ISSN 2186-7437

NII Shonan Meeting Report

No. 2017-9

Reverse Execution in Testing – Improving Security and Reliability

Michael Kirkedal Thomsen
Kazutaka Matsuda
Mohammad Reza Mousavi

July 03–06, 2017

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-Ku, Tokyo, Japan

Reverse Execution in Testing – Improving Security and Reliability

Organizers:

Michael Kirkedal Thomsen
(University of Copenhagen, Denmark)

Kazutaka Matsuda
(Tohoku University, Japan)

Mohammad Reza Mousavi
(Halmstad University, Sweden and University of Leicester, UK)

July 03–06, 2017

Overview

Implementing programs without errors are important for reliably and security of software. Many approaches to this have over the years been developed, exemplified by numerous testing techniques, formal verification, and static program analysis.These techniques can be used identify problematic parts of programs or to some degree verify the correctness of them. However, achieving any guarantees is often impossible or (at the least) very cumbersome, which to some extend is due to the complexity of the conventional deterministic computation model. So using a more restricted computation model has the potential to improve this.

A way to achieve this is to use a computation model that has a notion of reverse execution. Here, the ability of compute from any reached state back to any previous state is the restriction over a deterministic model; but at the same time the restriction gives the possibility for better reasoning within the model. There exist several different notions of which one can achieve reverse execution. Two of the Reversible Computations are often designed to be on completely information lossless and information generating. The initial studies can be dated back to the years around 1960 with three different studies that based on quite different computation models and motivations: Huffman studied information lossless finite state machines for their applications to data transmission, Landauer came to study reversible logic in his quest to determine the sources of energy dissipation in a computing system, and Lecerf studied reversible Turing machines for their theoretical properties. The field is often motivated by Landauers work that based on a gedankenexperiment found that, theoretically, the energy consumption of computations is not correlated with how much information is processed, but only with the amount of information that is lost during the computations. A result that have since been experimentally verified. However, modern research in the area have a larger focus on applications where the information preservation has an advantage.

Bidirectional Transformations are transformations, or more generally mechanisms, that keep the consistency of two or more data. Originally, bidirectional transformation was studied mainly around 80s in the database community as the view-update problem, in which one transformation is a usual query and the other transformation is translation of updates on a view to updates on the original table. Recently, the problem draws our attention again, influenced by the rise of programming languages and techniques for bidirectional transformations (such as lenses), and the needs in software development processes being complex, especially in model-driven software development.

Testing meets Reversible Computations and Bidirectional Transformations

While both reversible computations and bidirectional transformations are small but growing research fields, testing is large and well established. No one will doubt that testing is immensely important to reliability of software and new techniques are often presented; from finite state-machine based models to recent years frameworks for automatic randomized test generation.

It is interesting that all three fields share the possibility to improve reliability of programs, though there is a difference in the foundations behind. In comparison testing bidirectional transformations (BX) and reversible computations (RC) takes a much different approach. In these models specific properties are intended to be upheld by design. For bidirectional transformations we want a guarantee that updates between two different copies of the same information are propagated correctly, while the essence of reversible computation is to preserve information altogether. It has been recognised that there are similarities between BX and RC, though the two communities have little overlap. This is why it is important to increased cross-collaboration that can result in interesting new research. This both researching test algorithms for reversible and bidirectional programming models, and exploiting the structures of backward execution models in testing subsets programs without backwards execution.

Overview of Tutorials

An introduction to BiGUL, a minimalist putback-based bidirectional programming language

Hsiang-Shang Ko, National Institute of Informatics, Japan

Putback-based bidirectional programming allows the programmer to write only one putback transformation, from which the unique corresponding forward transformation is derived for free. BiGUL, short for the Bidirectional Generic Update Language, is designed to be a minimalist putback-based bidirectional programming language. BiGUL was originally developed in the dependently typed programming language Agda, and its well-behavedness has been completely formally verified; this package is the Haskell port of BiGUL.

Overview of Technical Presentations

Semantic theories for communicating transactions which have automatic reversibility built-in

Matthew Hennessy, Trinity College Dublin, Ireland

We develop a theory of bisimulations for a simple language containing communicating transactions, obtained by dropping the isolation requirement of standard transactions. Such constructs have emerged as a useful programming abstraction for distributed systems.

