Reversible Execution for Robustness in Embodied AI and Industrial Robots

Department: Software Technology
Editor: Markus Schordan

Reversible Execution for Robustness in Embodied AI and Industrial Robots

I. Lanese
University of Bologna/INRIA

U. P. Schultz
University of Southern Denmark

I. Ulidowski
University of Leicester

***Abstract*—Reversible computation is a computing paradigm where execution can progress backwards as well as in the usual, forward direction. It has found applications in many areas of computer science, such as circuit design, programming languages, simulation, modelling of chemical reactions, debugging and robotics. In this article, we give an overview of reversible computation focusing on its use in robotics. We present an example of programming industrial robots for assembly operations where we combine classical AI planning with reversibility and embodied AI to increase robustness and versatility of industrial robots.**

INTRODUCTION

Reversibility can be defined as the ability of a program or a system to execute in reverse in order to undo the effects of its (forward) computation. Reversibility has interested scientists for many years. Landauer has discovered over 60 years ago that erasing information in computers requires energy and that loss of information, such as erasing a value stored in a variable, during computation is manifested by release of heat [1]. The scientists thought at the time that if we could build logic circuits and, ultimately, hardware that reduces or even avoids completely the need to remove information, then computers would be more energy efficient. Subsequently, Fredkin and Toffoli developed reversible universal logic gates as an alternative to the traditional CMOS technology gates [2]. This meant that, at least in theory, it was possible to design and manufacture reversible computers. There has been a significant amount of research on reversible computers since the discovery of reversible logic gates, culminating in many projects to develop reversible circuits and hardware, but these have not changed the way modern hardware is built yet. Apart from this original motivation for *phys-*

 Published by the IEEE Computer Society

ical reversibility, there are many other reasons for, and benefits of, *logical* reversibility [3]. The latter form of reversibility concerns enhancing systems and software (that run on a physically irreversible hardware) with the ability to undo (or simulate undoing of) computation. There are reversible programming languages such as Janus [4] and there are techniques for reversing traditional imperative programming languages such as C [5]. We have also discovered the basics of how to reverse computation of concurrent programs and systems [6], [7], [8], [9].

The purpose of this article is to introduce the topic of reversible computation by presenting a robotics case study where logical reversibility has made a difference. The case study, and more generally, reversible computation research in Europe were partially supported by COST Action IC1405 on Reversible Computation - Extending Horizons of Computing [10]. We shall touch gently on the theories we have developed and explain how they assisted us in solving practical problems of the case study. We will also indicate how we have adjusted our formal techniques to strengthen a traditional AI planning approach to produce a full working solution.

Our case study is about programming industrial robots performing assembly operations (i.e., building a physical product) in a way that, based on a fixed assembly sequence generated by an AI-based planner, achieves automatic error recovery and even automatic disassembly. Error recovery is achieved by temporarily reversing the direction of execution, effectively undoing recent steps, and then trying again. This approach works well in the physical world of robots because slight imprecisions can cause the robot to get stuck, but partially disassembling the object and trying again can often solve the problem. Taken to an extreme, the entire assembly sequence can be reversed, effectively providing an automatic way to disassemble an object.

We thus demonstrate how a traditional AI-based planning approach is enriched by an underlying reversible execution model that relies on the embodiment of the robot system to provide a robust, probabilistic way of executing the plan. The approach is based on the principles of the Janus reversible programming language [4], where every step of the computation must in itself be reversible, thus ensuring that the program as a whole is reversible. In Janus this means that certain irreversible operations, such as multiplication by zero, are not allowed. Similarly, for the robots, reversible execution can not be applied to intrinsically irreversible steps such as cutting or welding.

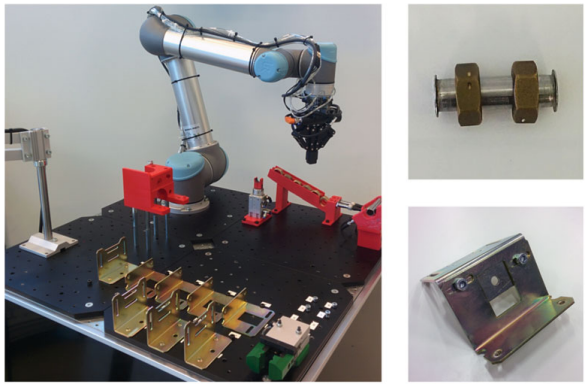

Figure 1. Experimental setup: industrial robot (left) and the two product parts being automatically assembled and disassembled (right)

REVERSIBILITY IN ROBOTICS

Robots act upon the physical world, and depending on the type of robot, may be capable of performing actions that can be considered reversible. Consider the specific case of an industrial robot, i.e., a general-purpose robot arm as depicted in Fig. 1 (left), normally consisting of six or more joints connected in series and programmed using a special-purpose robot programming language. Moving an object from one location to another, or screwing two pieces of metal together using a bolt, could be considered reversible actions. Conversely, breaking an object in two or welding two pieces of metal together would not be considered reversible. If a robot is performing a sequence of operations that can be considered reversible, such as the steps required to assemble a kitchen appliance or a photocopier, then could the entire sequence of operations be perhaps considered a reversible program?

