomes full), while a take request on its output node remains pending. Conversely, in the FULL state, a write request on its input node remains pending, while a take request on its output node causes a data item to flow from the buffer to the output node (i.e., the buffer becomes empty). The first row of Table 1 depicts the three primitives discussed. In general, we define primitives as follows. Let $\mathcal{N}\mathsf{ode}$ be a denumerable set of nodes.

Definition 1 (Primitive). *A primitive P of arity k is a list $(n_1^{j_1}, \ldots, n_k^{j_k})$ such that $n_i \in \mathcal{N}\mathsf{ode}$, $j_i \in \{\text{“i”}, \text{“o”}\}$, and $[\text{if } i \neq i', \text{ then } n_i \neq n_{i'}]$ for all $1 \leq i, i' \leq k$.*

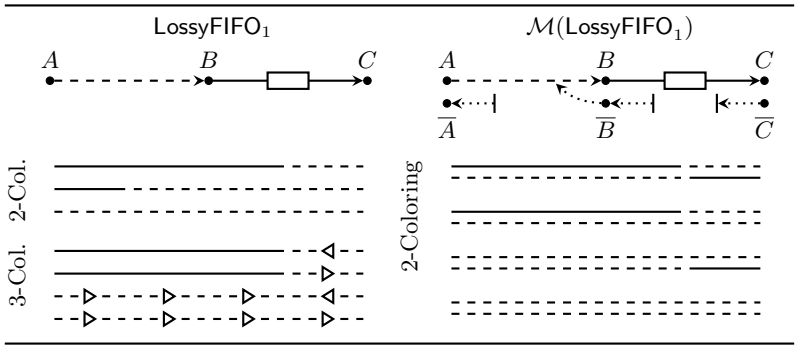
One can construct complex connectors from simpler constituents using *composition*. In this view, a connector consists of a set of nodes, a set of primitives connecting these nodes, and a subset of *boundary nodes* on which components can perform I/O-operations. Although primitives have only boundary nodes, this generally does not hold for composed connectors. For instance, composing LossySync and FIFO_1, by *joining* the former's output node with the latter's input, causes their shared node to become *internal* to the composed connector. This connector, called $\mathsf{LossyFIFO}_1$, appears in the top–left cell of Table 2. We proceed with the formal definitions.

Definition 2 (Connector). *A connector C is a tuple $\langle N, B, E\rangle$ such that N is the set of nodes occurring in C, $\emptyset \neq B \subseteq N$ is a set of boundary nodes, and E is a set of primitives.*

Definition 3 (Composition of connectors). *Let $C_1 = \langle N_1, B_1, E_1\rangle$ and $C_2 = \langle N_2, B_2, E_2\rangle$ be connectors such that $E_1 \cap E_2 = \emptyset$. Their composition, denoted $C_1 \times C_2$, is defined as: $C_1 \times C_2 = \langle N_1 \cup N_2, (B_1 \cup B_2) \setminus (B_1 \cap B_2), E_1 \cup E_2\rangle$.*

Thus, to compose two connectors, we merge their sets of nodes, compute a new set of boundary nodes, and merge the primitives that constitute them.

Table 2. Empty $\mathsf{LossyFIFO}_1$ and its $\mathcal{M}$-transformation.



Thus far, we presented only the structure of connectors; next, we focus on their behavior. More specifically, we discuss *connector coloring* [3], the most relevant model to this paper, in some detail; we mention other models in Section 5. Connector coloring works by assigning *colors* to the nodes of a connector. These colors specify whether data items may flow at a node. For instance, when using two colors, one color expresses that data can flow at a node (i.e., the flow-color: ——), while the other expresses the opposite (i.e., the no-flow color: - - - -). We call a total map from the nodes of a connector to colors a *coloring*.

Definition 4 (Coloring [3]). *Let $N \subseteq \mathcal{N}\mathsf{ode}$ and $\mathcal{C}\mathsf{olors}$ a set of colors. A coloring over N, denoted c, is a total map $N \to \mathcal{C}\mathsf{olors}$. We denote c's domain by $\mathsf{dom}(c)$.*

To model a connector's different behavior in different states, we use *coloring tables*. A coloring table consists of a number of colorings and corresponds to a configuration of a connector; each coloring describes one way in which nodes can fire synchronously in this configuration.

Definition 5 (Coloring table [3]). *A coloring table, denoted T, is a set of colorings with mutually equal domains, denoted $\mathsf{dom}(T)$, and co-domains.*

When certain nodes fire synchronously, a connector's configuration may change (e.g., a full FIFO_1 can become empty). We use *next functions*, which describe transitions from one coloring table to the next, to model this change.

Definition 6 (Next function [5]). *Let S be a set of coloring tables such that $\mathsf{dom}(T_1) = \mathsf{dom}(T_2)$ for all $T_1, T_2 \in S$. A next function over S, denoted η, is a map $S \times \{\mathsf{dom}(S) \to \mathcal{C}\mathsf{olors}\} \to S$ in which $\mathsf{dom}(S) = [\mathsf{dom}(T)$ for any $T \in S]$ is the domain of any coloring in $\bigcup_{T \in S} T$.*

Coloring tables that consist of 2-colorings for the previously discussed primitives appear in the third row of Table 1. For instance, the top coloring of Sync denotes the presence of flow between A and B; its bottom coloring denotes the absence of flow. The middle coloring of $\mathsf{LossySync}$ denotes that data items flow only at A, causing them to get lost before reaching B.

To compute the behavior of a composed connector whose constituents have coloring tables and next functions as semantic model, we use the composition operators for coloring tables and next functions. The formal definitions appear below; shortly, we discuss an example (LossySync).

Definition 7 (Composition of colorings [3]). *Let c_1 and c_2 be colorings such that $c_1(n) = c_2(n)$ for all $n \in \mathsf{dom}(c_1) \cap \mathsf{dom}(c_2)$. Their composition, denoted $c_1 \cup c_2$, is defined as:*

$$c_1 \cup c_2 = \left\{ n \mapsto \kappa \;\middle|\; n \in \mathsf{dom}(c_1) \cup \mathsf{dom}(c_2) \;\; \textit{and} \;\; \kappa = \begin{pmatrix} c_1(n) \;\textit{if}\; n \in \mathsf{dom}(c_1) \\ c_2(n) \;\textit{otherwise} \end{pmatrix} \right\}$$

Definition 8 (Composition of coloring tables [3]). *Let T_1 and T_2 be coloring tables. Their composition, denoted $T_1 \cdot T_2$, is defined as:*

$$T_1 \cdot T_2 = \left\{ c_1 \cup c_2 \;\middle|\; \begin{array}{l} c_1 \in T_1 \;\; \textit{and} \;\; c_2 \in T_2 \;\; \textit{and} \\ c_1(n) = c_2(n) \;\; \textit{for all} \;\; n \in \mathsf{dom}(c_1) \cap \mathsf{dom}(c_2) \end{array} \right\}$$

