Supporting Interactive System Testing with Interaction Sequences

Jessica Turner
Department of Computer Science, University of Waikato
Hamilton, New Zealand
jdt11@students.waikato.ac.nz

Judy Bowen
Department of Computer Science, University of Waikato
Hamilton, New Zealand
jbowen@waikato.ac.nz

Steve Reeves
Department of Computer Science, University of Waikato
Hamilton, New Zealand
stever@waikato.ac.nz

EICS '17, June 26-29, 2017, Lisbon, Portugal.

ACM ISBN 978-1-4503-5083-9/17/06 $15.00.
http://dx.doi.org/10.1145/3102113.3102149

Abstract
In software engineering testing is an important part of the development process. In interactive systems human input introduces the possibility of human error, which increases the testing requirements. In safety-critical systems user or system error can result in injury or death to a user. We propose using interaction sequences as a way of supporting interactive system testing to help address these issues.

Author Keywords
Formal methods; Abstraction modelling and modularity; Regular languages; interaction devices

ACM Classification Keywords
D.2.4 [Software Program/Verification]: Formal methods; F.1.1 [Models of Computation]: Automata (e.g. finite); D.4.7 [Organization and Design]: Interactive systems

Introduction
In this research an interactive system is defined as software or a hardware device with a user interface that requires human input. Testing interactive systems behave as expected is an important part of the development process, as with human input comes the possibility of human error, which adds to the potential problems that may exist in the system. In safety critical situations, which can result in injury or death to the user of the system (see [3, 6, 7]), it is essential

to ensure that interactive systems behave correctly in order to help prevent possible incidents from occurring.

An example of a safety critical interactive system is an infusion pump which is used in medical settings to disperse medicine. The U.S. Food and Drug Administration (FDA) have received approximately 56000 different reports from 2005 to 2009 of incidents due to failure with infusion pumps, several of these incidents resulted in injury or death [9]. We can conclude from this evidence that current testing approaches are still inadequate.

One reason for this is that testing interactive systems is hard. It requires a substantial knowledge of the testing approach being applied as well as an understanding of how the system works and an understanding of how users might interact. While neither of these are impossible to obtain, it is difficult to decide what should be tested and how, commonly safety critical interactive systems have conceptually infinite state spaces of interactions (in practice this is untrue as user cannot interact infinitely).

There are some existing approaches which have used interaction sequences to support system testing (see [5, 8, 2]). However, these approaches are incomplete due to the way they restrict the sequences and lack of test oracles generation. Test oracles allow us to distinguish correct behaviour from potentially incorrect behaviour [1]. It is imperative that test oracles are used to ensure that the system meets its specifications.

In this research we propose using interaction sequences, which are sequences of actions that the user can take, as an abstraction. We are currently investigating useful ways of constraining them to a finite length to help aid the testing process and determine coverage metrics.

Research Questions

Our main goal for this research is to create a better testing approach (when compared with existing approaches) for interactive systems using interaction sequences. However, conceptually interaction sequences have the possibility to be never ending, and also have the possibility for never ending combinations of those never ending sequences. Thus we need to discover ways to constrain the sequences in order to make them tractable, without loss of system coverage. Furthermore, we need to make decisions about what interaction is, and whether testing a component once is sufficient as testing it multiple times. This has led to the following research questions:

- Can interaction sequences be used to develop a better testing approach for interactive systems compared to existing approaches?
- How can we automatically generate sequences for testing purposes?
- How do we limit this state space but still adequately test the system?
- Is interacting with a component once the same as interacting with that component a number of times?

Progress

Work on this research has been ongoing for 2 years and the current focus is on investigating ways of representing interaction sequences so that they can be automatable for testing purposes. Current investigations have involved using regular grammars and finite state automata (FSA). These allow us to formally describe and reason about the interaction sequences such that we can generate new sequences, experimenting with changing lengths and types.

Furthermore, they allow us to draw on existing theory (see [4]).

