COST Action IC1405
Reversible Computation
Extending Horizons of Computing

Working Group 2 - Software and Systems

Grant Period 2 report

Editors:
Claudio Antares Mezzina and Rudolf Schlatte

Contributors:
Paola Giannini, Rumyana Neykova, Claudio Antares Mezzina, Ulrik Schultz, German Vidal

June 6, 2017

1 Introduction

In this document we report the research activity that the Working Group 2 (WG2) on Software and System has actively undertaken during the second grant period of the Cost Action IC1405. According to the Action's Memorandum of Understanding [18] the WG2 mission is to provide linguistic abstractions, languages and tools to develop safer and more reliable applications. To do so, one of the main objective is to exploit reversibility to define actual reliability constructs and frameworks for programming reliable applications. This implies on one hand to integrate reversibility with mainstream programming languages and well know paradigms and on the other hand to understand what is the connection between reversibility and established techniques to achieve reliability such as transactions and checkpointing. Lastly, WG2 topics also include reversibility in robotics and control theory.

2 Progress of the WG

In a previous document [12], we have reported the relevant literature for the WG2. In this document, instead, we report the progress we have done into achieving the WG2 objectives, and in particular the integration of reversibility with respect to:

- modularity aspects such as classes and object
- type systems, with a special care on behavioural types (e.g., session types)
- mainstream languages (e.g., Erlang [1])
- recovery techniques in shared memory and massage passing concurrency models

Object oriented. Initial ideas for the design and implementation of reversible object-oriented language have been presented, based on extending Janus with object-oriented concepts such as classes that encapsulate behavior and state, inheritance, virtual dispatching, as well as constructors. Schultz and Axelsen showed that virtual dispatching can be seen as reversible decision mechanism that is easily translatable to a standard reversible programming model such as Janus, and argued that reversible management of state can be accomplished using reversible constructors [17]. These concepts were informally described and implemented by source-to-source translation from the reversible object-oriented language Joule to Janus. A similar design was adopted for the ROOP reversible object-oriented language, but fully formalized and implemented by compiling to the Pendulum reversible ISA [7, 8]. ROOP demonstrates how to generate low-level code implementing reversible vtable-based dispatching, and investigates the restrictions that must be imposed on reversible object-oriented programs to avoid run-time aliasing checks as found in Joule.

Session Types. In a series of works [4, 3] multiparty session types (aka global types) have been enriched with checkpoint labels on choices that mark points of the protocol where the computations may roll back. In [4] a simple model

in which rollback could be done any time after a participant had crossed the checkpointed choice. In [3] a more refined model is presented, in which the programmer can define points where the computation may revert to a checkpointed label, and rollback has to be triggered by the participant that made the decision.

Reversibility and monitored semantics for session types has been recently studied by Mezzina and Pérez [14, 13]. In their works, they propose a *monitor as memory* mechanism in which information about the monitor of a process can be used to enable its reversibility. Moreover, by adding *modalities* information at the level of session types, reversibility can be controlled.

Erlang. Recently, [16, 10] have also introduced reversibility in the context of the functional and concurrent programming language Erlang [1]. In particular, a modular semantics for (a subset of) *Core Erlang* [2] is first introduced in [16], which is particularly appropiate to define a reversible extension. A reversible, *uncontrolled* (according to the terminology in [9]) semantics is then defined in [10]. By defining an appropriate notion of concurrent transitions, the causal consistency of the reversible semantics is proved. Finally, adding control to the reversible semantics in the form of a *rollback operator* allows the design of a method to undo the actions of a given process up to a given checkpoint—introduced by the programmer. In order to ensure causal consistency, the rollback action might be propagated to other, dependent processes. In contrast to previous approaches to reversibility in μOz [11, 5], a main difference is that μOz is not distributed: messages move atomically from sender to a chosen queue, and from the queue to the receiver. Each of the two actions is performed by a specific thread, hence the action is naturally part of the associated thread history. In our case, message delivery is not directly performed by a thread, and only potentially observed when the target thread performs the receive action (but not necessarily observed, e.g., if the message does not match the patterns in the receive). The definition of the notions of causality and concurrency in this setting is as a consequence much more tricky than in μOz. This difficulty carries over to the definition of the history information that needs to be tracked, and to how this information is exploited in the reversible semantics.

Recovery. Recently [15], a static analysis based on multiparty session types that can efficiently compute a safe global state from which a system of interacting processes should be recovered, has been integrated with the Erlang recovery mechanism. From a global description of the program communication flow, given in multiparty protocol specification, causal dependencies between processes are extracted. This information is then used at runtime by a recovery mechanism, integrated in Erlang, to determine which process has to be terminated and which one has to be restarted upon a node failure. Experimental results indicate that the proposed framework outperforms a built-in static recovery strategy in Erlang when a part of the protocol can be safely recovered.

