ICT COST Action IC1405

COST Action IC1405
Reversible Computation
Extending Horizons of Computing

Working Group 2 - Software and Systems

Grant Period 3 report

Editors:
Claudio Antares Mezzina and Rudolf Schlatte

Contributors:
Paola Giannini, Claudio Antares Mezzina,
Jaroslaw Miszczak, Jorge A. Pérez, Irek Ulidowski,
Ulrik Schultz, German Vidal

May 21, 2018

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 third grant period (GP3) of the Cost Action IC1405. According to the Action's Memorandum of Understanding [22] 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 [18], we have reported the relevant literature for the WG2, while in [17] we have reported the second year activities of the WG2 towards achieving our objectives. By continuing the work done in [17], in this document we report the progress that have been done during the third year of the Action. In particular, we report on the integration of reversibility with respect to:

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

Session Types. In a series of works [6, 4] 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 [6] a simple model in which rollback could be done any time after a participant had crossed the checkpointed choice. In [4] 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.

The interplay of reversibility and monitored semantics for *multiparty session types* has been recently studied by Mezzina and Pérez [19]. In the context of multiparty session types, global types describe the message-passing behaviour of a set of participants in a system from a global point of view. A global type can be projected onto each participant so as to obtain local types, which describe individual contributions to the global protocol. The work [19] extends global and local types to keep track of the stage of the protocol that has been already executed; this enables reversible steps in an elegant way. The authors develop

a rigorous process framework for multiparty communication, which improves over prior works by featuring asynchrony, decoupled rollbacks and process passing. In this framework, concurrent processes are untyped but their forward and backward steps is governed by monitors. The main technical result is that the developed multiparty reversible semantics is causally-consistent.

Erlang. A general framework for reversible computation in the concurrent and distributed functional language Erlang [1] has been introduced in [16]. First, a standard semantics of the language is presented. This semantics is *modular*: one layer defines the semantics of expressions and another layer the reductions of complete systems. This modular design greatly simplifies the definition of a reversible extension since only the system layer needs to be changed. The authors first present an uncontrolled reversible semantics which is proved to be causal consistent. Then, a controlled version is introduced, where the rules are driven by a set of "rollbacks". The controlled semantics is proved sound with respect to the uncontrolled semantics and confluent. Finally, a proof-of-concept implementation is presented, which witnesses the viability of the reversible semantics in practice.

More recently, a tool called CauDEr [15], a causal-consistent reversible debugger for Erlang, has been developed following (and extending) the results in [20, 16]. In particular, CauDEr includes a new rollback semantics which is especially tailored to debugging Erlang programs. A set of rollback commands allow one to go backwards, in a causal consistent way, up to the spawning of a process, the sending of a particular message, etc. In this way, the user can focus on the actions that are most likely responsible of the faulty behaviour. The paper shows that some bugs can be more easily located using CauDEr, thus filling a gap in the collection of debugging tools for Erlang. CauDEr is publicly available from `https://github.com/mistupv/cauder`.

Object Oriented. The recent work on reversible object-oriented (OO) languages started with a discussion of the key elements of reversible OO languages, introduced with a prototype of the Joule language [21]. Reversible OO language were subsequently formally described for the ROOPL language [13]. The original Joule prototype relied on static and stack allocation of objects, which does not permit garbage-free OO programming: common patterns such as factories are for example not possible [21]. The initial presentation of the ROOPL language relied exclusively on stack allocation [13], but has subsequently been extended with a heap-based memory manager [5] — this is however a significantly more complex system that may not be entirely garbage-free [2].

Recovery. Distributed programs are hard to get right because they are required to be open, scalable, long-running, and tolerant to faults. In particular, the recent approaches to distributed software based on (micro-)services where different services are developed independently by disparate teams exacerbate the problem. In fact, services are meant to be composed together and run in open context where unpredictable behaviours can emerge. This makes it necessary to adopt suitable strategies for monitoring the execution and incorporate recovery and adaptation mechanisms so to make distributed programs more flexible and robust. The typical approach that is currently adopted is to embed

such mechanisms in the program logic, which makes it hard to extract, compare and debug. An approach that employs formal abstractions for specifying failure recovery and adaptation strategies has been proposed in [3]. Although implementation agnostic, these abstractions would be amenable to algorithmic synthesis of code, monitoring and tests. Message-passing programs (à la Erlang, Go, or MPI) are considered, since they are gaining momentum both in academia and industry. In [7] an instance of the framework proposed in [3] is given. More precisely, this approach imbues the communication behaviour of multi-party protocols with minimal decorations specifying the conditions triggering monitor adaptations. It is then shown that, from these extended global descriptions, one can (i) synthesise actors implementing the normal local behaviour of the system prescribed by the global graph, but also (ii) synthesise monitors that are able to coordinate a distributed rollback when certain conditions (denoting abnormal behaviour) are met. The synthesis algorithm produces Erlang code. More precisely, for each role in the global description are generated two Erlang actors: one implementing the normal (forward) behaviour of the system and a second one (the monitor) in charge of implementing the reversible behaviour of the role. When certain condition are met at runtime, then the monitors will coordinate each other in order to bring back, if possible, the system. One interesting property of such approach is that the two semantics are highly decoupled, meaning that the system is always able to normally execute (e.g., going forward) even in case of some monitor crash.

Shared memory. The interplay between reversibility and imperative programming language with shared memory is studied in [14]. This paper proposes an approach and a subsequent extension for reversing imperative programs. Firstly, both an augmented version and a corresponding inverted version of the original program are produced. Augmentation saves reversal information into an auxiliary data store, maintaining segregation between this and the program state, while never altering the data store in any other way than that of the original program. Inversion uses this information to revert the final program state to the state as it was before execution. It is shown that augmentation and inversion work as intended, and the approach is illustrated via several examples. Also a modification to the first approach to support non-communicating parallelism is suggested. Execution interleaving introduces a number of challenges, each of which the extended approach considers. This requires to define annotation and redefine inversion to use a sequence of statement identifiers, making the interleaving order deterministic in reverse.

