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

Sung-Shik T.Q. Jongmans¹,⋆, Christian Krause²,⋆⋆, and Farhad Arbab¹

¹ Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands
          jongmans@cwi.nl

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-insensitive 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.
** 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 [15].

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 memoryless primitives, primitives can have buffers to store data items in. Such primitives