Reducing complexity of the sequences is a significant goal in this work as this will allow us to constrain them in a tractable way. By using existing techniques for removing non-determinism and minimising automata we can reduce the number of states required to formally describe a sequence. We are investigating the effect that this has on the sequences.

In addition to this we are also exploring the use of abstraction to simplify the sequences, such that sequences can be used in a more modular fashion. We are investigating this by using equivalence techniques to see if an abstracted and non-abstracted FSA for the same sequence accept the same language.

In order to abstract different parts of sequences we have defined a property called self-containment, this allows us to identify different parts of the sequences to abstract. For example, there may be parts of a system which we wish to ignore for various reasons (such as being non-safety critical or software defined), abstracting in this way allows us to hide this information. However, if we discover we need to investigate this abstracted part of the sequence later we are able to retrieve this accordingly.

Our current focus is on ensuring that the definitions for self-containment are broad enough for different types of interaction sequences, and consequently proving that these are true. It is important that we have a reliable way of hiding and retrieving this information as we could inadvertently change the meaning of the sequence if information is lost.

However, we have already discovered an issue with the self-containment property for reducing complexity. In highly connected machines, this property will be true for the machine as a whole, preventing modularisation. This will not allow us to simplify the overall sequence for testing purposes and will be particularly problematic for models at the widget-level.

Should we be successful in using abstraction in this way it would be interesting to see how this compares with the abstractions humans automatically make when following their own interactions sequences. This would be the next step in finding useful ways of using interaction sequences for testing purposes.

Open Questions

- Can abstraction be used to modularise sequences in order to make them less complex, without affecting the sequence meaning?
- Can existing equivalence theory be used to determine if two sequences are isomorphically equivalent, in order to reduce complexity?
- Does the self-containment property hold true for all machines?
- If all previous questions are true, how does the abstraction of the system sequences compare with actual user abstractions of the same interaction sequences?

REFERENCES

1. E.T. Barr, M. Harman, P. McMinn, M. Shahbaz, and Shin Yoo. 2015. The Oracle Problem in Software Testing: A Survey. *Software Engineering, IEEE Transactions on* 41, 5 (May 2015), 507–525. DOI: http://dx.doi.org/10.1109/TSE.2014.2372785
2. Fevzi Belli. 2003. A Holistic View for Finite-State Modeling and Testing of User Interactions. In

Proceedings of the 1st South-East European Workshop on Formal Methods.

3. Daily Mail Reporter. 2011. Mother Dies after Nurse makes Error Administering Drug. (23 February 2011). Retrieved July 27, 2015 from http://www.dailymail.co.uk/health/article-1359778/Mother-dies-nurse-administers-TEN-times-prescribed-drug.html.
4. John E Hopcroft. 1979. *Introduction to Automata Theory, Languages, and Computation*. Pearson Education India.
5. Bao N Nguyen, Bryan Robbins, Ishan Banerjee, and Atif Memon. 2014. GUITAR: an Innovative Tool for Automated Testing of GUI-driven Software. *Automated Software Engineering* 21, 1 (2014), 65–105.
6. Anne Marie Porrello. n.d. Death and Denial: The Failure of the THERAC-25, A Medical Linear Accelerator. (n.d.). Retrieved July 27, 2015 from http://users.csc.calpoly.edu/~jdalbey/SWE/Papers/THERAC25.html.
7. The Economist. 2012. When code can kill or cure. (2 June 2012). Retrieved December 9, 2015 from http://www.economist.com/node/21556098.
8. Harold Thimbleby. 2004. User Interface Design with Matrix Algebra. *ACM Transactions on Computer-Human Interaction (TOCHI)* 11, 2 (2004), 181–236.
9. U.S. Food and Drug Administration. 2016. Infusion Pumps. (23 February 2016). Retrieved September 29, 2016 from http://www.fda.gov/MedicalDevices/ProductsandMedicalProcedures/GeneralHospitalDevicesandSupplies/InfusionPumps/.