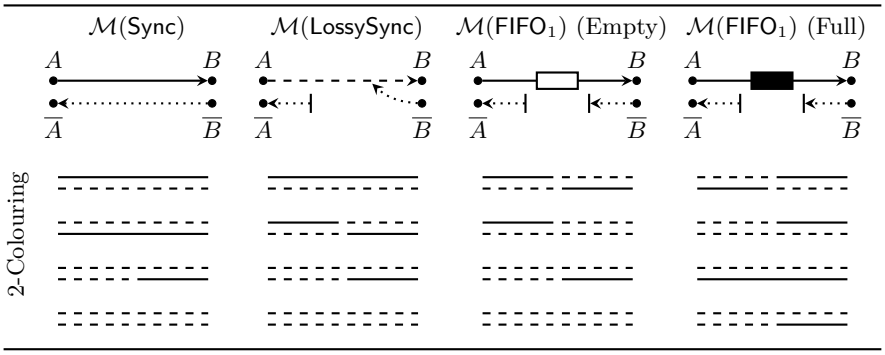
Definition 9 (Composition of next functions [5]). *Let η_1 and η_2 be next functions over sets of coloring tables S_1 and S_2, respectively, and let $S_1 * S_2 = \{\, T_1 \cdot T_2 \mid T_1 \in S_1 \;\; \textit{and} \;\; T_2 \in S_2 \,\}$. Their composition, denoted $\eta_1 \otimes \eta_2$, is defined as:*

$$\eta_1 \otimes \eta_2 = \left\{ (T_1 \cdot T_2, c_1 \cup c_2) \mapsto \eta_1(T_1) \cdot \eta_2(T_2) \;\middle|\; \begin{array}{l} T_1 \cdot T_2 \in S_1 * S_2 \\ \textit{and} \;\; c_1 \cup c_2 \in T_1 \cdot T_2 \end{array} \right\}$$

The expressiveness of connector coloring depends on the instantiation of $\mathcal{C}\mathsf{olors}$ in Definitions 4, 5, and 6. With two colors, we obtain *2-coloring models* in which $\mathcal{C}\mathsf{olors} = \{$ ——— , - - - - $\}$. Whereas 2-coloring models can express synchronization, they cannot express context-dependency: to model context-sensitive connectors, three colors seem necessary. With three colors, we obtain *3-coloring models* in which $\mathcal{C}\mathsf{olors} = \{$ ——— , - -▷- - , - -◁- - $\}$. Instead of one no-flow color as in 2-coloring models, two colors to express the absence of flow exist in 3-coloring models. As a result, in 3-coloring models, one can express *why* data does not flow, whereas in 2-coloring models, one can express only *that* data does not flow. More precisely, in 3-coloring models, the direction of the arrow of the no-flow colors indicates where the reason for the absence of flow comes from. Loosely speaking, an arrow pointing in the same direction as the flow indicates that a node has no pending write requests, while an arrow pointing in the opposite direction indicates that a node has no pending take requests. In text, we associate - -▷- - with the former case and - -◁- - with the latter. We prefix "coloring" by "2-" (respectively, "3-") if $\mathcal{C}\mathsf{olors}$ in Definitions 4, 5, and 6 accords with 2-coloring (respectively, 3-coloring) models.

To illustrate the previous, 3-colorings for Sync, LossySync and FIFO_1 appear in the fourth row of Table 1, and composed 2-coloring and 3-coloring tables for $\mathsf{LossyFIFO}_1$ appear in the two bottom–left cells of Table 2. The middle coloring in the 2-coloring table of the empty $\mathsf{LossyFIFO}_1$ describes an inadmissible behavior: if A fires, but B does not, LossySync loses a data item between

Table 3. $\mathcal{M}$-transformation of common primitives.



A and B despite the empty buffer. Such a coloring does not exist in the 3-coloring table of the empty $\mathsf{LossyFIFO}_1$. Thus, 3-coloring models can capture context-dependency—through the propagation of the reason for the absence of flow—whereas 2-coloring models cannot.

Finally, we define *colored connectors* (respectively, 2-colored, 3-colored connectors), which are connectors whose semantics are defined in terms of a coloring model (respectively, 2-coloring model, 3-coloring model), and their composition operator, which preserves well-formedness by Proposition 3.3.5 in [5].

Definition 10 (Colored connectors). *A colored connector over a set of coloring tables S, denoted $\mathcal{C}^{\mathsf{Col}}$, is a tuple $\langle C, \eta \rangle$ in which $C = \langle N, B, E \rangle$ is a connector, and η is a next function over S such that $\mathsf{dom}(S) = N$.*

Definition 11 (Composition of colored connectors). *Let $\mathcal{C}_1^{\mathsf{Col}} = \langle C_1, \eta_1 \rangle$ and $\mathcal{C}_2^{\mathsf{Col}} = \langle C_2, \eta_2 \rangle$ be colored connectors. Their composition, denoted $\mathcal{C}_1^{\mathsf{Col}} \times \mathcal{C}_2^{\mathsf{Col}}$, is defined as: $\mathcal{C}_1^{\mathsf{Col}} \times \mathcal{C}_2^{\mathsf{Col}} = \langle C_1 \times C_2, \eta_1 \otimes \eta_2 \rangle$.*

3 From Three to Two Colors

In the literature, 2-coloring models are considered not expressive enough to capture context-dependency of connectors. In this section, however, we show the converse: at the expense of making the models of the primitives more complex, we encode context-dependent behavior using only two colors (and without altering the existing composition operators for coloring models). Our encoding comprises a generic transformation from 3-colored connectors to 2-colored connectors. Essentially, we trade a more complex semantic model—i.e., 3-coloring—with simple primitives for a simpler semantic model—i.e., 2-coloring—with more complex primitives. We start by introducing our transformation operator, denoted $\mathcal{M}$, which we liberally overload for different types of arguments for notational convenience. In Section 3.1, we prove the correctness of the transformation by showing that flow through nodes of a 3-colored connector $\mathcal{C}^{\mathsf{Col}}$ implies corresponding flow

through its transformation $\mathcal{M}(\mathcal{C}^{\mathsf{Col}})$ (a 2-colored connector); in Section 3.2, we discuss the distributivity properties—important for compositionality—of $\mathcal{M}$.

We begin with the $\mathcal{M}$-transformation for connectors. Informally, this transformation clones all nodes in a connector and inverts the direction of the flow through these clones. The latter facilitates the backwards propagation of the reason for the absence of flow in case the connector lacks appropriate take requests (in a similar spirit as the - -◁- - color). Henceforth, we call a node n of the original connector a *base node* and its unique clone, denoted $\overline{n}$, a *context node*. Base and context nodes correspond *one-to-one*, and we consider them each other's *duals*. Next, let N be a set of base nodes. We define its $\mathcal{M}$-transformation, denoted $\mathcal{M}(N)$, as $\mathcal{M}(N) = \bigcup_{n \in N} \{n, \overline{n}\}$, that is, the set of base nodes and their duals. Finally, let *inv* be the inverse map of "i" and "o", that is, $inv = \{\text{"i"} \mapsto \text{"o"}, \text{"o"} \mapsto \text{"i"}\}$. We can now define $\mathcal{M}$ for connectors, starting with a definition of $\mathcal{M}$ for primitives.