In [6] a rollback operator, based on the notion of causal-consistent reversibility, is defined for a language with shared memory. A rollback is defined as the minimal causal-consistent sequence of backward steps able to undo a given action.

References

[1] Joe Armstrong, Robert Virding, Claes Wikström, and Mike Williams. *Concurrent programming in Erlang (2nd edition)*. Prentice Hall, 1996.

[2] Richard Carlsson, Björn Gustavsson, Erik Johansson, Thomas Lindgren, Sven-Olof Nyström, Mikael Pettersson, and Robert Virding. Core Erlang 1.0.3. Language specification, 2004. Available from `https://www.it.uu.se/research/group/hipe/cerl/doc/core_erlang-1.0.3.pdf`.

[3] Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Paola Giannini. Concurrent reversible sessions. 2017.

[4] Mariangiola Dezani-Ciancaglini and Paola Giannini. Reversible multiparty sessions with checkpoints. In *EXPRESS/SOS'16*, volume 222 of *EPTCS*, pages 60–74, 2016.

[5] Elena Giachino, Ivan Lanese, and Claudio Antares Mezzina. Causal-consistent reversible debugging. In Stefania Gnesi and Arend Rensink, editors, *Proc. of the 17th International Conference on Fundamental Approaches to Software Engineering (FASE 2014)*, volume 8411 of *Lecture Notes in Computer Science*, pages 370–384. Springer, 2014.

[6] Elena Giachino, Ivan Lanese, Claudio Antares Mezzina, and Francesco Tiezzi. Causal-consistent rollback in a tuple-based language. *J. Log. Algebr. Meth. Program.*, 88:99–120, 2017.

[7] T. Haulund. Design and implementation of a reversible object-oriented programming language. Master's thesis, University of Copenhagen, DIKU, 2016.

[8] T. Haulund, T. Mogensen, and R. Glück. Implementing reversible object-oriented language features on reversible machines. In *International Conference on Reversible Computation*. Springer, 2017. To appear.

[9] Ivan Lanese, Claudio Antares Mezzina, and Francesco Tiezzi. Causal-consistent reversibility. *Bulletin of the EATCS*, 114, 2014.

[10] Ivan Lanese, Naoki Nishida, Adrián Palacios, and Germán Vidal. A Theory of Reversibility for Erlang, 2017. Submitted for publication.

[11] Michael Lienhardt, Ivan Lanese, Claudio Antares Mezzina, and Jean-Bernard Stefani. A reversible abstract machine and its space overhead. In Holger Giese and Grigore Rosu, editors, *Proceedings of the Joint 14th IFIP WG Int'l Conf. on Formal Techniques for Distributed Systems (FMOODS 2012) and the 32nd IFIP WG 6.1 International Conference (FORTE 2012)*, volume 7273 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2012.

[12] Claudio Antares Mezzina and Rudolf Schlatte (eds). State of the art report, Working Group 2, Software and Systems. COST Action IC1405, Reversible Computation, 2016.

[13] Claudio Antares Mezzina and Jorge A. Pérez. Reversible sessions using monitors. In *Proceedings of the Ninth workshop on Programming Language Approaches to Concurrency- and Communication-cEntric Software, PLACES 2016, Eindhoven, The Netherlands, 8th April 2016.*, volume 211 of *EPTCS*, pages 56–64, 2016.

[14] Claudio Antares Mezzina and Jorge Andrés Pérez. Reversible semantics in session-based concurrency. In *Proceedings of the 17th Italian Conference on Theoretical Computer Science, Lecce, Italy, September 7-9, 2016.*, volume 1720 of *CEUR Workshop Proceedings*, pages 221–226. CEUR-WS.org, 2016.

[15] Rumyana Neykova and Nobuko Yoshida. Let It Recover: Multiparty Protocol-Induced Recovery. In *26th International Conference on Compiler Construction*, pages 98–108. ACM, 2017.

[16] Naoki Nishida, Adrián Palacios, and Germán Vidal. A Reversible Semantics for Erlang. In Manuel Hermenegildo and Pedro López-García, editors, *Proc. of the 26th International Symposium on Logic-Based Program Synthesis and Transformation, LOPSTR 2016*, Lecture Notes in Computer Science. Springer, 2017. To appear.

[17] U.P. Schultz and H.B. Axelsen. Elements of a reversible object-oriented language. In *International Conference on Reversible Computation*, pages 153–159. Springer, 2016.

[18] Irek Ulidowski. IC1405 - Reversible Computation: extending horizons of computing - Memorandum of Understanding. `https://e-services.cost.eu/files/domain_files/ICT/Action_IC1405/mou/IC1405-e.pdf`.