SDU
University of Southern Denmark

Reversible control of robots

Schultz, Ulrik Pagh

Published in:
Reversible Computation

DOI:
10.1007/978-3-030-47361-7_8

Publication date:
2020

Document version:
Final published version

Document license:
CC BY

Citation for pulished version (APA):
Schultz, U. P. (2020). Reversible control of robots. In I. Ulidowski, I. Lanese, U. P. Schultz, & C. Ferreira (Eds.), *Reversible Computation: Extending Horizons of Computing - Selected Results of the COST Action IC1405* (pp. 177-186). Springer. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) Vol. 12070 LNCS https://doi.org/10.1007/978-3-030-47361-7_8

Go to publication entry in University of Southern Denmark's Research Portal

Terms of use
This work is brought to you by the University of Southern Denmark.
Unless otherwise specified it has been shared according to the terms for self-archiving.
If no other license is stated, these terms apply:

- You may download this work for personal use only.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying this open access version

If you believe that this document breaches copyright please contact us providing details and we will investigate your claim.
Please direct all enquiries to puresupport@bib.sdu.dk

Download date: 22. May. 2022

Reversible Control of Robots

Ulrik Pagh Schultz(✉)

SDU UAS, MMMI, University of Southern Denmark, Odense, Denmark
ups@mmmi.sdu.dk

Abstract. 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. Moreover reverse execution permits automatically deriving programs that physically reverse the operations of an industrial robot. This can be useful in industrial assembly, where a disassembly program can be automatically derived from the assembly program.

In this case study we investigate robotic assembly from the point of view of reversibility, investigating to what extent program inversion of a robotic assembly sequence for a given product can be considered to derive a robotic disassembly sequence for this same product, and investigating to what extent changing the execution direction at runtime (i.e., backtracking and retrying) using program inversion can be used as an automatic error handling procedure. The programming model used to reversibly control industrial robots is based on an abstract semantics-based model, extended with various features required for reversible control of industrial robots in real-world scenarios, and implemented as a domain-specific programming language.

1 Introduction

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 industrial assembly process [5,6]. 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 [10]. The useful application of reversibility to this type of robot is however only observed for self-reconfiguration operations, significantly limiting the notion of

The author acknowledges partial support of COST Action IC1405 on Reversible Computation - Extending Horizons of Computing.

© The Author(s) 2020
I. Ulidowski et al. (Eds.): RC 2020, LNCS 12070, pp. 177–186, 2020.
https://doi.org/10.1007/978-3-030-47361-7_8

reversibility and real-world interaction that can be studied using this type of robot. To better understand the underlying relation between reversible computation and physical reversibility, we in this case study investigate reversible control of industrial 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 [11]. 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 language (DSL) for specifying such assembly sequences [5,11]. 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.

The result of this case study is significant progress in the area of reversibility for industrial robots [4]. Key developments include an improved understanding of the interaction between reversible computing and real-world systems that only are partially reversible, as well as a substantial experimental evaluation of the use of reversible languages to control industrial robots performing assembly and disassembly in the context of small-batch production. Overall this work experimentally demonstrates the use of reversible computing to improve system reliability.

2 Related Work

Reversibility has previously been investigated for self-reconfigurable robots. Self-reconfigurable, modular robots are distributed robotic devices that can autonomously change their physical shape [13]. 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 can bring the robot back to its initial shape, as has been experimentally demonstrated using the DynaRole reversible language [10]. 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 behaviours. Initial ideas on generalising the DynaRole language to support a wider range of modular robot control scenarios retain the possibility of reversing distributed sequences [8,9], but have neither been formalised nor experimentally demonstrated.

Large-scale modular robotic systems can be considered as intensive parallel systems [7]. Reversibility for intensive parallel systems was studied by Agrigoroaiei and Ciobanu [1]. Here, the process of reversing is presented as a form of duality (a notion from category theory). A related approach presenting reversibility for the bio-inspired formalism of membrane systems is given by the same authors [2].

Partial reversibility has been studied for reversible programming languages [12] using logging of program state to handle irreversible operations. This approach would in our case correspond to recording the motions of the robot and replaying them in reverse, which is applicable to any operation but does not normally serve to reverse actions in the real world. Rather, our approach relies on the programmer explicitly writing reverse code that, through a different sequence of operations, brings the system back to a previous state. This approach can be compared to causal-consistent reversibility [3] in the sense that the observable events (i.e., the state of the system the robot is working on) is reversed in a consistent way; unlike causal-consistent reversibility we however require the programmer to manually implement the basic reverse operations using the notion of indirect reversibility.