This thought experiment motivated the study and development of *reversible* domain-specific robot programming languages. (Domain-specific languages are special purpose languages designed to solve specific problems such as robot programming [11]). The key insight is that if the robot is

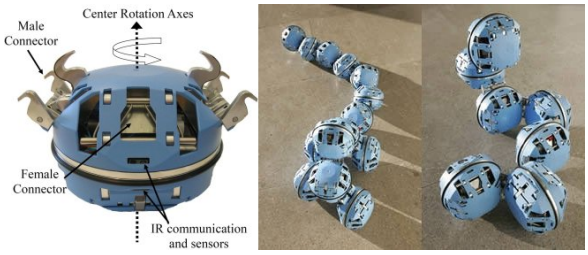

Figure 2. The ATRON modular robot: a single module (left), car/snake configurations (middle), and the snake changing shape (right).

constrained to only perform operations that are physically reversible, then an entire sequence of operations (i.e., a robot program) can be considered physically reversible. Reversible robot programming languages have been studied for industrial robots and for modular self-reconfigurable robots. Industrial robots are the topic of this article. Self-reconfigurable robots are robots that can physically rebuild themselves to take on different shapes [12]. As a concrete example, the ATRON modular robot [13] is shown in Figure 2: each module is an individual robot, and a snake robot composed of multiple modules can rebuild itself into a car robot. A modular robot shaped as a car can for example rebuild itself into a snake to traverse an obstacle, and then afterwards return to the car shape to continue normal operations. This process, referred to as self-reconfiguration, can be considered reversible if each of the steps performed by the individual modules is reversible and can be repeated in reverse order [14].

Industrial Robots, AI, and Reversibility

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. The classical AI-based approach is to derive a plan for the sequence of operations required to complete a given task [15], [16]. Such plans however break down in case of errors, resulting in the need to replan the sequence of operations. Replanning can be costly, in particular in complex operations where errors could occur quite often [17].

We propose to generate plans as *reversible operation sequences*, such that if random failure causes a step to fail, the system can automatically backtrack and retry without replanning. Reverse execution here allows the robot to back out of an erroneous situation, after which the operation can be automatically retried. In a perfectly predictable system retrying would usually result in the same errors, but the embodiment of the robot here works to our advantage: retrying the same physical operation multiple times can produce different results. In effect the AI-generated plan is made robust towards specific kinds of errors, and can be executed robustly without any need for costly dynamic replanning. The combination of automatic retries based on reversibility and probabilistic operations can be considered a form of embodied AI, where reversibly retrying the same operation multiple times results in increased robustness.

As we will see, the combination of AI-based planning and reversibility is amongst others useful for automatic error recovery for small-sized batch production of assembly operations, where precisely specifying error-free operations would be time-consuming and expensive, and dynamic replanning would add a significant overhead to the operation. Moreover, reversibility can in this case be used to automatically derive a disassembly sequence from a given assembly sequence, or vice versa. These capabilities can be achieved using a reversible domain-specific language for specifying assembly sequences.

Robotic assembly and disassembly is done in terms of sequences of operations, such as placement of objects, insertions, screwing operations and so forth. All are challenged by uncertainties from sensors, robot kinematics, and part tolerances. Not all operations are reversible, some are not even repeatable! Many physical phenomena and actions can nevertheless be considered reversible, depending on the abstraction level at which they are observed. For example, an industrial robot that pushes an object to a new position could easily move this object back to its original position, but cannot simply do this by reversing its pushing movements, as pulling requires gripping the object first. Moreover, some operations, such as cutting and welding, are in practice nonreversible. A study of 13 real-world industrial cases showed roughly 76% of the operations to be reversible [18], but many

of the operations require the robot to perform a different physical action to reverse a given action. Taking inspiration from this study, we divide reversible operations into two categories: *directly reversible* operations which simply can be reversed by performing the forwards action in reverse; and *indirectly reversible* operations which can be reversed, but require a different sequence of instructions, which must be manually specified by the programmer.

