COST Action IC1405
Reversible computation

extending horizons of computing

Working Group 4 Report on Case Studies

Editors:
Ulrik Pagh Schultz and Carla Ferreira

Contributors:
Carla Ferreira, Stefan Kuhn, Ivan Lanese,
Markus Schordan, Ulrik Pagh Schultz, and Irek Ulidowski

July 28, 2016

Contents

Preface

This report presents the case studies initially selected for subsequent investigation in the context of the COST Action IC 1405 on reversible computation, collected and edited by Working Group 4. The descriptions of the specific case studies have been provided by the researchers interested in investigating the corresponding case studies.

More on the project:

- `http://www.cost.eu/COST_Actions/ict/IC1405`
- `http://www.revcomp.eu/`

1 Overview

1.1 Introduction

As outlined in the Memorandum of Understanding (MoU), the theories, techniques and support software tools delivered by Working Groups 1 to 3 will be validated and their suitability for practical applications will be assessed using case studies. The case studies will be used to provide feedback on the effectiveness of the reversibility-inspired theories, techniques, and solutions, and can point to new applications and research questions.

During the first year of the action case studies have been selected and developed based on an interest-driven bottom-up approach, matching member research interests with case studies carrying a potential for external collaboration, either with industry or with researchers outside the area of reversible computing.

1.2 Selected case studies

Five case studies have been selected based on interest from members of the action:

1. Debugging of concurrent and distributed systems
2. Reliability of distributed systems through reversibility
3. Reversibility in chemical reactions
4. Reversible control of robots
5. Simulator for optimistic parallel discrete event simulation

In addition, case studies on synthesis of reversible circuits, reversible execution in transactional memory and reversible algorithms are likely to be developed during the second year. A potential case study on catalytic-space computation was not developed since it did not fit with the research topics of the members. Compared to the topics outlined in the MoU, only the suggested case study on "Demonstration versions and prototypes for reversible circuits" is not currently planned for development.

1.3 Research areas

The research areas covered by the selected case studies are outlined in Table 1. The areas are divided into Reversible Computing (RC) topics and non-RC topics. There is a good coverage of RC topics from Working Groups 1 and 2, but not currently any RC topics from Working Group 3. The proposed case studies on synthesis for reversible circuits and on reversible algorithms should however involve Working Group 3.

All case studies are based on significant interaction with researchers from areas outside reversible computing. Moveover, the case studies on debugging, robots and simulation all have significant potential for industrial collaboration.

	RC topics	Non-RC topics
Debugging of concurrent and distributed systems	reversible language, checkpointing, causal-consistent reversibility, causal analysis	debugging, concurrent and distributed systems
Reliability of distributed systems through reversibility	reversible language, automatic recovery, reversible transactions, causal-consistent reversibility	concurrent and distributed systems, transactions, replication
Reversibility in chemical reactions	reversible language, reversible process calculi, out-of-causal order reversibility, local reversibility	modelling of chemical reactions, life sciences modelling
Reversible control of robots	reversible language, inversion, backtracking, reversible debugging, reversible concurrency	robotics, industrial robots, domain-specific languages, modular robots, swarm robots
Simulator for optimistic parallel discrete event simulation	reversible language, out-of-causal order reversibility	simulation, high-performance computing

Table 1: Research areas covered by the selected case studies

2 Case studies

We now present the five case studies, in each case providing an overall conceptual description, a concrete example, and an estimate of the potential impact.

2.1 Debugging of Concurrent and Distributed Systems

Involved members: Ivan Lanese, Claudio Mezzina

2.1.1 Conceptual description

Finding bugs inside software has always been a main activity in the software development life cycle [BJC12], and reversibility can play a role here [Eng12, KDC05, GLM14]. In fact, reversibility allows one to go back in the past of the program, looking for the bug that caused a given misbehaviour. Indeed, many debuggers provide features for reversible execution, including GDB [JC09] and UndoDB [UDB]. However, current reversible debuggers either do not support concurrency/distribution, or they just fix an interleaving among threads and stick to it. It is not yet clear which is the good way to debug a concurrent and/or distributed system, what is potentially very interesting. Indeed, finding bugs inside concurrent and distributed software is more difficult than inside

sequential software [LPS08], since faults may appear or disappear according to the speed of the different processes and of the network communications. The bugs generating these faults, called Heisenbugs, are particularly difficult to find. Note that, by exploring back and forward the same computation one is guaranteed that Eisenbugs that occurred will not disappear (provided that the approach preserves causal dependencies between actions from different threads).