3 Reversible Assembly Tasks

We investigate robotic assembly tasks from the point of view of reversibility, investigating to what extent program inversion of a robotic assembly sequence for a given product can be considered to derive a robotic disassembly sequence for this same product, and investigating to what extent changing the execution direction at runtime (i.e., backtracking and retrying) using program inversion can be used as an automatic error handling procedure [4].

3.1 Robotics, Assembly, and Reversibility

Robotic assembly and disassembly is done in terms of sequences of operations such as precise placement of objects, insertions with tight fits, screwing operations and so forth. All are challenged by uncertainties from sensors, robot kinematics and part tolerances; not all are reversible, some are not even repeatable. Our approach has been tested with a standard robotic platform based on a Universal Robots UR5, shown in Fig. 1 together with the two industrial assembly cases used to evaluate the approach [4].

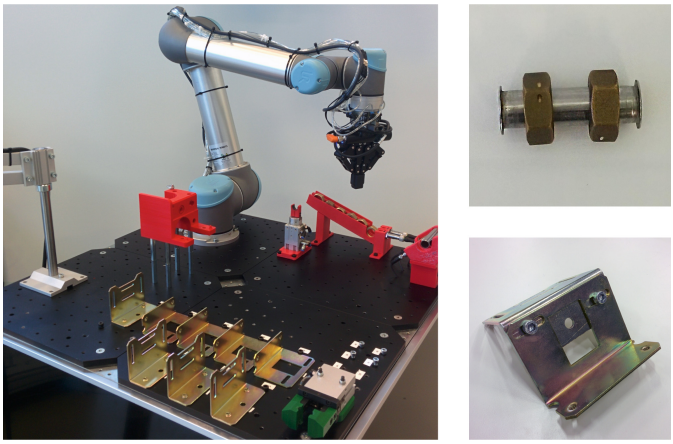

Fig. 1. The experimental platform and two assembly test cases (from [4]).

3.2 Reversibility

Many physical phenomena and actions are in principle reversible, although this reversibility may depend 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 movements as pulling requires gripping the object first. Moreover, some operations, such as cutting, should in general be considered nonreversible. A study of 13 real-world industrial cases showed roughly 76% of the operations to be reversible [4], but many of the operations require the robot to perform different physical actions to reverse a given action. Based on this observation, we can divide the reversible operations into two categories: *directly reversible* and *indirectly reversible* operations. Operations which can be reversed through program inversion are considered directly reversible. Indirectly reversible operations on the other hand can be reversed, but require a different sequence of instructions.

3.3 Repeatability

Unlike Janus-style reversible computing, where programs can be said to be time-invertible [14], with robotics physical changes made to the environment from the execution influences the repeatability of operations. Operations that can be done again and again can be referred to as *fully repeatable*. Other actions can only be done a limited number of times, e.g., due to wear and tear, and are said to be *partially-repeatable*. Last, *nonrepeatable* operations are those that cannot be retried.

3.4 Reversibility and Repeatability

Considering reversibility and repeatability together leads to a classification of robotic assembly operations [4]. Operations that are fully repeatable and directly reversible can be automatically managed using a program inversion approach, whereas indirectly reversible operations require explicit reverse code to be provided by a programmer, partially repeatable operations limit how many times a program can be reversed, and certain operations are fundamentally irreversible and thus mark points across which the program cannot be reversed.

4 Programming Model

The programming model developed in our case study is based on an abstract semantics-based model [11] extended with various features required for reversible control of industrial robots in real-world scenarios [4].

4.1 Basic Model

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 relevant due to the presence of random behaviour of the physical operations: reversing and re-executing an operation may produce a different results. Each operation represents high-level assembly case logic and is a sequence of instructions. *Instructions* are either reversible, providing a two-way reversible forward/backward mapping of hardware instructions, or non-reversible, providing a single-directional mapping. Instructions are implemented using traditional nonreversible programming. Taking inspiration from Janus [14], it is possible to both call and uncall operations, the latter causing the operation to be interpreted in reverse.

