Property-Based Testing of Agent-Based Simulations

A functional approach

Jonathan Thaler
Thorsten Altenkirch
jonathan.thaler@nottingham.ac.uk
thorsten.altenkirch@nottingham.ac.uk
University of Nottingham
Nottingham, United Kingdom

ABSTRACT

This paper presents a new approach to test the implementation of an agent-based simulation, called property-based testing which allows to test specifications much more directly than unit tests. Although being more expressive than unit tests, property-based testing is seen as a complementary and does not make unit-testing obsolete. We present two different models as case-studies in which we will show how to apply property-based testing to exploratory and explanatory agent-based models and what its limits are. We conduct our implementations in the pure functional programming language Haskell, which is the origin of property-pased testing. We show that simply by switching to such a language one gets rid of a large class of run-time bugs and is able to make stronger guarantees of correctness already at compile time without writing tests for some parts. Further, it makes isolated unit-tests quite easier.

TODO: this would be ideal to submit to an ABS conference so i can also discuss functional programming

TODO: write introduction TODO: implement case study 1: property-testing of SIR TODO: implement case study 2: property-testing of Sugarscape TODO: Show the use of Haskell Titan TODO: write discussion TODO: write background TODO: write related research TODO: write conclusion & further research

KEYWORDS

Agent-Based Simulation, Functional Reactive Programming, Property-Based Testing, Haskell

ACM Reference Format:
Jonathan Thaler and Thorsten Altenkirch. 2019. Property-Based Testing of Agent-Based Simulations: A functional approach. In *Proceedings of International Symposium on Implementation and Application of Functional Languages (IFL'18).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

When implementing an agent-based simulation (ABS) it is of utmost importance that the implementation is correct up to a specification, which is the model. To ensure that an implementation matches a specification, one uses verification, which ensures that *"we are building the model right"* (TODO: cite). With the established approaches in the field of ABS, which are primarily the object-oriented programming languages Java and Python, it is very difficult, or might even be impossible to *formally* prove that an implementation is correct up to a specification. Also one can say in general that formal proofs of correctness are highly complex and take a lot of effort and are almost always beyond the scope of a project and just not feasible. Still, not checking the correctness of the implementation in *some* way would be highly irresponsible and thus software engineers have developed the concept of unit-testing and test-driven development (todo: cite). Put shortly, in test-driven development one implements unit-tests for each feature to implement before actually implementing the feature. Then the features is implemented and the tests for it should pass. This cycle is repeated until the implementation of all requirements has finished. Of course it is important to cover the whole functionality with tests to be sure that all cases are checked which can be supported by code coverage tools to ensure that all code-paths have been tested. Thus we can say that test-driven development in general and unit-testing together with code-coverage in particular, allow to guarantee the correctness of an implementation to some informal degree which has been proven to be sufficiently enough through years of practice in the software industry all over the world. Also a fundamental strength of such tests is that programmers gain much more confidence when making changes to code - without such tests all bets are off and there is no reliable way to know whether the changes have broken something or not. The work of [3] discusses the use of test-driven development with unit-tests to implement ABS and we support the position that every ABS which has some degree of complexity and is used for some form of policy or decision making should be thoroughly tested.

In this paper we discuss a complementary method of testing the implementation of an ABS, called property-based testing, which to our best knowledge has not been discussed in the field of ABS yet. We present two models for case-studies. First, the SIR model, which is of explanatory nature, where we show how to express formal model-specifications in property-tests. Second, the SugarScape model, which is exploratory nature, where we show how to express hypotheses in property-tests.

Property-based testing has its origins in the pure functional programming language Haskell (TODO: cite quickcheck papers and history of haskell paper) so we discuss it from that perspective. We show that besides property-based testing, Haskell has a lot more to offer in regards to testing and guaranteeing the correctness of an ABS implementation. Simply by switching to this language will remove a large class of run-time bugs and allows to make stronger guarantees about the correctness of the ABS implementation, making the implementation *very likely* to be correct.

IFL'18, August 2019, Lowell, MA, USA
2019. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The aim of this paper is to investigate the potential of property-based testing which allows to directly express model-specifications in code and test them.

The contributions of this paper are:

- Showing that by simply switching to a pure functional programming language removes a large class of run-time errors and allows much stronger guarantees of correctness already at compile time and without actual need of tests.
- Expanding the testing techniques of ABS implementations by property-based testing, an additional, highly expressive method from which we hope it makes testing easier and ABS implementations which utilise it more likely to be correct.

The structure of the paper is as follows. To make this paper sufficiently self-contained we give a brief introduction of the concepts of pure functional programming in Haskell in Section TODO. Then we introduce property-based testing in general in Section TODO. Then we present both case-studies in Section TODO and Section TODO. We follow with related work in Section TODO. Finally we conclude and give further research in Section TODO.

2 RELATED WORK

TODO: Test-driven agent-based simulation development [3] TODO: Back To the Future: Time Travel in FRP [4] TODO: Testing and Debugging Functional Reactive Programming [5]

3 BACKGROUND

TODO: List of Common Bugs and Programming Practices to avoid them [6] TODO: QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs [1] TODO: Testing Monadic Code with QuickCheck [2]

ACKNOWLEDGMENTS

The authors would like to thank

REFERENCES

[1] Koen Claessen and John Hughes. 2000. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00)*. ACM, New York, NY, USA, 268–279. https://doi.org/10.1145/351240.351266

[2] Koen Claessen and John Hughes. 2002. Testing Monadic Code with QuickCheck. *SIGPLAN Not.* 37, 12 (Dec. 2002), 47–59. https://doi.org/10.1145/636517.636527

[3] N. Collier and J. Ozik. 2013. Test-driven agent-based simulation development. In *2013 Winter Simulations Conference (WSC)*. 1551–1559. https://doi.org/10.1109/WSC.2013.6721538

[4] Ivan Perez. 2017. Back to the Future: Time Travel in FRP. In *Proceedings of the 10th ACM SIGPLAN International Symposium on Haskell (Haskell 2017)*. ACM, New York, NY, USA, 105–116. https://doi.org/10.1145/3122955.3122957

[5] Ivan Perez and Henrik Nilsson. 2017. Testing and Debugging Functional Reactive Programming. *Proc. ACM Program. Lang.* 1, ICFP (Aug. 2017), 2:1–2:27. https://doi.org/10.1145/3110246

[6] V. Vipindeep and Pankaj Jalote. 2005. *List of Common Bugs and Programming Practices to avoid them.* Technical Report. Indian Institute of Technology, Kanpur.

Received May 2018