Definition 12 ($\mathcal{M}$-transformation of primitives). *Let $P = (n_1^{j_1}, \ldots, n_k^{j_k})$ be a primitive. Its $\mathcal{M}$-transformation, denoted $\mathcal{M}(P)$, is defined as: $\mathcal{M}(P) = \left(n_1^{j_1}, \ldots, n_k^{j_k}, \overline{n}_1^{inv(j_1)}, \ldots, \overline{n}_k^{inv(j_k)}\right)$.*

Definition 13 ($\mathcal{M}$-transformation of connectors). *Let $C = \langle N, B, E \rangle$ be a connector. Its $\mathcal{M}$-transformation, denoted $\mathcal{M}(C)$, is defined as: $\mathcal{M}(C) = \langle \mathcal{M}(N), \mathcal{M}(B), \mathcal{M}(E) \rangle$ in which $\mathcal{M}(E) = \{\ \mathcal{M}(P) \mid P \in E\ \}$.*

One can straightforwardly show that $\mathcal{M}$ for primitives yields primitives, that is, preserves well-formedness with respect to Definition 1 [6]. The same holds for $\mathcal{M}$ for connectors (the proof uses preservation of well-formedness by $\mathcal{M}$ for primitives).

Proposition 1 ($\mathcal{M}$-transformation of primitives and connectors preserves well-formedness). *$\mathcal{M}$-transforming a primitive yields a primitive. $\mathcal{M}$-transforming a connector yields a connector.*

The $\mathcal{M}$-transformations of Sync, LossySync, and FIFO_1 appear in the first row of Table 3, while the top–right cell of Table 2 depicts the $\mathcal{M}$-transformation of $\mathsf{LossyFIFO}_1$. The figures exemplify that data flow in the opposite direction through context nodes when compared with the direction of the flow through base nodes. As mentioned before, this resembles how the 3-coloring model communicates the reason for no-flow backwards through the connector. Furthermore, context nodes nowhere communicate with base nodes: they form a *context circuit* that influences the behavior of the *base circuit* and vice versa, but data items cannot flow from one of these circuits to the other. The $\mathcal{M}$-transformation of LossySync exemplifies this influence: the new dotted arrow tangent to the original dashed arrow indicates that data may disappear between A and B iff data flow through $\overline{B}$.

To describe the behavior of $\mathcal{M}$-transformed connectors, we proceed with the definition of $\mathcal{M}$ for colorings, coloring tables, and next functions. We first present their formal definitions, and clarify these afterwards.

Definition 14 ($\mathcal{M}$-transformation of colorings). *Let c be a 3-coloring. Its $\mathcal{M}$-transformation, denoted $\mathcal{M}(c)$, is defined as:*

$$\mathcal{M}(c) = \bigcup_{n \in \mathsf{dom}(c)} \begin{cases} \{\, n \mapsto \text{- - - -}, \overline{n} \mapsto \text{- - - -} \,\} & \textit{if } c(n) = \text{- -▷- -} \\ \{\, n \mapsto \text{- - - -}, \overline{n} \mapsto \text{——} \,\} & \textit{if } c(n) = \text{- -◁- -} \\ \{\, n \mapsto \text{——}, \overline{n} \mapsto \text{- - - -} \,\} & \textit{if } c(n) = \text{——} \end{cases}$$

Definition 15 ($\mathcal{M}$-transformation of coloring tables). *Let T be a 3-coloring table. Its $\mathcal{M}$-transformation, denoted $\mathcal{M}(T)$, is defined as: $\mathcal{M}(T) = \{\, \mathcal{M}(c) \mid c \in T \,\}$.*

Definition 16 ($\mathcal{M}$-transformation of next functions). *Let η be a next function over a set of 3-coloring tables S. Its $\mathcal{M}$-transformation, denoted $\mathcal{M}(\eta)$, is defined as: $\mathcal{M}(\eta) = \{\, (\mathcal{M}(T), \mathcal{M}(c)) \mapsto \mathcal{M}(\eta(T, c)) \mid T \in S \text{ and } c \in T \,\}$.*

Informally, $\mathcal{M}$ applied to a 3-coloring c clones its domain (similar to the way $\mathcal{M}$ for connectors clones nodes) and maps each node in the new domain to either —— or - - - -. The idea behind these mappings follows below.

- If c maps n to ——, $\mathcal{M}(c)$ also maps n to ——, while it maps $\overline{n}$ to - - - -. This ensures that data never flow through the same parts of the base and the context circuits synchronously. If we would allow such synchronous flow, for instance, data items could flow between the base nodes and through the context circuit of a LossySync (i.e., this LossySync has pending write and take requests) at the same time. This would mean, however, that this LossySync may lose the data item flowing through its base circuit *without reason* (because of the pending take request). This is inadmissible behavior.
- If c maps n to - -◁- - (i.e., the no-flow color indicating that n lacks take requests), $\mathcal{M}(c)$ maps n to - - - - (because flow cannot appear out of nowhere), while it maps $\overline{n}$ to —— (because the absence of pending take requests may cause lossy channels to lose data items).
- If c maps n to - -▷- - (i.e., the no-flow color indicating that n lacks write requests), $\mathcal{M}(c)$ maps n to - - - - (because flow cannot appear out of nowhere), and the same holds for $\overline{n}$ (because the absence of pending write requests may never cause loss of data).

Next, we discuss preservation of well-formedness [6]. Let c be a 3-coloring. We make two observations: (i) because context nodes correspond one-to-one to base nodes, $\mathcal{M}(c)$ maps all nodes in $\mathcal{M}(\mathsf{dom}(c))$ exactly once, and (ii) $\mathcal{M}(c)$ maps all nodes in its domain to either —— or - - - -. Hence, $\mathcal{M}(c)$ defines a 2-coloring over the set $\mathcal{M}(\mathsf{dom}(c))$. Well-formedness of $\mathcal{M}$ for 3-coloring tables then follows immediately. Finally, we argue that $\mathcal{M}$ for next functions preserves well-formedness; let η be a next function over a set of 3-coloring tables S. Since $\mathcal{M}$ for 3-colorings (respectively, 3-coloring tables) yields well-formed 2-colorings (respectively, 2-coloring tables), and since S is a set of 3-coloring tables, $\mathcal{M}(\eta)$ defines a map from [2-coloring tables and 2-colorings] to 2-coloring tables. Hence, $\mathcal{M}(\eta)$ defines a next function over a set of 2-coloring tables.

Proposition 2 ($\mathcal{M}$-transformation of colorings, coloring tables, and next functions preserves well-formedness). *$\mathcal{M}$-transforming a 3-coloring c yields a 2-coloring over $\mathcal{M}(\mathsf{dom}(c))$. $\mathcal{M}$-transforming a 3-coloring table yields a 2-coloring table. $\mathcal{M}$-transforming a next function over a set of 3-coloring tables S yields a next function over a set of 2-coloring tables $\mathcal{M}(S) = \{\ \mathcal{M}(T) \mid T \in S\ \}$, and $\mathsf{dom}(\mathcal{M}(S)) = \mathsf{dom}(\mathcal{M}(T))$ for any $T \in S$.*

Finally, we present the $\mathcal{M}$-transformation of colored connectors. Both the definition and its preservation of well-formedness turn out straightforwardly. To $\mathcal{M}$-transform a colored connector, we take the $\mathcal{M}$-transformations of its constituents; preservation of well-formedness then follows from Propositions 1 and 2.

Definition 17 ($\mathcal{M}$-transformation of colored connectors). *Let $\mathcal{C}^{\mathsf{Col}} = \langle C, \eta \rangle$ be a colored connector over a set of 3-coloring tables. Its $\mathcal{M}$-transformation, denoted $\mathcal{M}(\mathcal{C}^{\mathsf{Col}})$, is defined as: $\mathcal{M}(\mathcal{C}^{\mathsf{Col}}) = \langle \mathcal{M}(C), \mathcal{M}(\eta) \rangle$.*

Proposition 3 ($\mathcal{M}$-transformation of colored connectors preserves well-formedness). *$\mathcal{M}$-transforming a colored connector over a set of 3-colorings yields a colored connector over a set of 2-colorings.*

3.1 Correctness of $\mathcal{M}$

In this subsection, we show the *correctness* of $\mathcal{M}$ for colored connectors. To define "correctness" in this context, we first introduce the concept of *paintings*, which are, essentially, (infinite) executions of a colored connector.