2.1.2 Concrete example

The causal-consistent approach to reversibility [LMT14] allows one to undo any action provided that its consequences, if any, are undone beforehand. This relates reversibility to causality, instead of time, making it more suitable to distributed systems where a unique notion of time may not exist. In debugging, the causal-consistent approach allows one to find bugs by following backward causal dependencies from the misbehaviour to the bug causing it. This is supported by primitives allowing one to undo a past action, together with all and only its distributed consequences [LMS11]. For instance, if one observes that, e.g., the value of a variable x is 5 instead of the expected 6, one can undo the last assignment to x, reaching a state where the assignment could have been performed. A causal-consistent reversible debugger for the toy language muOz has been presented in [GLM14], and is available at [CRD].

2.1.3 Potential impact

Debugging is a topic very relevant from the practical point of view, but mostly neglected by academic research. Concerning impact, a 2012 study by the Judge Business School of the University of Cambridge, UK, sponsored by Rogue Wave Software, goes as far as to estimate that the time spent in debugging corresponds to 49.9% of total programming time [BJC12]. According to another recent study [US14], the cost of debugging software amounts to $312 billions annually. Hence, any improvement in debugging techniques has potential huge practical impact. Given that systems are becoming more and more complex, and concurrency and distribution is more and more relevant, the relevance of debugging will not decrease in the close future.

There is a significant potential for industrial collaboration with Undo Software (Cambridge, UK), which is a company concentrated in reversible debugging. They are mainly concerned with sequential software, and in case of concurrency their approach is currently to consider a linearization of the execution. One of their main concerns and selling points is the limited overhead of their debugger, both in time and size of the logs. Still, they want to continue improving these features. Their approach is based on taking snapshots of the state at variable rates, according to how likely the current part of the execution will need to be reversed. Most of their work is in low-level optimizations and in providing support for different compilers and operating system configurations. Undo Software currently has a limited interest in concurrency and distribution, with the more relevant open points for them being parallel logging (since sequen-

tial logging requires to linearize execution, hence an unacceptable slow down), and debugging of MPI programs (since their customers are interested in this programming library).

2.2 Reliability of distributed systems through reversibility

Involved members: Carla Ferreira, Ivan Lanese, Claudio Mezzina, Vasileios Koutavas

2.2.1 Conceptual description

The emergence of global services, operating over massive distribution of data and computation, has lead to the recent paradigm of cloud computing. In this global context building reliable systems is very hard, but reversibility can be used to achieve reliability in a natural way. If something goes wrong just undo the previous actions and return to a past consistent state. However, in these highly distributed systems there is no notion of global time, so there might not be a clear previous action. Instead, actions are causally ordered. Leveraging on [DK04], that defined causal-consistent reversibility, this case study will demonstrate the use of reversibility as a recovery mechanism for highly distributed systems. The goal is that a programmer can build a reversible distributed application for free, in the sense s/he just has to program the forward actions and whenever something goes wrong all actions can be reverted automatically. This was already explored in the context of long-running transactions (see [CP13] for an overview of calculi with support for compensations / backward actions). In long-running transactions, backward actions were defined by the programmer and the relation between forward and backward actions could be quite tenue. In this case study, we would want to explore how backward actions can be determined automatically for concurrent and distributed systems over a shared state.

Another axis to be explored by the case study is the use of replicated databases to achieve service availability and scalability, as it is common practice in global services. In this setting, clients execute operations over the closest database replica and later these operations are propagated to other remote replicas. Ensuring automatic reversibility in this case is even harder as coordination is reduced (on weaker models of consistency coordination is avoided altogether). It should be explored how to ensure reversibility in this new context.

2.2.2 Concrete example

The concrete examples should include two classes of distributed systems. Distributed and concurrent systems that operate over a shared state, and distributed systems that operate over a replicated state. Several well known distributed systems case studies used in the literature can be used: transactional

web server for a online store [TPC], an auction system, and a student registration system. These specific case studies should show the applicability and relevance of reversibility for distributed systems.