In systems with communicating transactions actions are tentative, waiting for certain transactions to commit before they become permanent. Our theory captures this by making bisimulations history-dependent, in that actions performed by transactions need to be recorded. The main requirement on bisimulations is the systems being compared need to match up exactly in the permanent actions but only those.

The resulting theory is fully abstract with respect to a natural contextual equivalence and, as we show in examples, provides an effective verification technique for comparing systems with communicating transactions.

Multiple Mutation in Model-Based Testing

Alexandre Petrenko, Centre de recherche informatique de Montreal (CRIM), Canada

Model-based testing is focused on testing techniques which rely on the use of models. The diversity of systems and software to be tested implies the need for research on a variety of models and methods for test automation. We briefly review this research area and introduce several papers selected from the 22nd International Conference on Testing Software and Systems (ICTSS).

Using reverse test executions in robust testing of hybrid systems

Georgios Fainekos, GRASP Laboratory, University of Pennsylvania, Philadelphia, PA, USA

This talk deals with the robust Metric Temporal Logic (MTL) testing and verification of linear systems with parametric uncertainties. This is a very general class of systems that includes not only Linear Time Invariant (LTI) systems with unknown constant parameters, but also Linear Time Varying (LTV) systems and certain classes of nonlinear systems through abstraction. The two main tools for the solution of this problem are the approximate bisimulation relations and a notion of robustness for temporal logic formulas.

Programming Lenseswithout Combinators

Kazutaka Matsuda, Tohoku University, Japan

Combinator-based frameworks and languages such as the lens framework are common in bidirectional programming, in which we compose smaller bidirectional transformations by combinators that preserve "bidirectionality" of transformations. However, in such combinator-based frameworks or languages, users must write their bidirectional transformations by stylized combinators in the point-free style, which limits the scalability of bidirectoinal programming.

To remedy this situation, we develop a new bidirectional programming language, in which bidirectional transformations are represented as ordinary functions, and combinators are mapped to language constructs with binders. This enables us to write bidirectional transformations in an ordinary functional style, and at the same time access the power of exiting combinator-based framework (e.g., lenses) including bidirectionality guarantee.

Reversible semantics for Erlang or Term rewriting

German Vidal, Universitat Politcnica de Valncia, Spain

In a reversible language, any forward computation can be undone by a finite sequence of backward steps. Reversible computing has been studied in the context of different programming languages and formalisms, where it has been used for debugging and for enforcing fault-tolerance, among others. In this paper, we consider a subset of Erlang, a concurrent language based on the actor model. We formally introduce a reversible semantics for this language. To the best of our knowledge, this is the first attempt to define a reversible semantics for Erlang.

Proving decidability of equational theories by second-order rewriting

Makoto Hamana, Gunma University, Japan

We present a general methodology of proving decidability of equational theory of programming language concepts in the framework of second-order algebraic theories of Fiore, Hur and Mahmoud. We propose a Haskell-based analysis tool SOL, Second-Order Laboratory, which assists the proofs of confluence and strong normalisation of computation rules derived from second-order algebraic theories.

To cover various examples in programming language theory, we combine and extend both syntactical and semantical results of second-order computation in non-trivial manner. In particular, our choice of Yokoyama's deterministic second-order patters as a syntactic construct of rules is important to cover a wide range of examples, such as Hasegawa's linear lambda-calculus. We demonstrate how to prove decidability of various algebraic theories in the literature. It includes the equational theories of monad and computational lambda-calculi, Staton's theory of reading and writing bits, Plotkin and Power's theory of states, and Stark's theory of pi-calculus.

Identified Open Problems

After having listened to the tutorials, we held a brainstorming session and identified the following problems to be discussed and concretized in the remaining 2 days of the workshop:

- Learning and testing as bidirectional transformations,
- Reversible embedding as complement in bidirectional transformation,
- Exploiting reversibility to guide test-case and test-data generation,
- Landauer and Bennett embeddings for non-deterministic and concurrent systems semantics,
- Reversibility for the semantics of timed systems, and
- Reversibility for the semantics stochastic and probabilistic programs.