Definition 18 (Painting). *Let $\mathcal{C}^{\mathsf{Col}} = \langle C, \eta \rangle$ be a colored connector over S and $T_0 \in S$ the coloring table corresponding to its initial configuration. A painting of $\mathcal{C}^{\mathsf{Col}}$ is a sequence $[T_0, c_0, T_1, c_1, \ldots]$ such that $c_i \in T_i$, and $T_{i+1} = \eta(T_i, c_i)$ for all $i \geq 0$. The set of all $\mathcal{C}^{\mathsf{Col}}$'s paintings is denoted $\mathsf{Painting}(\mathcal{C}^{\mathsf{Col}})$.*

We call $\mathcal{M}$ for colored connectors correct if, for each painting of $\mathcal{C}^{\mathsf{Col}}$, there exists a *corresponding* painting of $\mathcal{M}(\mathcal{C}^{\mathsf{Col}})$ and vice versa; paintings correspond if, for all indexes, (i) the respective colorings assign flow to the same shared nodes—i.e., nodes that occur in both of the colored connectors—and (ii) the respective coloring tables correspond to the same configuration. We formulate our correctness theorem more formally below; a proof follows shortly.

Theorem 1 (Correctness of $\mathcal{M}$). *Let $\mathcal{C}^{\mathsf{Col}} = \langle C, \eta \rangle$ be a colored connector over a set of 3-coloring tables S and $\mathcal{M}(\mathcal{C}^{\mathsf{Col}}) = \langle \mathcal{M}(C), \mathcal{M}(\eta) \rangle$ a colored connector over a set of 2-coloring tables $\mathcal{M}(S)$ (by Proposition 3). Then:*

I. *if:* $[T_0, c_0, \ldots] \in \mathsf{Painting}(\mathcal{C}^{\mathsf{Col}})$
then: $[\mathcal{M}(T_0), \mathcal{M}(c_0), \ldots] \in \mathsf{Painting}(\mathcal{M}(\mathcal{C}^{\mathsf{Col}}))$ *such that for all* $0 \geq i$:
$\{\ n \mid c_i(n) = \text{——}\ \} = \{\ n \in \mathsf{dom}(c_i) \mid (\mathcal{M}(c_i))(n) = \text{——}\ \}$

II. *if:* $[\mathcal{M}(T_0), \mathcal{M}(c_0), \ldots] \in \mathsf{Painting}(\mathcal{M}(\mathcal{C}^{\mathsf{Col}}))$
then: $[T_0, c_0, \ldots] \in \mathsf{Painting}(\mathcal{C}^{\mathsf{Col}})$ *such that for all* $0 \geq i$:
$\{\ n \mid c_i(n) = \text{——}\ \} = \{\ n \in \mathsf{dom}(c_i) \mid (\mathcal{M}(c_i))(n) = \text{——}\ \}$

Later, we sketch a proof by induction that establishes the theorem. For the sake of conciseness, however, we first move large parts of the inductive step to the following two lemmas. Lemma 1 states that $\mathcal{M}$ for next functions over 3-coloring tables preserves the flow behavior of the connector. That is, if an untransformed coloring assigns flow to some base node, the $\mathcal{M}$-transformed coloring (i) exists, and (ii) also assigns flow to this base node. The same must hold in the opposite direction. Lemma 2 states that $\mathcal{M}$ for next functions preserves transitions from one configuration to the next. Note that these two lemmas correspond to the two conditions for "correspondence" given above.

Lemma 1 ($\mathcal{M}$ for colored next functions preserves flow). *Let η be a next function over a set of 3-coloring tables S, let $\mathcal{M}(\eta)$ be its $\mathcal{M}$-transformation, that is, a next function over a set of 2-coloring tables $\mathcal{M}(S)$ (by Proposition 2), and let $n \in \mathsf{dom}(S)$ be a node. Then:*

$$\begin{pmatrix} T \in S \text{ and } c \in T \\ \text{and } c(n) = \text{——} \end{pmatrix} \text{ iff } \begin{pmatrix} \mathcal{M}(T) \in \mathcal{M}(S) \text{ and } \mathcal{M}(c) \in \mathcal{M}(T) \\ \text{and } (\mathcal{M}(c))(n) = \text{——} \end{pmatrix}$$

*Proof. We first prove the left-to-right direction (*ONLY IF*), and proceed with the right-to-left direction (*IF*).*

ONLY IF — *We start by deriving the first two conjuncts of the right-hand side (RHS) from the first two conjuncts of the left-hand side (LHS). This turns out straightforwardly: $T \in S$ implies $\mathcal{M}(T) \in \mathcal{M}(S)$ by the definition of $\mathcal{M}(S)$ in Proposition 2, and $c \in T$ implies $\mathcal{M}(c) \in \mathcal{M}(T)$ by Definition 15 of $\mathcal{M}$ for 3-coloring tables. Finally, we derive the RHS's third conjunct from the third conjunct of the LHS. By the premise, $c(n) =$ ——. Then, by Definition 14 of $\mathcal{M}$ for 3-colorings, $\{ n \mapsto$ ——, $\overline{n} \mapsto$ - - - - $\} \subseteq \mathcal{M}(c)$. Hence, $(\mathcal{M}(c))(n) =$ ——.*

IF — *The first two conjuncts of the LHS follow from the first two conjuncts of the RHS similar to the* ONLY IF *case. Next, by the premise, $(\mathcal{M}(c))(n) =$ ——, that is, $n \mapsto$ —— $\in \mathcal{M}(c)$. By Definition 14 of $\mathcal{M}$ for 3-colorings, this happens only if $c(n) =$ ——.* □

Lemma 2 ($\mathcal{M}$ for colored next functions preserves transitions). *Let η be a next function over a set of 3-coloring tables S, let $\mathcal{M}(\eta)$ be its $\mathcal{M}$-transformation, that is, a next function over a set of 2-coloring tables $\mathcal{M}(S)$ (by Proposition 2), and let $n \in \mathsf{dom}(S)$ be a node. Then:*

$$\begin{pmatrix} T, T' \in S \text{ and } c \in T \\ \text{and } \eta(T, c) = T' \end{pmatrix} \text{ iff } \begin{pmatrix} \mathcal{M}(T), \mathcal{M}(T') \in \mathcal{M}(S) \text{ and } \mathcal{M}(c) \in \mathcal{M}(T) \\ \text{and } (\mathcal{M}(\eta))(\mathcal{M}(T), \mathcal{M}(c)) = \mathcal{M}(T') \end{pmatrix}$$

Proof. The implication, in both directions, follows from the definition of $\mathcal{M}(S)$ in Proposition 2 (first conjunct), Definition 15 of $\mathcal{M}$ for 3-coloring tables (second conjunct), and Definition 16 of $\mathcal{M}$ for next functions (third conjunct). □

Finally, given the previous two lemmas, we sketch a proof of Theorem 1.