2.2.3 Potential impact

Building concurrent and distributed systems is hard and reversible languages can play an important role in helping programmers developing reliable software by providing reversibility by construction. Currently, error handling has to be done explicitly by the programmer, but if reversibility could be achieved by pressing a button it would have an impact on the quality of the programs developed. In fact, reversibility could be used as a way of providing speculative execution in cloud applications.

2.3 Reversibility in chemical reactions

Involved members: Irek Ulidowski, Stefan Kuhn, Gabriel Ciobanu, Bogdan Aman

2.3.1 Conceptual description

Biological reactions, pathways and reactions networks have been studied using various techniques, including process calculi. The initial research focused on reaction rates, by modelling and simulating networks of reactions, in order to analyse or even predict the common paths through the network. Reversibility was not considered there explicitly. Once a more structural approach was taken reversibility started to be taken into account, since it plays a crucial role in many processes, typically by going back to a previous state in the system. For example, if a gene is regulated by a protein binding to and unbinding from it, then reversibility is needed if this to be modelled directly. Such examples can be modelled by backtracking along the path taken to the current state. Beyond backtracking and causal reversibility, there are more general forms of reversibility, specifically the out-of-causal-order reversibility, which make it possible to get to states which cannot be reached by forward reactions alone. Such sequences of forward and reversible reactions are important as they lead to new chemical structures and new reactions, which would not be possible if the out-of-causal reversibility was not used.

Looking at proteins and other macromolecules means working at relatively high levels of abstraction. It also means that such biological entities are often difficult to represent as there is still no complete understanding of their behaviour. Also their interactions with other similar molecules are hard to enumerate as they are sometime not fully understood. This could also mean that some aspects of reversibility are not clearly modelled since we have abstracted too much detail. Therefore, we suggest to consider chemical reactions when studying reversibility in the context of living systems.

Chemical reactions involve various forms of reversibility and offer good case studies for the modelling of reversibility. Specifically, we can find interesting

examples of the out-of-causal order reversibility in many chemical reactions, and can study the intermediate steps of such reactions where some bonds are only helping to achieve the overall aim of the reaction: are only formed to be broken before the end such reactions. The level of detail we have about such chemical reactions should help with seeing reversibility in action in the ways not obvious at higher levels. As an additional benefit, in many cases a good understanding of the underlying reaction mechanisms exists.

2.3.2 Concrete example

We suggest to study the following three types of reactions:

- The hydration of formaldehyde in water: This is a basic textbook reaction, but there are different paths through the reaction, some quite rare, and the reaction is reversible. Clearly backtracking is not sufficient to model this, since it is experimentally possible to go forward via one path and undo reactions via a different path.

- The Monsanto process: This is a typical catalytic cycle involving a metal catalyst. We have two input molecules and want to produce another molecule by combining them. They cannot interact directly because too much energy would be required. By introducing the metal catalyst this can be overcome the catalyst breaks the input molecules and enables their recombination. The catalyst is restored to its original form after the process making it a cycle. Clearly, there must be undoing of bonds created by the catalyst. If fact, some bonds are undone via the out-of-causal order computation.

- The glucose-fructose isomerization: This is a reaction forming part of the Krebs cycle, an important pathway in the metabolism of organisms. It is an enzyme-assisted catalytic process, where an enzyme interacts with the glucose to produce fructose. Similarly to the Monsanto process the protein emerges unchanged. The mechanism, however, involving both causal and out-of-causal order reversibility, is different.

These examples display various types of reversibility, including the out-of-causal order reversibility. All examples have been studied extensively and are well understood at a very detailed level. There are also aspects of reactions which may be included at a later stage, for example the energy levels of the various steps of reactions.

The aim is to model these reactions using different existing techniques. We currently envisage the use of the kappa-calculus [DL04], P systems [CDK03, Pau00] and reversible process calculi [DK03, LMS11, PU07], including the recently developed [KU15]. The case study should give a clearer view of what the individual calculi can do for each of the examples and clarify their strengths and weaknesses.

2.3.3 Potential impact

The work can help to identify work which needs to be done to get a wider range of reversible processes represented in computing. Furthermore it can help with building knowledge about chemical reactions. Regarding the potential for external/industrial collaboration, there is interest in reaction simulation in the chemoinformatics area. A collaboration could be a long-term goal.