Selected Open Problems

Learning and testing as bidirectional transformations

Participants: Alexandre Petrenko, Mohammad Mousavi, Mahsa Varshosaz, Kazutaka Matsuda, Matthew Hennessy, João Saravia, Michael Johnson, German Vidal, Makoto Hamana

The idea behind this problem came during the initial presentations of the three topics. The basic idea is that the feedback-loop that is the foundation of

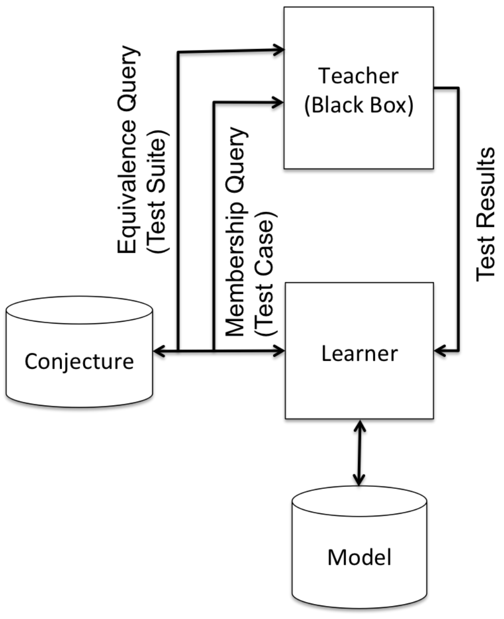

Figure 1: Schematic View of Model Learning

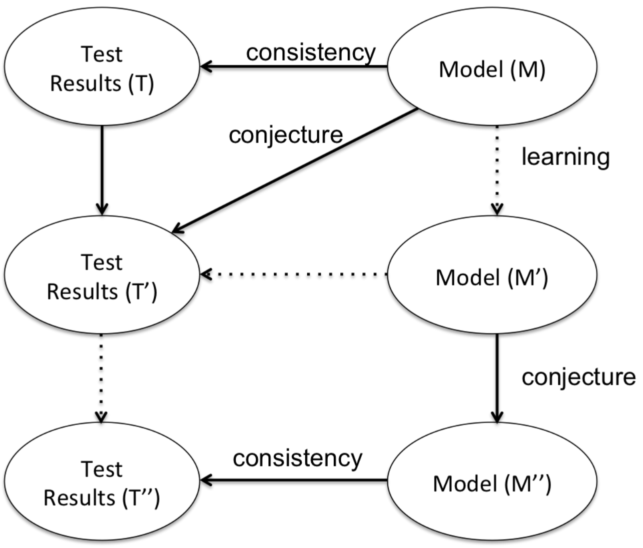

Figure 2: Learning as Bidirectional Transformations

learning systems have many similarities to the updates made in bidirectional transformations. In a leaning system the testing model is gradually changed based on tests that are performed on a real system; please see Figure 1.

We consider the process of model learning as a set of bidirectional transformations, where new observations/test on the real system must be mapped back to the conjectured model, see Figure 2.

It is important to note that such a system does uphold the basic laws from bidirectional transformations; i.e., the get-put, put-get, and put-put laws. In particular, the put-put law seems to have interesting impacts

It is at the moment unclear in which direction this work will go. However, it was agreed that it holds promising ideas.

Reversible embedding as complement in bidirectional transformation

Participants: Robin Kaarsgaard, Kanae Tsushima, Hsiang-Shang Ko, Michael Kirkedal Thomsen

It is recognised that there exist an overlap between bidirectional transformations and reversible computations. There is, however, little work on investigating the extent of this overlap and finding a unified model. Finding this was the goal of this open problem. The work took hold of the view/update problem from BX, which have two fundamental operations: get and put. get is a function that given a database will return a view, while update is a function that given a (possibly updated) view and a database will return the (possible) updated database. There exist variations to this where the update function is also given the specific updates that are made on the view. It is clear that both operations are not reversible.

Embedding $\mathsf{get} :: S \to V$ (read function get of type from Source returning a View) into a reversible function is straightforward with a copying semantics that returns both the generated view and the initial database. This is actually the semantics of the normal version of get as it is the intention to preserve the database for later use. In other words we will make the Bennett embedding with the semantics

$$\mathsf{get}_r :: S_i \to S_i \times V.$$

Note that we have subscripted the types with labels to note what is initial (i) and what is updated (u). Here everything is initial.

Embedding $\mathsf{put} :: S \times V \to S$ is less obvious. First approach was to return both initial and updated database. This is however much too large and will over time create a trace of all databases. Instead the idea is make the Landauer embedding, such that changes conceptually are stored in a change-log,

$$\mathsf{put}_r :: S_i \times V \to S_u \times \Delta S.$$

This ensures that only the necessary information is stored and creates a link to modern computer systems, which we will explore a bit later. With respect to bidirectional transformations, it is also worth to note that with reversibility, the get function can be directly generated from the put_r function as the inverse of a put_r that generated a empty change-log.