Proof (Of Theorem 1; Sketch). Both I. and II. follow from induction on the length of a painting's prefix. The base case (prefix of length 1) follows from preservation of well-formedness of $\mathcal{M}$ for next functions (recall $\mathcal{M}(S) = \{ \mathcal{M}(T) \mid T \in S \}$), and because $T_0 \in S$ by Definition 18. To prove the inductive step, first, suppose there exists a painting with prefix of length $2j-1$ on which the theorem holds, for some $j \geq 1$ (note that the $(2j-1)$-th element is a coloring table). Next, apply Lemma 1 to establish that there exists a painting with a prefix of length $2j$ on which the theorem holds (note that the $(2j)$-th element is a coloring). Finally, apply Lemma 2 to establish that there exists a painting with a prefix of length $2j+1 = 2(j+1)-1$ on which the theorem holds. □

3.2 Distributivity of $\mathcal{M}$

Previously, we showed that by applying $\mathcal{M}$ to a 3-colored connector, we obtain a corresponding 2-colored connector. Though an essential result, it not yet suffices: to properly construct a complex 2-colored connector from context-dependent constituents, we still must compose a corresponding 3-colored connector from 3-colored primitives first. Only thereafter, we can apply $\mathcal{M}$ to obtain the desired 2-colored connector. Instead, we would prefer (i) to apply $\mathcal{M}$ only once to the 3-colored primitives (yielding, among others, the primitives in Table 3), and (ii) to construct context-dependent 2-colored connectors by composing these $\mathcal{M}$-transformed primitives. We prefer this approach, because we speculate that an implementation of Reo that operates on 2-coloring models can compute connector composition more efficiently than an implementation that operates on 3-coloring models. In this section, we develop the theory that accommodates this: we show the *compositionality* of $\mathcal{M}$. This means that it does not matter whether we (a) first apply $\mathcal{M}$ to 3-colored connectors and then the composition operator on the resulting 2-colored connectors, or (b) first apply the composition operator on 3-colored connectors and then $\mathcal{M}$ to the resulting composition. Specifically, we show that $\mathcal{M}$ distributes over composition of connectors (Definition 2) and composition of next functions (Definition 9). Distributivity over composition of colored connectors (Definition 10) then follows straightforwardly.

We start, however, with a proposition stating that $\mathcal{M}$ for sets of nodes (defined in the second paragraph of Section 3) distributes over the set operators $\cup$, $\cap$, and $\setminus$. Our complete proof [6] consists of a series of straightforward applications of the definitions and the distributivity laws of these operators, while making use of the one-to-one correspondence between base and context nodes.

Proposition 4 ($\mathcal{M}$ for sets of nodes distributes over $\cup, \cap, \setminus$ for sets). *Let $N_1, N_2 \subseteq \mathcal{N}\mathsf{ode}$ be sets of nodes. Then: $\mathcal{M}(N_1) \cup \mathcal{M}(N_2) = \mathcal{M}(N_1 \cup N_2)$, $\mathcal{M}(N_1) \cap \mathcal{M}(N_2) = \mathcal{M}(N_1 \cap N_2)$, and $\mathcal{M}(N_1) \setminus \mathcal{M}(N_2) = \mathcal{M}(N_1 \setminus N_2)$.*

We proceed with a compositionality lemma that concerns $\mathcal{M}$ for connectors.

Lemma 3 ($\mathcal{M}$ for connectors distributes over $\times$ for connectors). *Let C_1 and C_2 be connectors. Then: $\mathcal{M}(C_1) \times \mathcal{M}(C_2) = \mathcal{M}(C_1 \times C_2)$.*

Proof. Suppose $C_1 = \langle N_1, B_1, E_1 \rangle$ *and* $C_2 = \langle N_2, B_2, E_2 \rangle$ *(without loss of generality). Applying Definition 13 of* $\mathcal{M}$ *for connectors and Definition 2 of* $\times$ *to rewrite the above equation, we obtain the following:*

$$\left\langle \begin{array}{l} \mathcal{M}(N_1) \cup \mathcal{M}(N_2), \\ \mathcal{M}(B_1) \cup \mathcal{M}(B_2) \setminus \mathcal{M}(B_1) \cap \mathcal{M}(B_2), \\ \mathcal{M}(E_1) \cup \mathcal{M}(E_2) \end{array} \right\rangle = \left\langle \begin{array}{ll} \mathcal{M}(N_1 \cup N_2), & \text{(I)} \\ \mathcal{M}(B_1 \cup B_2 \setminus B_1 \cap B_2), & \text{(II)} \\ \mathcal{M}(E_1 \cup E_2) & \text{(III)} \end{array} \right\rangle$$

Sub-equations (I) *and* (II) *follow from Proposition 4. Sub-equation* (III) *holds because, by Definition 13 of* $\mathcal{M}$ *for sets of primitives:* $\mathcal{M}(E_1) \cup \mathcal{M}(E_2) = \{\ \mathcal{M}(P) \mid P \in E_1\ \} \cup \{\ \mathcal{M}(P) \mid P \in E_2\ \} = \{\ \mathcal{M}(P) \mid P \in E_1 \cup E_2\ \} = \mathcal{M}(E_1 \cup E_2) = \mathcal{M}(E_1 \cup E_2)$. □

To show that $\mathcal{M}$ distributes over composition of next functions, we, as before, start with a proposition. More specifically, Proposition 5 states that $\mathcal{M}$ distributes over composition of colorings and coloring tables. We consider our complete proofs [6], though rather technical and detailed, straightforward. They rely on the following observations: (i) context nodes correspond one-to-one to base nodes, (ii) the colors assigned to a base node and its dual context node by an $\mathcal{M}$-transformed 2-coloring uniquely define the color assigned to the base node by the 3-coloring (by Definition 14 of $\mathcal{M}$ for 3-colorings), and (iii) each context node that corresponds to a base node in the domain-intersection of two untransformed 3-colorings occurs in the domain-intersection of their $\mathcal{M}$-transformations.

Proposition 5 ($\mathcal{M}$ **for colorings and coloring tables distributes over** $\cup$ **for colorings and** $\cdot$ **for coloring tables).** *Let* c_1 *and* c_2 *be 3-colorings. Then,* $\mathcal{M}(c_1) \cup \mathcal{M}(c_2) = \mathcal{M}(c_1 \cup c_2)$. *Let* T_1 *and* T_2 *be 3-coloring tables. Then,* $\mathcal{M}(T_1) \cdot \mathcal{M}(T_2) = \mathcal{M}(T_1 \cdot T_2)$.

We proceed with a compositionality lemma that concerns $\mathcal{M}$ for next functions.

Lemma 4 ($\mathcal{M}$ **for next functions distributes over** $\otimes$ **for next functions).** *Let* η_1 *and* η_2 *be next functions over sets of 3-coloring tables* S_1 *and* S_2. *Then:* $\mathcal{M}(\eta_1) \otimes \mathcal{M}(\eta_2) = \mathcal{M}(\eta_1 \otimes \eta_2)$.

Proof. Follows from Table 4. □

Finally, we present the compositionality theorem of $\mathcal{M}$, which states that $\mathcal{M}$ distributes over composition of colored connectors. As mentioned before, this result follows straightforwardly from the previous lemmas.