In the design of our robot programming language we take inspiration from the Janus reversible programming language, where programs are said to be time-invertible [4]. Each computational step in Janus has a specific inverse, and a given program that when executed forwards computes a function, will compute the inverse of this function when executed backwards. Subtracting a constant is for example the inverse of adding a constant. In our robot programming language, each physically reversible operation similarly has an inverse. In the case of directly reversible operations, the inverse is automatically derived by the system. In the case of indirectly reversible operations, the programmer must manually specify the sequence of operations that constitutes the reverse of a given operation. Executing such a manually specified inverse can temporarily bring the system into a *state not normally encountered during forward execution*. Moreover, switching execution direction in the middle of such a manually specified inverse might again take the system into a new state. This contrasts a main property of reversibility in programming languages like Janus and of causal-consistent reversibility [6], [7], a notion used in concurrent systems, which says that any reachable state is forwards reachable. In causal-consistent reversibility, any step of computation of a concurrent system can be undone provided that all its effects, if any, are undone first [9]. Such a property also fails in other contexts, e.g., in some biological systems [19]. Since an operation and its (indirect) reverse are paired, the program has unique starting and ending states, and execution will only terminate in one of these two states. Infinite loops of error correction can manifest, and are handled using a monitoring heuristic that detects if the assembly operation might be stuck.

The programming model we have developed

```
// SCREWING OPERATION
  sequence("insert_screw_operation").
    action(insert_screw).
      reverseWith("remove_screw").nonreversible().
    call("insert_screw_suboperation").
    move(qScrewInside).
    move(qScrewOutside);
```

Figure 3. Sample reversible assembly program: a sequence is defined to consist of an "insert screw" action, a call to another sequence, and two move actions. The "insert screw" action is not itself reversible, and is marked as indirectly reversible using the "remove screw" action.

is based on this abstract semantics-based model extended with various features required for reversible control of industrial robots in real-world scenarios [18]. The actual implementation is in the form of an internal DSL in C++, meaning that a sequence of C++ method calls is used to build a model of the reversible assembly sequence, as shown in Fig. 3. A robot assembly task is programmed as a sequential flow of operations. It is sequential since in practice assembly tasks tend to be a simple sequence of operations (except for error handling, but we aim to automatically handle errors using reverse execution). Reversibility is nevertheless still relevant due to the presence of random behaviour of the physical operations: reversing and re-executing an operation may produce a different result. Each operation (denoted by the keyword "sequence") represents a high-level assembly case logic and is a sequence of primitive instructions. Each instruction is either directly reversible (default), indirectly reversible (indicated by the keyword "reverseWith"), or non-reversible (indicated by "nonreversible").

Our approach was evaluated experimentally using two industrial assembly use-cases [18], Fig. 1 shows the physical robot platform (left) and the two assembled use-cases (right). Both use-cases were used to test the principle of reversible assembly and the use of reverse execution for error correction. For reversible assembly the program was executed forwards to assemble each use-case. Afterwards the finished object was manually placed back into the system, and the program was executed backwards to disassemble the object. This was done multiple times for each use-case with no errors.

The use of reverse execution as an effective error correction tool was experimentally demonstrated by assembling a large number of objects, as follows. The workcell was set to assemble 100 objects of each type consecutively and without pause. During these 200 assemblies a total of 22 errors occurred, of which 18, corresponding to 82%, were automatically resolved and corrected using reverse execution. Errors that were automatically corrected include failed peg-in-hole operations (fixed by backtracking and trying again), dropping a tube (fixed by reversing until a new tube was picked from the feeder), failed to grasp a screw, and screwing failing due to misalignment. Errors that could not be automatically corrected include air-tubing from the gripper getting stuck on the platform, causing the gripper to misalign, and a screw being inserted at a skewed angle causing a bracket to misalign, which could not be corrected as the system had no means of detecting the bracket misalignment.

CONCLUSION

Reversing of computation is conceptually and technically a challenging task even if we only consider logical reversibility. We have illustrated significant potential benefits of reversibility to improve AI-planning in the robotics case study. We have presented briefly some of the recently developed theoretical underpinnings for the case study, concentrating mainly on explaining how reversibility helps. Exploring this application area helped us to exemplify the richness of different forms of reversibility.

While the case study we have discussed is based on sequential reversibility, the notion of causal-consistent reversibility [6], [7] is key to scalable reversible programming of modular robotic systems such as the ATRON robot shown in Figure 2. This is because the modules perform operations in parallel and hence reversing the system must respect dependencies between the actions of individual modules. Provided the ATRON robot does not perform any irreversible steps, we have this strong property: any reachable (by an arbitrary combination of reverse and forward steps) state is forwards reachable. Applying causal consistency to achieve scalable and robust reversible programming for swarming robot systems like the ATRON and Unmanned Aerial Vehicles (UAVs, drones) is considered future work. In the specific case of UAVs, a programming model based on reversibility would hypothetically allow a drone swarm as a whole to "reverse" distributed control decisions, thus easing requirements on reaching consensus (before beginning new operations) and increasing the robustness of the system.

There are also other forms of reversibility suitable for different applications. Probably the best known is *backtracking*, where steps of computation are undone in the inverse order of execution. Apart from many traditional applications of backtracking such as, for example, in search algorithms or logic programming, it has been used to undo concurrent C-like programs for debugging [20], [10].

