Journal of Logical and Algebraic Methods in Programming 84 (2015) 684–707

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

Reversible session-based pi-calculus [☆]

Francesco Tiezzi [a,*], Nobuko Yoshida [b]

[a] University of Camerino, Italy
[b] Imperial College London, UK

ARTICLE INFO

Article history:
Received 31 August 2014
Received in revised form 31 March 2015
Accepted 31 March 2015
Available online 15 April 2015

Keywords:
Reversible computing
The pi-calculus
Session types
Session-based programming

ABSTRACT

In this work, we incorporate reversibility into structured communication-based programming, to allow parties of a session to automatically undo, in a rollback fashion, the effect of previously executed interactions. This permits to take different computation paths along the same session, as well as to revert the whole session and start a new one. Our aim is to define a theoretical basis for examining the interplay in concurrent systems between reversible computation and session-based interaction. We thus propose *ReSπ* a session-based variant of π-calculus using memory devices to keep track of the computation history of sessions in order to reverse it. We show how a session type discipline of π-calculus is extended to *ReSπ*, and illustrate its practical advantages for static verification of safe composition in communication-centric distributed software performing reversible computations. We also show how a fully reversible characterisation of the calculus extends to *committable* sessions, where computation can go forward and backward until the session is committed by means of a specific irreversible action.

1. Introduction

In the field of programming languages, *reversible computing* aims at providing a computational model that, besides the standard forward executions, also permits backward execution steps to undo the effect of previously performed forward computations. Despite being a subject of study for many years, reversible computing is recently experiencing a rise in popularity. This is mainly due to the fact that reversibility is a key ingredient in different application domains. In particular, for what specifically concerns our interest, many researchers have put forward exploiting this paradigm in the design of reliable concurrent systems. In fact, it permits us to understand existing patterns for programming reliable systems (e.g., compensations, checkpointing, transactions) and, possibly, to develop new ones.

A promising line of research on this topic advocates reversible variants of well-established process calculi, such as CCS [2] and π-calculus [3], as formalisms for studying reversibility mechanisms in concurrent systems. By pursing this line of research, in this work we incorporate reversibility into a variant of π-calculus equipped with *session* primitives supporting communication-based programming. A (binary) session consists in a series of reciprocal interactions between two parties, possibly with branching and recursion. Interactions on a session are performed via a dedicated private channel, which is generated when initiating the session. Session primitives come together with a session type discipline offering a simple

[☆] This work is a revised and extended version of [1], presented in the Proceedings of the 7th Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES). The work has been partially sponsored by EPSRC EP/K011715/1, EP/K034413/1, EP/L00058X/1, the COST Action BETTY (IC1201), the EU projects ASCENS (257414) and FP7-612985 UpScale, and the Italian MIUR PRIN project CINA (2010LHT4KM).

[*] Corresponding author.
E-mail addresses: francesco.tiezzi@unicam.it (F. Tiezzi), n.yoshida@imperial.ac.uk (N. Yoshida).

http://dx.doi.org/10.1016/j.jlamp.2015.03.004

checking framework to statically guarantee the correctness of communication patterns. This prevents programs from interacting according to incompatible patterns.

Practically, combining reversibility and sessions paves the way for the development of session-based communication-centric distributed software intrinsically capable of performing reversible computations. In this way, without further coding effort by the application programmer, the interaction among session parties is relaxed so that, e.g., the computation can automatically go back, thus allowing to take different paths when the current one is not satisfactory. As an application example, used in this paper for illustrating our approach, we consider a simple scenario involving a client and multiple providers offering the same service (e.g., on-demand video streaming). The client connects to a provider to request a given service (specifying, e.g., title of a movie, video quality, etc.). The provider replies with a quote determined according to the requested quality of service and to the servers status (current load, available bandwidth, etc.). Then, the client can either accept, negotiate or reject the quote; in the first two cases, the interaction between the two parties shall continue. If a problem occurs during the interaction between the client and the provider for finalising the service agreement, the computation can be automatically reverted. This allows the client to partially undo the current session, in order to take a different computation path along the same session, or even start a new session with (possibly) another provider.

The proposed reversible session-based calculus, called *ReSπ* (*Reversible Session-based π*-calculus), relies on memories to store information about interactions and their effects on the system, which otherwise would be lost during forward computations. This data is used to enable backward computations that revert the effects of the corresponding forward ones. Each memory is devoted to record data concerning a single event, which can correspond to the taking place of a communication action, a choice or a thread forking. Memories are connected with one other, in order to keep track of the computation history, by using unique thread identifiers as links. Like all other formalisms for reversible computing in concurrent settings, forward computations are undone in a *causal-consistent* fashion [4,5]. This means that backtracking does not have to necessarily follow the exact order of forward computations in reverse, because independent actions can be undone in a different order. Thus, an action can be undone only after all the actions causally depending on it have already been undone.

Concerning the session type discipline, *ReSπ* inherits the notion of types and the typing system from π-calculus. Thus, the related results are mainly based on the ones stated for π-calculus. Besides the possibility of taking advantage of the theory already defined for π-calculus, this also allows our investigation to focus on a standard session type setting, rather than on an ad-hoc one specifically introduced for our calculus.

The resulting formalism offers a theoretical basis for examining the interplay between reversible computations and session-based structured interactions. We notice that reversibility enables session parties not only to partially undo the interactions performed along the current session, but also to automatically undo the whole session and restart it, possibly involving different parties. The advantage of the reversible approach is that this behaviour is realised without explicitly implementing loops, but simply relying on the reversibility mechanism available in the language semantics. On the other hand, the session type discipline affects reversibility as it forces concurrent interactions to follow structured communication patterns. If we would consider only a single session, due to linearity, a causal-consistent form of reversibility would not be necessary, i.e. concurrent interactions along the same session are forbidden and, hence, the rollback would follow a single path. Instead, in the general case, concurrent interactions along different sessions may take place, thus introducing causal dependences. In this case, a session execution has to be reverted in a causal-consistent fashion. Notably, interesting issues concerning reversibility and session types are still open questions, especially for what concerns the validity in the reversible setting of standard properties (e.g., progress enforcement) and possibly new properties (e.g., reversibility of ongoing session history, safe closure of subordinate sessions).

It is worth noticing that the proposed calculus is *fully* reversible, i.e. backward computations are always enabled. Full reversibility provides theoretical foundations for studying reversibility in session-based π-calculus, but it is not suitable for a practical use on structured communication-based programming. In fact, reverting a completed session might not be desirable. Therefore, we also propose an extension of the calculus with an *irreversible* action for committing the completion of sessions. In this way, computation would go backward and forward, allowing the parties to try different interactions, until the session is successfully completed and, hence, irreversibly closed.

Summary of the rest of the paper. Section 2 reviews strictly related work. Section 3 recalls syntax and semantics definitions of the considered session-based variants of π-calculus. Section 4 introduces *ReSπ*, our reversible session-based calculus. Section 5 shows the results concerning the reversibility properties of *ReSπ*. Section 6 describes the associated typing discipline. Section 7 presents the extension of *ReSπ* with irreversible commit actions. Section 8 concludes the paper by touching upon directions for future work. Proofs of results are collected in Appendix A.

2. Related work

Our proposal combines the notion of (causal-consistent) reversibility with (typed) primitives supporting session-based interactions in concurrent systems. We review here some of the closely related works concerning either reversibility or session types.

Forms of reversible computation can be found in different formalisms in the literature. For example, backward reductions are considered in the λ-calculus to define equality on expressions [6]. Similar notions are used in the definitions of back and