Theorem 2 (Compositionality of $\mathcal{M}$**).** *Let* $\mathcal{C}_1^{\mathsf{Col}}$ *and* $\mathcal{C}_2^{\mathsf{Col}}$ *be colored connectors over sets of 3-coloring tables. Then:* $\mathcal{M}(\mathcal{C}_1^{\mathsf{Col}}) \times \mathcal{M}(\mathcal{C}_2^{\mathsf{Col}}) = \mathcal{M}(\mathcal{C}_1^{\mathsf{Col}} \times \mathcal{C}_2^{\mathsf{Col}})$.

Proof. Suppose $\mathcal{C}_1^{\mathsf{Col}} = \langle C_1, \eta_1 \rangle$ *and* $\mathcal{C}_2^{\mathsf{Col}} = \langle C_2, \eta_2 \rangle$ *(without loss of generality). Applying Definition 17 of* $\mathcal{M}$ *for 3-colored connectors and Definition 11 of* $\times$ *to rewrite the above equation, we obtain the following:*

$$\left\langle \begin{array}{l} \mathcal{M}(C_1) \times \mathcal{M}(C_2), \\ \mathcal{M}(\eta_1) \otimes \mathcal{M}(\eta_2) \end{array} \right\rangle = \left\langle \begin{array}{ll} \mathcal{M}(C_1 \times C_2), & \text{(I)} \\ \mathcal{M}(\eta_1 \otimes \eta_2) & \text{(II)} \end{array} \right\rangle$$

Sub-equation (I) *follows immediately from Lemma 3, while sub-equation* (II) *follows from Lemma 4.* □

Table 4. Proof: $\mathcal{M}(\eta_1) \otimes \mathcal{M}(\eta_2) = \mathcal{M}(\eta_1 \otimes \eta_2)$.

$$
\begin{array}{ll}
 & \mathcal{M}(\eta_1) \otimes \mathcal{M}(\eta_2) \\
= & /^* \text{ By Definition 9 of } \otimes \ */ \\
 & \left\{ \begin{array}{c} \langle \mathcal{M}(T_1) \cdot \mathcal{M}(T_2), \mathcal{M}(c_1) \cup \mathcal{M}(c_2) \rangle \\ \downarrow \\ (\mathcal{M}(\eta_1))(T_1, c_1) \cdot (\mathcal{M}(\eta_2))(T_2, c_2) \end{array} \middle| \begin{array}{c} \mathcal{M}(T_1) \cdot \mathcal{M}(T_2) \in \mathcal{M}(S_1) \cdot \mathcal{M}(S_2) \\ \text{and} \\ \mathcal{M}(c_1) \cup \mathcal{M}(c_2) \in \mathcal{M}(T_1) \cdot \mathcal{M}(T_2) \end{array} \right\} \\
= & /^* \text{ By the distributivity of } \mathcal{M} \text{ over } \cup \text{ and } \cdot \text{ in Proposition 5 } \ */ \\
 & \left\{ \begin{array}{c} \langle \mathcal{M}(T_1 \cdot T_2), \mathcal{M}(c_1 \cup c_2) \rangle \\ \downarrow \\ (\mathcal{M}(\eta_1))(T_1, c_1) \cdot (\mathcal{M}(\eta_2))(T_2, c_2) \end{array} \middle| \begin{array}{c} \mathcal{M}(T_1 \cdot T_2) \in \mathcal{M}(S_1 \cdot S_2) \\ \text{and} \\ \mathcal{M}(c_1 \cup c_2) \in \mathcal{M}(T_1 \cdot T_2) \end{array} \right\} \\
= & /^* \text{ Because, by the definition of } \mathcal{M}(S) \text{ in Proposition 2, } \mathcal{M}(T) \in \mathcal{M}(S) \text{ iff } T \in S, \\
 & \text{and because, by Definition 15 of } \mathcal{M} \text{ for 3-coloring tables, } \mathcal{M}(c) \in \mathcal{M}(T) \text{ iff } c \in T \ */ \\
 & \left\{ \begin{array}{c} \langle \mathcal{M}(T_1 \cdot T_2), \mathcal{M}(c_1 \cup c_2) \rangle \\ \downarrow \\ (\mathcal{M}(\eta_1))(T_1, c_1) \cdot (\mathcal{M}(\eta_2))(T_2, c_2) \end{array} \middle| \begin{array}{c} T_1 \cdot T_2 \in S_1 \cdot S_2 \\ \text{and} \\ c_1 \cup c_2 \in T_1 \cdot T_2 \end{array} \right\} \\
= & /^* \text{ Because, by Definition 16 of } \mathcal{M} \text{ for next functions,} \\
 & (\mathcal{M}(\eta_1))(T_1, c_1) = \mathcal{M}(\eta_1(T_1, c_1)) \text{ and } (\mathcal{M}(\eta_2))(T_2, c_2) = \mathcal{M}(\eta_1(T_2, c_2)) \ */ \\
 & \left\{ \begin{array}{c} \langle \mathcal{M}(T_1 \cdot T_2), \mathcal{M}(c_1 \cup c_2) \rangle \\ \downarrow \\ \mathcal{M}(\eta_1(T_1, c_1)) \cdot \mathcal{M}(\eta_2(T_2, c_2)) \end{array} \middle| \ T_1 \cdot T_2 \in S_1 \cdot S_2 \text{ and } c_1 \cup c_2 \in T_1 \cdot T_2 \right\} \\
= & /^* \text{ By the distributivity of } \mathcal{M} \text{ over } \cdot \text{ in Proposition 5 } \ */ \\
 & \left\{ \begin{array}{c} \langle \mathcal{M}(T_1 \cdot T_2), \mathcal{M}(c_1 \cup c_2) \rangle \mapsto \\ \mathcal{M}(\eta_1(T_1, c_1) \cdot \eta_2(T_2, c_2)) \end{array} \middle| \ T_1 \cdot T_2 \in S_1 \cdot S_2 \text{ and } c_1 \cup c_2 \in T_1 \cdot T_2 \right\} \\
= & /^* \text{ By Definition 16 of } \mathcal{M} \text{ for next functions, } \ */ \\
 & \mathcal{M}\left(\left\{ \begin{array}{c} \langle T_1 \cdot T_2, c_1 \cup c_2 \rangle \mapsto \\ \eta_1(T_1, c_1) \cdot \eta_2(T_2, c_2) \end{array} \middle| \ T_1 \cdot T_2 \in S_1 \cdot S_2 \text{ and } c_1 \cup c_2 \in T_1 \cdot T_2 \right\} \right) \\
= & /^* \text{ By Definition 9 of } \otimes \ */ \\
 & \mathcal{M}(\eta_1 \otimes \eta_2)
\end{array}
$$

4 Application: Context-Dependency in Vereofy

As an application, we present an implementation of our encoding in a constraint automata based model checker, which is considered as not expressive enough for the verification of context-dependent connectors. Specifically, we extend the Vereofy [7] model checking tool for the analysis of Reo connectors, developed at the TU of Dresden.[3] Vereofy uses two input languages: the *Reo Scripting Language* (a textual version of Reo) and the guarded command language *CARML* (a textual version of constraint automata). Vereofy allows the verification of temporal properties expressed in LTL and CTL-like logics and supports bisimulation equivalence checks. Moreover, it can generate counterexamples and provides a GUI integration with the Eclipse Coordination Tools (ECT).[4]