2.4 Reversible Control of Robots

Involved members: Ulrik Pagh Schultz, Gabriel Ciobanu

2.4.1 Conceptual description

Robots normally have one or more degrees of freedom controlled by a computational process; using reversible computing to control the robot potentially gives rise to new reverse behaviours. For example, major industrial robot manufacturers such as ABB and KUKA offer limited forms of ad-hoc reverse execution for interactive programming and debugging, but due to limitations in the underlying execution models, their programming models are incapable of reversing complex actions such as steps of an assembly process [LSE15, MAH10]. We attribute the ad-hoc limitations to the lack of an underlying reversible model. The first investigation of fully reversible robot behaviours was for self-reconfigurable robots [SBS11]. To better understand the underlying relation between reversible computation and physical reversibility, we will investigate reversible control of industrial robots and distributed robots.

Programming industrial robots is challenging due to the difficulty of precisely specifying general yet robust operations. As the complexity of these operations increases, so does the likelihood of errors. Certain classes of errors during industrial robot operations can however be addressed using reverse execution, allowing the robot to temporarily back out of an erroneous situation, after which the operation can be automatically retried. Specifically, this approach has been shown to be useful for automatic error recovery for small-sized batch production of assembly operations [SLE15]. Moreover, reversibility can in this case be used to automatically derive a disassembly sequence from a given assembly sequence, or vice versa. These results were demonstrated using an initial design and implementation of a reversible domain-specific languages (DSL) for specifying such assembly sequences [SLE15, LSE15]. The area however remains largely unexplored, both from a theoretical and practical point of view. There is for example a large design space for different programming language approaches, both in terms of the generality of the language and the means by which reversibility is achieved. At a more fundamental level, the notion of reversible control of a reversible physical system remains largely unexplored. From a practical point of view, only the specific case of assembly operations has been investigated, and only using a specific set of industrial use cases. There has been no attempt at integration into an existing robotics platform, although

we observe that many existing platforms offer limited notions of reversibility for using during programming and debugging.

Distributed robotic systems can make use of reverse execution as a means to reversing physical actions or correcting errors. As a concrete example, previous work has studied self-reconfigurable, modular robots which are distributed mechatronic devices that can autonomously change their physical shape. Self-reconfiguration from one shape to another is typically achieved through a specific sequence of actuation operations distributed across the modules of the robot. Automatically reversing the sequence of operations brings the robot back to its initial shape, which has been experimentally demonstrated using the DynaRole reversible language [SBS11]. DynaRole however only allows simple sequences of operations to be reversed, which is suitable for reversing self-reconfiguration sequences but lacks the generality needed to implement more complex behaviors. Initial ideas on generalizing the DynaRole language to support a wider range of scenarios, while retaining the possibility of reversing distributed sequences, have been proposed [Sch12]. Reversibility is used here as a practical feature, reducing the programming task of the programmer, and allowing error recovery by backing out of an error state using reverse execution. Currently there however is no underlying model allowing a distributed robotic systems to use reversibility for backing out of distributed operations that have become blocked, e.g., due to unexpected features of the environment, to under conflicting operations, or recover from partial hardware failures.

2.4.2 Concrete example

An appropriately robust and useful reversible DSL for control of industrial robots could be integrated into an existing robotics platform, and could be useful both for working interactively with the robot (programming and debugging) but also for error recovery during specific kinds of operations. Note that many robotic systems already integrate limited forms of reverse execution for interactive debugging [MAH10], better understanding and improving these approaches to interactive debugging is probably interesting.

Using reversible computing to allow a subset of a distributed robotic system to optimistically proceed with an operation, but then backtrack and undo the operation if the system globally decides not to perform this operation. This is interesting not only for modular robots but also for swarm robotic system e.g. drones. A single mobile robot navigating an environment can be modeled as a 2D Turing machine [Tho14], presumably a useful model for multi-robot systems navigating an environment could be based on a similar model.

2.4.3 Potential impact

From a scientific point of view, the design of reversible languages for controlling reversible processes remains an open question. Moreover, exploring reversible control of physical processes may help to better understand the principles of reversible computing. From a society point of view, industrial robots are key to

maintaining production in Europe, and reversible computation has the potential to increase robustness for specific kinds of operations such as assembly, and moreover facilitate the programming of such operations.