The programming model used to represent robot assembly tasks is built on the following principles. (1) Instructions always map the robot system from a known state to a known state, but may have different semantics for forward and reverse. (2) Indirect reversibility is achieved by modelling instruction sequences that are different for forwards and reverse execution using the principle of overridden reverse flow, where users can write different code for forwards and backwards execution. (3) Instructions can be marked as nonreversible. A directed graph is used to model the underlying reversible assembly sequence. In this graph each node corresponds to a primitive instruction which is executable on the physical platform. Furthermore, each node contains pointers to the next forward instruction and the next reverse instruction (if any). Overall the graph is evaluated through forward/backwards interpretation and each instruction is evaluated using instruction inversion in the sense that different semantics are applied for forward and backwards execution.

```
operation attach_nut_bolt {
  state begin_nut_bolt (...tool pos...) bolt:(...pos...) nut:(...pos...)
  moveto (...pos above table...)
  pickup (nut, fixed_gripper, (...pos of nut...))
  moveto (...)
  ...
}
operation apply_and_turn_nut { ...commands... }
reverse { ...commands that undo apply_and_turn_nut...  }
```

Fig. 2. Sample RASQ program, vector constants are omitted for clarity (adapted from [11]).

4.2 Implementation

The basic model provides the foundation for programming realistic assembly cases [4]. The principle of indirect reversibility is in practice instantiated in many different ways, such as movement or error detection instructions that only activate in one execution direction. Error handling is implemented in the interpreter: when an error is detected during forwards execution the direction is immediately reversed for a number of steps, after which forwards execution is again resumed. The same model is applied for execution in reverse, and even applies recursively, i.e., if an error is detected during reverse execution triggered due to an error. Each instruction carries specific information describing how to handle switching of execution direction, specifically whether the instruction should be repeated in reverse or not when switching direction due to the instruction failing. A simple error handling strategy that changes execution direction for a random number of steps and that ensures termination by bounding the total number of steps was observed to work well in practice.

4.3 Language

The idea of reversible control of industrial robots was initially presented using a high-level programming language [11]. An example is shown in Fig. 2. The program declares two operations, `attach_nut_bolt` and `apply_and_turn_nut`. The operation `attach_nut_bolt` only specifies a single (forwards) body for both forwards and reverse execution, so reverse execution will inversely evaluate the forwards body in reverse order. The first statement is a state assertion, named `begin_nut_bolt`, specifying the spatial positioning of the tool and the respective positions of the bolt and nut objects. The next statement of the program is a move, which moves the robot to the given position (again, the position is given as a constant, not shown). After the move follows a pick up instruction that causes the pickup operation associated with the name `fixed_gripper` and the object `nut` to be evaluated. Last follows the declaration of the second operation `apply_and_turn_nut`, which is not shown in detail, but has both a forwards and a reverse body, so forwards execution evaluates the forwards body in forwards

```
operation("screwdriver_activate").
  io(screwdriver, Switch::on).
  wait(0.3).
  wait(screwingFinished).
  reverseWith("screwing_finished_backwards");
  io(screwdriver, Switch::off).
  io(screwdriverBackwards, Switch::off);
```

Fig. 3. Sample SCP-RASQ program (adapted from [4]).

order, and reverse execution evaluates the reverse body in forwards order (i.e., in the order written in the program).

In practice it turned out to be more useful to rely on an internal DSL implemented in C++, using a model-driven approach that serialises the program to an XML structure that can subsequently be instantiated as the graph structure used by the reversible interpreter. This internal DSL, named SCP-RASQ for "Simple C++ RASQ", is exemplified in Fig. 3. This program declares an operation that performs IO operations to communicate with the screwdriver, and shows how indirect reversibility can be programmed in-place using the `reverseWith` declaration.

5 Results

This section will give an overview of the experimental results demonstrated in earlier work on several industrial use cases [4].

5.1 Methodology

Error recovery using reverse execution was tested using two industrial assembly tasks use-cases; the physical robot platform and the assembled products are shown in Fig. 1. An SCP-RASQ program was created for each of the use cases. Both cases include a final step where the finished product is discarded into a box. This step was not performed when running the programs backwards, as it is a nonreversible task since our current setup cannot bin-pick the part out again.

5.2 Experiment 1: Reversing the Programs

Both use-cases were used to test the principle of reversible assembly. Forward execution performs assembly while reverse execution performs disassembly. For each case the program is executed forward to assemble an object. Afterwards the finished objects is then manually placed back into the system, and the program is then executed backwards to disassemble the object. This was done a total of three times for each case, with no errors.

In our test programs directly reversible operations made up 45% of all operations. Moreover, directly reversible operations such as the "pick screwdriver"

were used in both their forward and backwards form in the same program using the call and uncall functionality. Both use-cases could be made almost entirely reversible using either directly or indirectly reversible operations through the execution model and the programming language. We believe that if the reversibility concept was to be integrated more deeply into the design of assembly processes and external equipment such as feeders, an even greater degree of directly reversible instructions could be achieved.