[3] Vereofy homepage: `http://www.vereofy.de`
[4] ECT homepage: `http://reo.project.cwi.nl`

```
#include "builtin"
// Non-deterministic LossyFIFO:
CIRCUIT LOSSY_FIFO_ND {
  new LOSSY_SYNC_ND(A;M);
  new FIFO1(M;B);
  M = NULL;
}
```

```
#include "builtin_CD.carml"
// Context-dependent LossyFIFO:
CIRCUIT LOSSY_FIFO_CD {
  new LOSSY_SYNC_CD(A,nM;M,nA);
  new FIFO1_CD(M,nB;B,nM);
  M = NULL; nM = NULL;
}
```

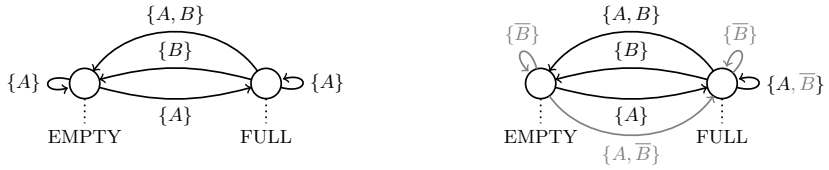


Fig. 1. Non-deterministic (left) vs. context-dependent (right) LossyFIFO$_1$ in Vereofy.

Vereofy operates on constraint automata and, thus, does not natively support context-dependent behavior. However, in the previous section, we showed that we can transform 3-colored connectors to 2-colored connectors, while preserving their context-sensitive semantics. Moreover, 2-coloring models and constraint automata correspond to each other (informal arguments appear in [3, 5], while [6] contains a formal account). Hence, by using the $\mathcal{M}$-transformation, we can construct context-dependent constraint automata as follows. First, we transform the 3-colored primitives to context-dependent 2-colored primitives. Next, we compute the constraint automata corresponding to the resulting 2-colored primitives. Note that the resulting automata can have context-sensitive behavior (because the 2-colored primitives to which they correspond can have such behavior). Finally, we compose the resulting constraint automata to form more complex context-sensitive connectors (possible due to Theorem 2). Although simple and straightforward, this recipe enables the analysis of context-sensitive connectors in Vereofy. For this purpose, we have adapted Vereofy's library of built-in primitives: using the $\mathcal{M}$-transformation, we wrote a new library containing context-dependent versions of the basic Reo primitives.[5]

As an example, Figure 1 depicts a listing of the non-deterministic and the context-dependent versions of the LossyFIFO$_1$ example, and two constraint automata generated from them using Vereofy. For simplicity, we have hidden the internal node M, used a singleton set as data domain, and removed all data constraints in the generated automata. The constraint automata on the left and right correspond to the non-deterministic and the context-dependent versions, respectively. The latter uses our new context-dependent primitives. The crucial difference between the two is that the non-deterministic version contains an illegal transition via port A in the EMPTY state. This corresponds to the connector losing a data item in a situation where the FIFO$_1$ buffer is empty and should, in any case, accept the data item. In the context-sensitive version, however, this illegal transition does not exists. (Note that if we hide all context nodes—i.e.,

[5] CD-Library and examples: `http://reo.project.cwi.nl/vereofy_CD.tar.gz`

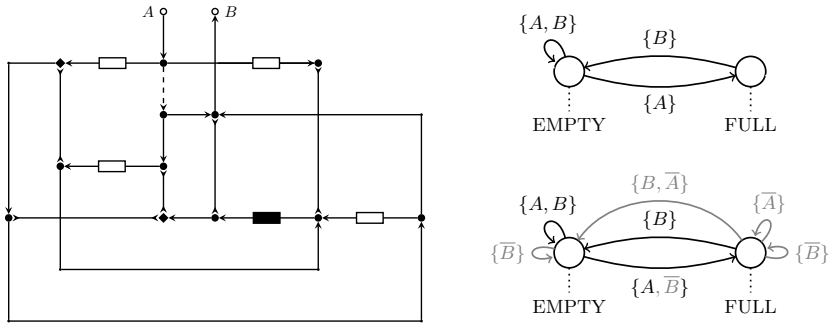


Fig. 2. SyncFIFO$_1$: its composition (left), its ordinary constraint automaton (top–right), and its context-dependent constraint automaton (bottom–right).

disregard all gray transitions in Figure 1—we obtain the non-deterministic automaton without the illegal transition.)

A more complex example concerns SyncFIFO$_1$, a connector with an input node, an output node, and a buffer of size 1. SyncFIFO$_1$ behaves identically to FIFO$_1$, except for the case in which it has an empty buffer and pending I/O-requests on both of its nodes: then, SyncFIFO$_1$ routes a data item from its input node *past its buffer* to its output node in one atomic step (thus behaving as a Sync). Instead of modeling SyncFIFO$_1$ as a primitive without inner structure, we can construct it by composing other primitives as depicted in Figure 2 (left); for reasons of space, we do not discuss the interaction and characteristics of the primitives involved in this composition (more details appear in [6]).

A first attempt to model SyncFIFO$_1$ using our library of context-sensitive primitives failed due to the presence of *causality loops* in the resulting composition.[6] Since one cannot detect and remove causality loops from constraint automata, we removed the colorings that contain causality loops from the composed 3-coloring model of SyncFIFO$_1$ and, afterwards, applied $\mathcal{M}$ to this filtered model. This process yielded a 2-coloring model, whose equivalent constraint automaton we encoded in CARML. In Figure 2, we depict the constraint automaton resulting from the procedure just sketched (bottom–right). Additionally, we depict the constraint automaton that one obtains when composing the ordinary primitives (top–right) instead of the context-sensitive ones. (As before, we hide internal nodes.) At first sight, these automata seem very similar. In fact, if we hide all context nodes in the context-dependent constraint automaton—i.e., disregard its gray transitions—we obtain two identical automata.

The crux of the difference between the two automata, therefore, lies exactly in these context nodes: in contrast to LossyFIFO$_1$, SyncFIFO$_1$ itself exhibits

[6] Causality loops may occur if a composed connector has one or more circular subcircuits (as in the case of SyncFIFO$_1$) and can cause, among other anomalous phenomena, a reason for the absence of flow to appear out of nowhere. Colorings that contain causality loops, therefore, describe inadmissible behavior. In [5], Costa proposes an algorithm for the detection of colorings that contain causality loops.

context-dependent behavior (instead of only the primitives that constitute it, namely LossySync). Recall that in the EMPTY state, if output node B lacks a take request, a write request on A causes a data item to flow into the buffer. However, if B has a pending take request, a write request on A causes a data item to flow immediately to node B. The ordinary constraint automaton of Sync-FIFO_1 does not capture this difference, which means that an implementation of this constraint automaton would non-deterministically choose one of these two options in case of a pending write request on A and a pending take request on B. In contrast, an implementation of the context-dependent constraint automaton of $\mathsf{SyncFIFO}_1$ always chooses the appropriate option, because in the absence of a take requests on B, data items with irrelevant content—i.e., *signals*—flow through $\overline{B}$. To illustrate this, we encourage the interested reader to compose $\mathsf{SyncFIFO}_1$ and FIFO_1 in the same way we composed LossySync and FIFO_1.[7]