Quantum. In [12] it is described QSWalk.jl package for Julia programming language [10], developed to simulate the evolution of open quantum systems. The package enables the study of reversible quantum procedures developed using stochastic quantum walks on arbitrary directed graphs. A detailed description of the implemented functions is provided along with some usage examples. The package has been used for the purpose of determining the differences between limiting properties in various models of quantum stochastic walks [11].

The idea of using Julia as a host language for the simulator of quantum computing was motivated by the solid support for numerical procedures available in this language. Moreover, the strong typing capabilities of Julia has been used

for developing type hierarchy for various models of quantum walks. The implementation of this hierarchy is available as a library of functions [9]. This library has been used for proposing a framework suitable for analysing the efficiency of attacks on quantum search algorithms [8].

References

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

[2] Holger Bock Axelsen and Robert Glück. Reversible representation and manipulation of constructor terms in the heap. In *RC2013*, pages 96–109. Springer, 2013.

[3] Ian Cassar, Adrian Francalanza, Claudio Antares Mezzina, and Emilio Tuosto. Reliability and fault-tolerance by choreographic design. In Adrian Francalanza and Gordon J. Pace, editors, *Proceedings Second International Workshop on Pre- and Post-Deployment Verification Techniques, PrePost@iFM 2017, Torino, Italy, 19 September 2017.*, volume 254 of *EPTCS*, pages 69–80, 2017.

[4] Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Paola Giannini. Concurrent reversible sessions. In *Proceedings 28th International Conference on Concurrency Theory, CONCUR 2017*, volume 85 of *LIPIcs*, pages 30:1–30:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[5] Martin Holm Cservenka. Design and implementation of dynamic memory management in a reversible object-oriented programming language. Master's thesis, DIKU, University of Copenhagen, 2018.

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

[7] Adrian Francalanza, Claudio Antares, and Emilio Tuosto. Reversible choreographies via monitoring in erlang. In Silvia Bonomi and Etienne Rivière, editors, *Distributed Applications and Interoperable Systems - 18th IFIP WG 6.1 International Conference, DAIS 2018*, Lecture Notes in Computer Science. Springer, 2018. to appear.

[8] A. Glos and J.A. Miszczak. Impact of the malicious input data modification on the efficiency of quantum algorithms. *arXiv:1802.10041*, 2018.

[9] A. Glos and J.A. Miszczak. QuantumWalk.jl: Package for building algorithms based on quantum walks, 2018. https://github.com/QuantumWalks/QuantumWalk.jl.

[10] A. Glos, J.A Miszczak, and M Ostaszewski. QSWalk.jl: simulating the evolution of open quantum systems on graphs, 2017. https://github.com/QuantumWalks/QSWalk.jl.

[11] A. Glos, J.A. Miszczak, and M. Ostaszewski. Limit properties of global interaction stochastic quantum walks on directed graphs. *J. Phys. A: Math. Theor*, 51:035304, 2018. arXiv:1703.01792.

[12] A. Glos, J.A. Miszczak, and M. Ostaszewski. Qswalk. jl: Julia package for quantum stochastic walks analysis. *arXiv:1801.01294*, 2018.

[13] Tue Haulund, Torben Ægidius Mogensen, and Robert Glück. Implementing reversible object-oriented language features on reversible machines. In Iain Phillips and Hafizur Rahaman, editors, *Reversible Computation*, pages 66–73, Cham, 2017. Springer International Publishing.

[14] James Hoey, Irek Ulidowski, and Shoji Yuen. Reversing imperative parallel programs. In Kirstin Peters and Simone Tini, editors, *Proceedings Combined 24th International Workshop on Expressiveness in Concurrency and 14th Workshop on Structural Operational Semantics and 14th Workshop on Structural Operational Semantics, EXPRESS/SOS*, volume 255 of *EPTCS*, pages 51–66, 2017.

[15] Ivan Lanese, Naoki Nishida, Adrián Palacios, and Germán Vidal. CauDEr: A Causal-Consistent Reversible Debugger for Erlang. In *Proceedings of the 14th International Symposium on Functional and Logic Programming, FLOPS 2018*, volume 10818 of *LNCS*. Springer, 2018. To appear.

[16] Ivan Lanese, Naoki Nishida, Adrián Palacios, and Germán Vidal. A theory of reversibility for erlang, 2018. Submitted for publication.

[17] Claudio Antares Mezzina and Rudolf Schlatte (eds). Working Group 2, Software and Systems, Report of the Second Year. COST Action IC1405, Reversible Computation, 2016. `http://topps.diku.dk/ic1405/wg2_soar.pdf`.

[18] Claudio Antares Mezzina and Rudolf Schlatte (eds). State of the art report, Working Group 2, Software and Systems. COST Action IC1405, Reversible Computation, 2017. `http://topps.diku.dk/ic1405/WG2yearendreport2017.pdf`.

[19] Claudio Antares Mezzina and Jorge A. Pérez. Causally consistent reversible choreographies: a monitors-as-memories approach. In Wim Vanhoof and Brigitte Pientka, editors, *Proceedings of the 19th International Symposium on Principles and Practice of Declarative Programming*, pages 127–138. ACM, 2017.

[20] 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.

[21] Ulrik Pagh Schultz and Holger Bock Axelsen. Elements of a reversible object-oriented language. In Simon Devitt and Ivan Lanese, editors, *Reversible Computation*, pages 153–159, Cham, 2016. Springer International Publishing.

[22] 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.