To ease the implementation of the, we expect that it will be advantageous to take the approach from edit-based bidirectional transformations. Here the idea is that you do the updates based on some trace of edits that is performed and *not* just the updated view. Thus we have that $\mathsf{put} :: S \times V \times \Delta V \to S$ and our reversible put will therefore be

$$\mathsf{put}_r :: S_i \times V \times \Delta V \to S_u \times \Delta S.$$

The task of put_r is therefore the following. First, uncompute V by inverse calling get_r. Second, migrate the view update to the source, thus transforming ΔV into ΔS. Finally, update the initial source using the ΔS

It is it now time to return to the comment on modern computing systems. In many cases reliability and accountability is an important or even required property and storing a log or trace of changes is important. A huge advantage of having this implemented in a reversible language is that it is guaranteed that the log will be complete; i.e. in a reversible model you cannot loose information, so the log needs to contain all needed information to added from a previous state.

The idea of implementing this will be based on the edit-base reversible semantics from before. To exemplify this on a transactional database, we have that put is divided into an update and commit function; i.e. a part that prepares and requests an update to the database and a part that finalises or declines the update.

For this, update will be given the information from the user, which is the view and updates made to it, and from this to should generate what is needed to update the source:

$$\mathsf{update}_r :: S_i \times V \times \Delta V \to S_i \times \Delta S.$$

Note that we use the database, but it is read only; nothing in it has been changed. This function is the exact same as the two first steps of the edit-based put_r function.

The commit is a more tricky as it can have two results. If the commit is accepted (it does not violate database consistency) it will return the updated database and the changes made. If the commit is rejected it should do no changes. Thus we have

$$\mathsf{commit}_r :: S_i \times \Delta VS \rightarrow (S_u \times \Delta S) + (S_i \times \Delta S).$$

It the case of a rejected commit the view and view updates can be computed by calling the inverse of update_r and these information could be returned to the user that then should give a consistent update.

The participants agree that the initial work is promising and it is expected to continue after the seminar. The combination of bidirectional transformations and reversible computations gives the update guarantees and information preservation, which is important for the target applications.

Reversibility for the semantics stochastic and probabilistic programs and Reversibility for the semantics of timed systems

Participants: Shoji Yuen, Georgios Fainekos, Kazunori Ueda

Combining reversibility and stochastic/probabilistic computations models looks hard at a first glans as the models seemingly opposing properties. These discussion did find approaches to a definition of a semantics, but it did not result in anything concrete. It is at the current moment unclear if it will lead to further research.

List of Participants

- Makoto Hamana, Gunma University, Japan
- Robin Kaarsgaard, University of Copenhagen, Denmark
- Mohammad Mousavi, Halmstad University, Sweden and University of Leicester, UK
- Kanae Tsushima, National Institute of Informatics, Japan
- German Vidal, Universitat Politcnica de Valncia, Spain
- Shoji Yuen, University of Nagoya, Japan
- Michael Johnson, Macquarie University, Australia
- Kazutaka Matsuda, Tohoku University, Japan
- Alexandre Petrenko, Centre de recherche informatique de Montreal (CRIM), Canada
- Kazunori Ueda, Waseda University, Japan
- Mahsa Varshosaz, Halmstad University, Sweden
- Hsiang-Shang Ko, National Institute of Informatics, Japan
- Georgios Fainekos, Arizona State University, USA
- Matthew Hennessy, Trinity College Dublin, Ireland
- Michael Kirkedal Thomsen, University of Copenhagen, Denmark
- João Saravia, Universidade do Minho, Portugal

Meeting Schedule

Check-in Day: July 02 (Sun)

- Welcome Banquet

Day1: July 03 (Mon)

- Introduction
- Division into interdisciplinary groups
- Presentation in groups and round, identification of topics and possible open questions
- Preparation of introductions to the three disciplinarians
- Technical talks

Day2: July 04 (Tue)

- Presentation of the three disciplinarians
- Identification of open problems and division into groups
- Discussions of open problems
- Technical talks

Day3: July 05 (Wed)

- Continued discussions of open problems
- Group Photo Shooting
- Excursion and Main Banquet

Day4: July 06 (Thu)

- Presentation of progress on open problems and discussions in plenum
- Wrap up