5 Related Work

In [8], Arbab et al. introduce a coalgebra-based semantic model—the first—for Reo. Some years later, in [2], Baier et al. present an automaton-based approach, namely constraint automata (CA), and prove correspondences with the coalgebra-based model. In [3], however, Clarke et al. observe that neither of these models can handle context-sensitivity, and they introduce the 2-coloring and 3-coloring models to mend this deficiency. Since then, other semantic models with the same aim have come to existence. In [5], Costa introduces *intentional automata* (IA) as an operational model with constructs for context-dependency. Unlike CA, whose states correspond one-to-one to the internal configurations of connectors, IA have more states than the connectors they model; each state of an IA contains information about not only the configuration of the connector, but also about the nodes that *intend* to fire (i.e., with a pending I/O-request). Similarly, transition-labels consist of two sets of nodes: those that intend to fire, and those that actually fire. By maintaining information about I/O-request on nodes, IA capture context-dependency. The number of states, however, quickly grows large, whereas our approach yields succinct CA. In [9], Bonsangue et al. introduce *guarded automata* (GA) as another automaton-based model for capturing context-dependency. Like CA, the states of GA correspond one-to-one to the configurations of connectors, which makes them significantly more compact than IA. To encode context-sensitivity, every transition-label of a GA consists

[7] The constraint automaton that results from composing the ordinary constraint automata of $\mathsf{SyncFIFO}_1$ and FIFO_1 includes a transition that describes an inadmissible behavior in which, in case both $\mathsf{SyncFIFO}_1$ and FIFO_1 have empty buffers, the input node of $\mathsf{SyncFIFO}_1$ fires and causes its own buffer to become full (while the buffer of FIFO_1 remains empty). The constraint automaton that results from composing the context-dependent constraint automata of $\mathsf{SyncFIFO}_1$ and FIFO_1, in contrast, does not include such a transition: if the input node of $\mathsf{SyncFIFO}_1$ fires when both $\mathsf{SyncFIFO}_1$ and FIFO_1 have empty buffers, the buffer of FIFO_1 becomes full (while the buffer of $\mathsf{SyncFIFO}_1$ remains empty).

of a *guard* and a *string*. Together, they express which nodes can fire (the string), given the presence and absence of requests at certain nodes (the guard). Guarded automata seem very similar to the CA we obtain with our approach: instead of guards that contain negative occurrences of (base) nodes to specify that these nodes have no pending I/O-requests, we make these negative occurrences explicit with the introduction of (flow through) context nodes.

Besides Vereofy, other approaches to model checking Reo connectors exist. In [10], Kokash et al. employ the mCRL2 toolset, developed at the TU of Eindhoven, for model checking connectors, combined with a translation tool that automatically generates mCRL2 specifications from graphical models of Reo connectors. The tool's original algorithm operated on constraint automata, making it impossible to verify context-dependent connectors using this approach. Later, however, Kokash et al. incorporated (3-)coloring information in the tool, thus facilitating verification of context-dependent connectors. This advantage of mCRL2 over Vereofy, which could not handle context-dependent connectors up to now, seems no longer valid as we have shown how to encode context-sensitivity in Vereofy. An advantage of Vereofy over mCRL2, on the other hand, is its ability to generate *counterexamples*, which mCRL2 cannot do. In [11], Kemper introduces a SAT-based approach to model checking *timed constraint automata* (TCA). In her work, Kemper represents TCA as formulas in propositional logic and uses existing SAT solvers for verification. This approach allows for model checking timed properties of Reo connectors, but it cannot handle context-dependency. In [12], Mousavi et al. develop a structural operational semantics in Plotkin's style for Reo, encode this semantics in the Maude term-rewriting language, and use Maude's LTL model checking module to verify Reo connectors. In [13], Khosravi et al. introduce a mapping from Reo to Alloy, a modeling language based on first-order relational logic, and apply the Alloy Analyzer for verification. Although the approach can handle some context-dependent connectors—using a *maximal progress rule* that removes undesired behavior—Khosravi et al. admit to have considerable performance issues.

6 Conclusions and Future Work

We showed how to encode context-sensitivity in the 2-coloring model and constraint automata by adding fictitious nodes to primitives, while both these models are considered incapable of capturing context-dependent behavior. Our approach, constituted by the $\mathcal{M}$-transformation, enables the application of tools and algorithms devised for such simpler semantic models to context-dependent connectors. As an example, we demonstrated how Vereofy can model check context-sensitive connectors, which seemed impossible up to now.

With respect to future work, we would like to investigate whether Reo's implementation can benefit from the results presented in this paper. We speculate that algorithms for the computation of connector composition run faster on $\mathcal{M}$-transformed 2-colored connectors (or their corresponding constraint automata) than on the original 3-colored connectors, because of the simpler semantic model.

Furthermore, we would like to study the relation between other formalisms for Reo that facilitate the proper modeling of context-dependent behavior (e.g., intentional automata and guarded automata).

Acknowledgments We are grateful to the Vereofy team for their support.

References

1. Arbab, F.: Reo: A channel-based coordination model for component composition. Mathematical Structures in Computer Science **14** (2004) 329–366
2. Baier, C., Sirjani, M., Arbab, F., Rutten, J.: Modeling component connectors in Reo by constraint automata. Science of Computer Programming **61**(2) (2006) 75–113
3. Clarke, D., Costa, D., Arbab, F.: Connector colouring I: Synchronisation and context dependency. Science of Computer Programming **66**(3) (2007) 205–225
4. Pourvatan, B., Sirjani, M., Arbab, F., Bonsangue, M.: Decomposition of constraint automata. In: Proceedings of the 7th International Workshop on Formal Aspects of Component Software (FACS 2010). To appear in LNCS. Springer (2011)
5. Costa, D.: Formal Models for Component Connectors. PhD thesis, Vrije Universiteit Amsterdam (2010)
6. Jongmans, S.S., Krause, C., Arbab, F.: Encoding context-sensitivity in Reo into non-context-sensitive semantic models. Technical Report SEN-1105, Centrum Wiskunde & Informatica (2011)
7. Baier, C., Blechmann, T., Klein, J., Klüppelholz, S.: Formal verification for components and connectors. In de Boer, F., Bonsangue, M., Madelaine, E., eds.: Formal Methods for Components and Objects. Volume 5751 of LNCS. Springer (2009) 82–101
8. Arbab, F., Rutten, J.: A coinductive calculus of component connectors. In Wirsing, M., Pattinson, D., Hennicker, R., eds.: Recent Trends in Algebraic Development Techniques. Volume 2755 of LNCS. Springer (2003) 34–55
9. Bonsangue, M.M., Clarke, D., Silva, A.: Automata for context-dependent connectors. In Field, J., Vasconcelos, V., eds.: Coordination Models and Languages. Volume 5521 of LNCS. Springer (2009) 184–203
10. Kokash, N., Krause, C., de Vink, E.P.: Verification of context-dependent channel-based service models. In de Boer, F.S., Bonsangue, M.M., Hallerstede, S., Leuschel, M., eds.: Formal Methods for Components and Objects. Volume 6286 of LNCS. Springer (2010) 21–40
11. Kemper, S.: SAT-based verification for timed component connectors. ENTCS **255** (2009) 103–118
12. Mousavi, M.R., Sirjani, M., Arbab, F.: Formal semantics and analysis of component connectors in Reo. ENTCS **154**(1) (2006) 83–99
13. Khosravi, R., Sirjani, M., Asoudeh, N., Sahebi, S., Iravanchi, H.: Modeling and analysis of Reo connectors using Alloy. In Lea, D., Zavattaro, G., eds.: Coordination Models and Languages. Volume 5052 of LNCS. Springer (2008) 169–183