Science of Computer Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

A procedure for splitting data-aware processes and its application to coordination ☆

S.-S.T.Q. Jongmans [a,*], D. Clarke [b], J. Proença [b]

[a] Centrum Wiskunde & Informatica, Science Park 123, 1098 XG, Amsterdam, Netherlands
[b] Katholieke Universiteit Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium

HIGHLIGHTS

- We present a procedure for splitting algebraic processes with multiactions and data.
- We prove its correctness (strong bisimilarity between original and split processes).
- We apply it to the process algebraic semantics of the coordination language Reo.
- This application justifies an optimization technique for Reo implementations.

ARTICLE INFO

Article history:
Received 3 February 2013
Received in revised form 31 August 2013
Accepted 13 February 2014
Available online xxxx

Keywords:
Process algebra
Coordination
mCRL2
Reo

ABSTRACT

We present a procedure for splitting processes in a process algebra with multiactions and data (the untimed subset of the specification language mCRL2). This splitting procedure cuts a process into two processes along a set of actions A: roughly, one of these processes contains no actions from A, while the other process contains only actions from A. We state and prove a theorem asserting that the parallel composition of these two processes is provably equal from a set of axioms (sound and complete with respect to strong bisimilarity) to the original process under some appropriate notion of synchronization.
We apply our splitting procedure to the process algebraic semantics of the coordination language Reo: using this procedure and its related theorem, we formally establish the soundness of splitting Reo connectors along the boundaries of their (a)synchronous regions in implementations of Reo. Such splitting can significantly improve the performance of connectors as shown elsewhere.

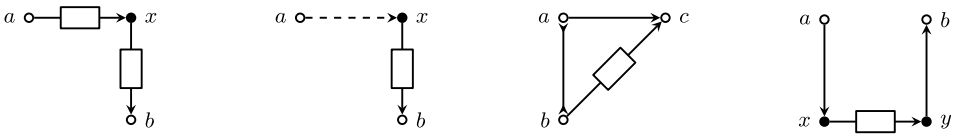
1. Motivation

Context Over the past decades, coordination languages have emerged for the specification and implementation of interaction protocols among entities running concurrently (components, services, threads, etc.). This class of languages includes Reo [2,3], a graphical language for compositional construction of *connectors*: communication media through which entities can interact with each other. Fig. 1 shows some example Reo connectors in their usual graphical syntax. Intuitively, connectors consist of one or more *channels* (i.e., the edges of a connector graph), through which data items flow, and a number of *nodes* (i.e., the vertices of a connector graph), on which channel *ends* (i.e., the endpoints of edges) meet. Through channel

☆ This research is partly funded by the EU project FP7-231620 HATS: Highly Adaptable and Trustworthy Software Using Formal Models (http://www.hats-project.eu/).

* Corresponding author.
E-mail addresses: jongmans@cwi.nl (S.-S.T.Q. Jongmans), dave.clarke@cs.kuleuven.be (D. Clarke), jose.proenca@cs.kuleuven.be (J. Proença).

http://dx.doi.org/10.1016/j.scico.2014.02.017
0167-6423/



(a) FIFO2 (b) LossyFIFO (c) Alternator (d) SyncFIFOSync

Fig. 1. Example connectors.

composition—the act of gluing channels together on nodes—engineers can construct complex connectors. Channels often used include the reliable synchronous channel, called sync, and the reliable asynchronous channel fifo$_n$, which has a buffer of capacity n. Importantly, while nodes have a fixed semantics, Reo features an open-ended set of channels. This allows engineers to define their own channels with custom semantics.

To use connectors in real applications, one must derive executable code from graphical specifications of connectors (e.g., those in Fig. 1). Roughly two implementation approaches currently exist. In the *distributed approach* [15,45,43,44], one implements the behavior of each of the k constituents of a connector and runs these k implementations concurrently as a distributed system; in the *centralized approach* [26,25,30], one computes the behavior of a connector as a whole, implements this behavior, and runs this implementation sequentially as a centralized system. Which of those two approaches to choose may depend on the hardware architecture on which to deploy the application. For example, in the case of a service-oriented choreography application, the distributed approach seems natural, because the services involved run on different machines and the network between them may play a role in their coordination. However, if coordination involves computation threads running on the same machine in some multithreading application, the centralized approach appears more appropriate, because it avoids communication among the constituents of a connector at run-time: in this approach, due to the computation of the behavior of an entire connector at compile time, one abstracts from the individual, smaller, concurrent constituents of a connector to obtain one big sequential program for the whole (which can run in its own dedicated thread at run-time, among the computation threads it coordinates).

One optimization technique applicable to both the distributed and the centralized approaches involves the identification of the *synchronous* and the *asynchronous regions* of a connector [44]. A synchronous region contains exactly those nodes and channels of a connector that synchronize collectively to decide on their individual behavior; an asynchronous region connects synchronous regions in an asynchronous way, typically involving a fifo$_1$ channel. For instance, the connector consisting of a sync channel, a fifo$_1$ channel, and another sync channel (see Fig. 1d) has two synchronous regions, connected by an asynchronous region.

Intuitively, two synchronous regions can run completely independently of each other. Otherwise, by definition, those two subconnectors do not qualify as separate synchronous regions (instead, they constitute the same synchronous region). In the distributed approach, this means that nodes and channels need to share information only with those nodes and channels in the same synchronous region—not with every node or channel in the connector [44]. In the centralized approach, this means that one does not need to compute the behavior of a connector as a whole, but rather on a per-region basis [25]. Supplementary, asynchronous regions connect synchronous regions to each other by transporting data and control information between them. Based on how asynchronous regions do this, one can distinguish different versions of the region-based optimization technique, with different guarantees and for different use cases. For example, an asynchronous region can transport control information *directly* (in which case transportation starts at the same time as the coordination step that triggered it and ends before the next), *atomically* (same as the previous case but transportation can start also after the coordination step that triggered it), or *interleaved* (same as the previous case but transportation does not need to end before the next coordination step). Recent work shows that the region-based optimization technique for Reo can significantly improve performance [15,30,43,44] (both at compile time and at run-time), to the extent that its use will become vital for real-world applications: without it, automatically deploying (including code generation) and running connectors quickly becomes infeasible as their size increases.

Problem The region-based optimization technique still has a serious problem: although we have reason to *believe* (based on intuition and loose informal reasoning) that it preserves the semantics of a connector, we do not *know* this for sure by lack of a formal proof. In fact, in [15], Clarke and Proença identify one implementation of the region-based optimization technique that produces incorrect behavior for a certain class of connectors. An optimization as important as the region-based optimization technique for Reo should have a formal proof of correctness. The problem addressed in this paper is that such a proof currently does not exist.

Contributions of the paper In this paper, using the existing process algebraic semantics of Reo [35,32–34], we prove the correctness of the region-based optimization technique for asynchronous regions with direct transportation.[1] In this semantics, expressed using the specification language mCRL2 [20,22], one associates every connector with a process describing

[1] In practice, an implementation of the direct transportation version requires some form of synchronization between the different sides of an asynchronous region. On shared memory architectures, one can implement such synchronization relatively cheaply. On distributed memory architectures with

Please cite this article in press as: S.-S.T.Q. Jongmans et al., A procedure for splitting data-aware processes and its application to coordination, Science of Computer Programming (2014), http://dx.doi.org/10.1016/j.scico.2014.02.017