5.3 Experiment 2: Assembling 100 Objects

By assembling a large number of objects the use of reverse execution as an effective error correction tool was demonstrated. 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.

This experiment shows that reverse execution is capable of solving a wide variety of errors and that the exact method for solving each kind of error need not always be the same, as backtracking was done randomly at different lengths and sometimes resulted in different solutions to the same problem. Moreover we see that the backtracking system is promising in handling errors related to small uncertainties in the assembly tasks, but that errors resulting in larger and mechanical failures still need to be addressed either in the design phase or by some other error handling mechanism. Last, the experiments also show that while reverse execution can be used for solving a wide variety of errors, it also places strong demands on the error detection system.

6 Conclusion

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 small-batch assembly, and moreover facilitate the programming of such operations. In this case study we have introduced a programming model which enables robot assembly programs to be executed in reverse. We have experimentally demonstrated that temporarily switching the direction of program execution can be an efficient error recovery mechanism. Moreover, we have shown that additional benefits arise from supporting

reversibility in our robotic assembly language, namely increased code reuse and automatically derived disassembly sequences.

This case study has resulted in an improved understanding of the interaction between reversible computing and real-world systems that only are partially reversible, as well as a substantial experimental evaluation of the use of reversible programming languages to control industrial robots performing assembly and disassembly in the context of small-batch production. Overall this case study has experimentally demonstrated the use of reversible computing to improve system reliability.

Acknowledgements. Thanks to Gabriel Ciobanu for help in describing the related work on reversibility of massively parallel systems.

References

1. Agrigoroaiei, O., Ciobanu, G.: Dual P systems. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2008. LNCS, vol. 5391, pp. 95–107. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-540-95885-7_7
2. Agrigoroaiei, O., Ciobanu, G.: Reversing computation in membrane systems. J. Logic Algebraic Program. **79**(3), 278–288 (2010)
3. Lanese, I., Mezzina, C.A., Tiezzi, F.: Causal-consistent reversibility. Bull. EATCS **114** (2014)
4. Laursen, J., Ellekilde, L., Schultz, U.: Modelling reversible execution of robotic assembly. Robotica **36**(5), 625–654 (2018)
5. Laursen, J.S., Schultz, U.P., Ellekilde, L.P.: Automatic error recovery in robot assembly operations using reverse execution. In: Evers, C., Sheaffer, J., Tourbabin, V., Naylor, P.A., Romanoni, A., Matteucci, M. (eds.) International Conference on Intelligent Robots and Systems (IROS 2015). IEEE/RSJ (2015)
6. Mühe, H., Angerer, A., Hoffmann, A., Reif, W.: On reverse-engineering the KUKA robot language. In: Schultz, U.P., Stinckwich, S., Ziane, M. (eds.) Proceedings of the First International Workshop on Domain-Specific Languages for Robotic Systems (DSLRob 2010) (2010). arXiv:1009.5004 [cs.RO]
7. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002). https://doi.org/10.1007/978-3-642-56196-2
8. Schultz, U.P.: Using scheme to control simulated modular robots. In: Danvy, O. (ed.) Proceedings of the 2012 Annual Workshop on Scheme and Functional Programming, pp. 90–95. ACM (2012)
9. Schultz, U.P.: Towards a general-purpose, reversible language for controlling self-reconfigurable robots. In: Glück, R., Yokoyama, T. (eds.) RC 2012. LNCS, vol. 7581, pp. 97–111. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36315-3_8
10. Schultz, U., Bordignon, M., Støy, K.: Robust and reversible execution of self-reconfiguration sequences. Robotica **29**, 35–57 (2011)
11. Schultz, U.P., Laursen, J.S., Ellekilde, L.-P., Axelsen, H.B.: Towards a domain-specific language for reversible assembly sequences. In: Krivine, J., Stefani, J.-B. (eds.) RC 2015. LNCS, vol. 9138, pp. 111–126. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20860-2_7

12. Tyagi, N., Lynch, J., Demaine, E.D.: Toward an energy efficient language and compiler for (partially) reversible algorithms. In: Devitt, S., Lanese, I. (eds.) RC 2016. LNCS, vol. 9720, pp. 121–136. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40578-0_8
13. Yim, M., et al.: Modular self-reconfigurable robot systems [grand challenges of robotics]. IEEE Robot. Autom. Mag. **14**(1), 43–52 (2007)
14. Yokoyama, T., Axelsen, H.B., Glück, R.: Principles of a reversible programming language. In: Proceedings of the 5th Conference on Computing Frontiers (CF 2008), pp. 43–54. ACM (2008)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.