There is a significant potential for external/industrial collaboration for industrial robots: Universal Robots ApS is a potential industrial collaborator, as are potentially also other robotics companies or perhaps companies performing large numbers of error-prone assembly operations. For modular robots, we however note that swarm and modular robots are mostly at the level of basic research in robotics, with the obvious exception of drones (but here reversibility seems less useful).

2.5 Simulator for optimistic parallel discrete event simulation

Involved members: Markus Schordan

2.5.1 Conceptual description

Discrete event simulation (DES) is a simulation paradigm suitable for systems whose states are modeled as changing discontinuously and irregularly at discrete moments of simulation time. DES is event-driven in that the times at which state changes occur are calculated dynamically rather than statically as in time-stepped simulations. Example applications include simulations of digital communication networks, vehicular traffic flow, markets and economies, epidemiological models, logistical models, ecological and population models, tactical and strategic military models, and many others. Most systems whose behavior is not describable by continuous equations and that are not suitable for simple time-stepped models are candidates for DES.

Reverse computation has become a central notion in parallel discrete event simulation (PDES) over the last decade [CPF99, Per13]. It is not just a theoretical line of research, but an immensely practical one necessary to achieve high performance for large parallel discrete event simulations. In fact, the most highly parallel and fastest discrete event simulation benchmarks ever executed have made essential use of reverse computation [BCJ13].

The fundamental issue that makes PDES so complex is the synchronization problem. Every logical process (LP) must execute all events from its own event queue in strictly non-decreasing timestamp order despite the fact that it does not generally know which LPs might be sending it events, or how many, or what timestamps they may carry. Furthermore there is no guarantee that event messages will arrive at an LP in increasing timestamp order. Optimistic synchronization is that all event executions are speculative or provisional, and are always subject to rollback if the simulation gets into local synchronization trouble. Most of the time that does not happen and the simulation proceeds forward in parallel, but occasionally a causality violation occurs that has to be corrected by rollback (achieved by reversible computation) and then the simulation continues forward again. Therefore, reversible computation is paramount

for achieving high-performance in optimistic parallel discrete event simulations on supercomputers. Various approaches to reversibility have been developed over the years and have they been successfully applied in supporting optimistic PDES, with recent contributions in incremental state-saving [MOJ16, SJB15], reverse code generation [PP14, LGC14], utilization of transactional memory [SIP15], and hybrid combinations of these approaches [PVQ15, CPQ15].

2.5.2 Concrete example

We suggest to use an existing simulator as basis for optimistic PDES which allows to plug-in reverse code for implementing the rollback as required for optimistic PDES. A well known simulator is ROSS (Rensselaer's Optimistic Simulation System), a general purpose discrete event simulator developed at RPI by C. Carothers et. al. [HC08].

ROSS helped to demonstrate that Time Warp simulators can be run efficiently both in terms of speed and memory usage relative to a high-performance sequential simulator and was used to achieve the world-record in 2013 for the most highly parallel and fastest discrete event simulation benchmarks ever executed [BCJ13]. In this strong scaling study of the ROSS simulator running Time Warp with reverse computation, had 251 million PHOLD logical processes and was executed in several configurations with up to a peak of 7.86 million MPI tasks running on 1,966,080 cores. At the largest scale it processed 33 trillion events in 65 seconds, yielding a sustained speed of 504 billion events/second using 120 racks of Sequoia. This is by far the highest event rate reported by any parallel discrete event simulation to date, whether running PHOLD or any other benchmark.

In order to use Time Warp in a ROSS model, a reverse event function must be provided. For a given event message it is responsible for undoing the state changes that the forward event function incurred for the same message. ROSS is open-source and also comes with a number of models (implemented in C/C++) with efficient hand-written reverse functions. These can be used as a basis for comparing different automated approaches to reversibility and perform comparative evaluations with the existing reversible ROSS implementations.