ACKNOWLEDGMENT

The authors acknowledge partial support from COST Action IC1405 on Reversible Computation - Extending Horizons of Computing. The first author has also been partially supported by French ANR project DCore ANR-18-CE25-0007.

References

1. R. Landauer, "Irreversibility and heat generated in the computing process," *IBM Journal of Research and Development*, vol. 5, 1961.
2. E. Fredkin and T. Toffoli, "Conservative logic," *International Journal of Theoretical Physics*, vol. 21, pp. 219–253, 1982.
3. C. Bennett, "Logical reversibility of computation," *IBM Journal of Research and Development*, vol. 17, 1973.
4. T. Yokoyama, H. B. Axelsen, and R. Glück, "Principles of a reversible programming language," in *Proceedings of CF 2008*. ACM, 2008, pp. 43–54.
5. K. Perumalla, *Introduction to Reversible Computing*. CRC Press, 2014.
6. V. Danos and J. Krivine, "Reversible communicating systems," in *Proceedings of CONCUR 2004*, ser. LNCS, vol. 3170. Springer, 2004, pp. 292–307.
7. I. Phillips and I. Ulidowski, "Reversing algebraic process calculi," *Journal of Logic and Algebraic Programming*, vol. 73, pp. 70–96, 2007.
8. I. Lanese, C. Mezzina, and J.-B. Stefani, "Reversing higher-order pi," in *Proceedings of CONCUR 2010*, ser. LNCS, vol. 6269. Springer, 2010, pp. 478–493.
9. I. Lanese, I. Phillips, and I. Ulidowski, "An axiomatics approach to reversible computation," in *Proceedings of*

FOSSACS 2020, ser. LNCS, vol. 12077. Springer, 2020, pp. 442–461.

10. I. Ulidowski, I. Lanese, U. P. Schultz, and C. Ferreira, Eds., *Reversible Computation: Extending Horizons of Computing - Selected Results of the COST Action IC1405*, ser. LNCS. Springer, 2020, vol. 12070.
11. M. Fowler, *Domain-Specific Languages*. Addison-Wesley, 2010.
12. M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian, "Modular Self-Reconfigurable Robot Systems [Grand Challenges of Robotics]," *IEEE Robotics and Automation Magazine*, vol. 14, no. 1, pp. 43–52, March 2007.
13. M. W. Jorgensen, E. H. Ostergaard, and H. H. Lund, "Modular ATRON: modules for a self-reconfigurable robot," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004, pp. 2068–2073.
14. U. Schultz, M. Bordignon, and K. Støy, "Robust and reversible execution of self-reconfiguration sequences," *Robotica*, vol. 29, pp. 35–57, 2011.
15. R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," in *Proceedings of the 2nd International Joint Conference on Artificial Intelligence (IJCAI' 71)*. Kaufmann Publishers Inc., 1971, pp. 608–620.
16. C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Backward-forward search for manipulation planning," in *Proceedings of he IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 6366–6373.
17. P. Loborg, "Error recovery supporting manufacturing control systems," Ph.D. dissertation, School of Engineering at Linkoping University, 1994.
18. J. Laursen, L. Ellekilde, and U. Schultz, "Modelling reversible execution of robotic assembly," *Robotica*, vol. 36, no. 5, pp. 625–654, 2018.
19. I. Phillips, I. Ulidowski, and S. Yuen, "A reversible process calculus and the modelling of the ERK signalling pathway," in *Proceedings of Reversible Computation 2012*, ser. LNCS, vol. 7581. Springer, 2013, pp. 218–232.
20. J. Hoey and I. Ulidowski, "Reversing imperative parallel programs and debugging," in *Proceedings of Reversible Computation 2019*, ser. LNCS, vol. 11497. Springer, 2019, pp. 108–127.

Ivan Lanese is an Associate Professor at the University of Bologna, Italy. He acted as the vice chair of the COST Action IC1405 on Reversible Computation. He is interested in formal methods and concurrency theory, with special focus on reversibility and reversible debugging. Further information is available at https://www.unibo.it/sitoweb/ivan.lanese/en. He can be contacted at ivan.lanese@gmail.com.

Ulrik P. Schultz is a Professor in Aerial Robotics at the University of Southern Denmark. He recently participated in the COST Action IC1405 on Reversible Computing where he chaired the Working Group on Applications. He is interested in programming languages for robotics, his full biography is available at http://www.sdu.dk/staff/ups and he can be contacted at ups@sdu.dk

Irek Ulidowski is an Associate Professor at the University of Leicester in England. He chaired the COST Action IC1405 on Reversible Computation. His interests include reversible computation and its application, and concurrent systems. Further information is available at https://www.cs.le.ac.uk/people/iu3. He can be contacted at irekulidowski@gmail.com.