2.5.3 Potential impact

It is important to investigate how different approaches to reversibility can attack the problem of making the execution of a model's event function reversible, because any overhead introduced in the forward execution or reverse computation (dependent on the approach to reversibility) impacts the performance of the simulation. In particular, the combination of approaches is interesting for large-scale simulation models, as they are usually a combination of components with different computational properties and challenges for reversibility. If successful, the impact would be that this can bring also other approaches (from the COST project) to be used — or at least to be evaluated to be suitable — to HPC supercomputers, and may also help to improve the performance of sim-

ulations. In particular, also reversible languages such as Janus [YAG08] that can be translated to C are candidates for integration with other approaches, and can be directly evaluated and compared to existing models utilizing reverse code generation. There is potential for industrial collaboration with Lawrence Livermore National Laboratory.

References

[**BCJ13**] Barnes, Jr., P.D., Carothers, C.D., Jefferson, D.R., LaPre, J.M.: Warp speed: Executing time warp on 1,966,080 cores. In: Proceedings of the 2013 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation. SIGSIM-PADS '13, New York, NY, USA, ACM (2013) 327-336.

[**BJC12**] T. Britton, L. Jeng, G. Carver, P. Cheak, and T. Katzenellenbogen. Reversible debugging software - quantify the time and cost saved using reversible debuggers. `http://www.roguewave.com`, 2012.

[**CDK03**] G. Ciobanu, R. Desai and A. Kumar. Membrane systems and distributed computing. In Membrane Computing. International Workshop, WMC-CdeA 2002 Curtea de Arges 2002, pages 187-202. Springer, 2003.

[**CP13**] Christian Colombo and Gordon J. Pace. 2013. Recovery within long-running transactions. ACM Comput. Surv. 45, 3, pages 28:1 28:35

[**CPF99**] Carothers, C.D., Perumalla, K.S., Fujimoto, R.M.: Efficient optimistic parallel simulations using reverse computation. ACM Trans. Model. Comput. Simul. 9(3) (July 1999) 224-253.

[**CPQ15**] D. Cingolani, A. Pellegrini, and F. Quaglia. Transparently mixing undo logs and software reversibility for state recovery in optimistic pdes. In Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, SIGSIM PADS '15, pages 211-222, New York, NY, USA, ACM, 2015.

[**CRD**] CaReDeb `http://www.cs.unibo.it/caredeb`

[**DK03**] V. Danos and J. Krivine. Formal molecular biology done in CCS-R. In Proceedings of the 1st Workshop on Concurrent Models in Molecular Biology BioConcur 2003, volume 180 of ENTCS, pages 3149, 2007.

[**DK04**] V. Danos and J. Krivine. Reversible communicating systems. In CONCUR04, volume 3170 of LNCS. Springer, 2004.

[**DL04**] V. Danos and C. Laneve. Formal molecular biology. Theoretical Computer Science, 325(1):69110, 2004.

[**Eng12**] J. Engblom, A review of reverse debugging, in: System, Software, SoC and Silicon Debug, IEEE, 2012, pp. 1-6.

[**GLM14**] E. Giachino, I. Lanese, C. A. Mezzina: Causal-Consistent Reversible Debugging. FASE 2014: 370-384

[**HC08**] A. O. Holder and C. D. Carothers. Analysis of time warp on a 32,768 processor IBM Blue Gene/L supercomputer. In Proceedings of the European Modeling and Simulation Symposium (EMSS), 2008.

[**JC09**] J. Conrod. Tutorial: Reverse debugging with GDB 7, 2009 `http://jayconrod.com/posts/28/tutorial-reverse-debugging-with-gdb-7`

[**KDC05**] S. T. King, G. W. Dunlap, and P. M. Chen. Debugging operating systems with time-traveling virtual machines. In USENIX Annual Technical Conference, General Track, pages 1-15, 2005.

[**KU15**] S. Kuhn, and I. Ulidowski. Towards modelling of local reversibility. In Proceedings of Reversible Computation 2015, pages 279284. Springer, 2015.

[**LGC14**] J. M. LaPre, E. J. Gonsiorowski, and C. D. Carothers. Lorain: A step closer to the pdes 'holy grail'. In Proceedings of the 2Nd ACM SIGSIM/PADS Conference on Principles of Advanced Discrete Simulation, SIGSIM-PADS '14, pages 3–14, New York, NY, USA, 2014. ACM.

[**LMS11**] I. Lanese, C.A. Mezzina, A. Schmitt, and J-B. Stefani. Controlling reversibility in Higher-Order Pi. In Proceedings of CONCUR 2011, pages 297-311, volume 6901 of Lecture Notes

[**LMT14**] I. Lanese, C. A. Mezzina, F. Tiezzi: Causal-Consistent Reversibility. Bulletin of the EATCS 114 (2014)

[**LPS08**] Shan Lu, Soyeon Park, Eunsoo Seo, Yuanyuan Zhou: Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. ASPLOS 2008: 329-339

[**LSE15**] J.S. Laursen, U.P. Schultz, and L.-P. Ellekilde, "Automatic error recovery in robot assembly operations using reverse execution," in International Conference on Intelligent Robots and Systems (IROS 2015), IEEE/RSJ, 2015.

[**MAH10**] Henrik Mhe and Andreas Angerer and Alwin Hoffmann and Wolfgang Reif. On reverse-engineering the KUKA Robot Language, 1st International Workshop on Domain-Specific Languages and models for ROBotic systems. 2010.

[**MOJ16**] Markus Schordan, Tomas Oppelstrup, David Jefferson, Peter D. Barnes Jr., Daniel J Quinlan. Automatic Generation of Reversible C++ Code and Its Performance in a Scalable Kinetic Monte-Carlo Application. SIGSIM-PADS 2016: 111-122, 2016.

[**Per13**] Perumalla, K.S.: Introduction to Reversible Computing, Chapman and Hall/CRC, 325 Pages, ISBN 9781439873403, 2013.

[**PP14**] K. S. Perumalla and A. J. Park. Reverse computation for rollback-based fault tolerance in large parallel systems. Cluster Computing, 17(2):303–313, June 2014.

[**PU07**] I.C.C. Phillips and I. Ulidowski. Reversing algebraic process calculi. Journal of Logic and Algebraic Programming, 73:7096, 2007.

[**Pau00**] Gh. Paun. Computing with Membranes, Journal of Computer and System Sciences, 61(1): 108-143, 2000.

[**PVQ15**] A. Pellegrini, R. Vitali, and F. Quaglia. Autonomic state management for optimistic simulation platforms. IEEE Transactions on Parallel and Distributed Systems, 26(6):1560–1569, June 2015.

[**SBS11**] Schultz, Ulrik, Mirko Bordignon, and Kasper Stoy. "Robust and reversible execution of self-reconfiguration sequences." Robotica 29.01 (2011): 35-57.

[**SIP15**] E. Santini, M. Ianni, A. Pellegrini, and F. Quaglia. Hardware-transactional-memory based speculative parallel discrete event simulation of very ne grain models. In 2015 IEEE 22nd International Conference on High Performance Computing (HiPC), pages 145-154, Dec 2015.

[**SJB15**] Reverse Code Generation for Parallel Discrete Event Simulation. Markus Schordan, David Jefferson, Peter Barnes, Tomas Oppelstrup, Daniel Quinlan. In Proceedings of the 7th Conference on Reversible Computation. Jean Krivine and Jean-Bernard Stefani (Eds.): Reversible Computation, LNCS 9138, pp. 95-110, ISBN 978-3-319-20859-6, Springer, 2015.

[**SLE15**] U.P. Schultz, J.S. Laursen, L.-P. Ellekilde, and H.B. Axelsen, "Towards a domain-specific language for reversible assembly sequence," in Reversible Computation, 2015.

[**Sch12**] Schultz, Ulrik. "Towards a general-purpose, reversible language for controlling self-reconfigurable robots." Reversible Computation. Springer Berlin Heidelberg, 2012. 97-111.

[**TPC**] Overview of the TPC-C Benchmark. `http://www.tpc.org/tpcc/detail.asp`

[**Tho14**] Mobile Robot API for Janus. Master thesis defence by Martin Nyborg Thomsen. University of Copenhagen, Denmark, 2014.

[**UDB**] UndoDB. Commercial debugger `http://undo-software.com/products/undodb/`

[**US14**] Undo Software. Increasing software development productivity with reversible debugging, 2014. in Computer Science, pages 297311. Springer, 2011.

[**YAG08**] Tetsuo Yokoyama, Holger Bock Axelsen, and Robert Glck. 2008. Principles of a reversible programming language. In Proceedings of the 5th conference on Computing frontiers (CF '08). ACM, New York, NY, USA, 43-54.