The Agents' new Cloths

Towards pure functional agent-based simulation

Jonathan Thaler
Thorsten Altenkirch
Peer-Olaf Siebers
jonathan.thaler@nottingham.ac.uk
thorsten.altenkirch@nottingham.ac.uk
peer-olaf.siebers@nottingham.ac.uk
University of Nottingham
Nottingham, United Kingdom

ABSTRACT

TODO: implement MSF SugarScape TODO: implement STM Sugarscape TODO: implement dependently typed SIR TODO: implement dependently typed Sugarscape TOOD: shortly discuss what other monads would allow in the context of FRP ABS: maybe, ...

So far, the pure functional paradigm hasn't got much attention in Agent-Based Simulation (ABS) where the dominant programming paradigm is object-orientation, with Java, Python and C++ being its most prominent representatives. We claim that pure functional programming using Haskell is very well suited to implement complex, real-world agent-based models and brings with it a number of benefits. To show that we implemented the seminal Sugarscape model in Haskell in our library *FrABS* which allows to do ABS the first time in the pure functional programming language Haskell. To achieve this we leverage the basic concepts of ABS with functional reactive programming using Yampa. The result is a surprisingly fresh approach to ABS as it allows to incorporate discrete time-semantics similar to Discrete Event Simulation and continuous time-flows as in System Dynamics. In this paper we will show the novel approach of functional reactive ABS through the example of the SIR model, discuss implications, benefits and best practices.

KEYWORDS

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

ACM Reference Format:
Jonathan Thaler, Thorsten Altenkirch, and Peer-Olaf Siebers. 2019. The Agents' new Cloths: Towards pure functional agent-based simulation. 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

The traditional approach to Agent-Based Simulation (ABS) has so far always been object-oriented techniques, due to the influence of the seminal work of Epstein et al [1] in which the authors claim "[..] object-oriented programming to be a particularly natural development environment for Sugarscape specifically and artificial societies generally [..]" (p. 179). This work established the metaphor in the ABS community, that *agents map naturally to objects* [3] which still holds up today.

In this paper we challenge this metaphor and explore ways of approaching ABS using the functional programming paradigm with the language Haskell. By doing this we expect to leverage the benefits of it [2]: TODO: this is already by far too technical here, we are writing for people who have basically no understanding of FP. higher expressivity through declarative code, being polymorph and explicit about side-effects through monads, more robust and less susceptible to bugs due to explicit data flow and lack of implicit side-effects.

As use-case we employ the famous SugarScape model [1] because it can be seen as one of the most famous models in ABS which also laid the foundations of using object-oriented programming.

TODO: this is not what we are doing here Over the course of four steps, we derive all necessary concepts required for a full agent-based implementation. We start from a very simple solution running in the Random Monad which has all general concepts already there and then refine it in various ways, making the transition to Functional Reactive Programming (FRP) [5] and to Monadic Stream Functions (MSF) [4].

The aim of this paper is to show *how* to implement ABS in functional programming as in Haskell and *why* it is of benefit of doing so. By doing this we give the reader a good understanding of what functional programming is, what the challenges are in applying it to ABS and how we solve these in our approach. The paper makes the following contributions:

1.1 Contributions

- first to systematically introduce functional programming concepts to agent-based simulation - applied property-based testing to ABS - applied STM to ABS

2 CONCLUSIONS AND FUTURE WORK

2.1 Advantages

- being explicit and polymorph about side-effects: can have 'pure' (no side-effects except state), 'random' (can draw random-numbers), 'IO' (all bets are off), STM (concurrency) agents

- hybrid between SD and ABS due to continuous time AND parallel dataFlow (parallel update-strategy)

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

- being update-strategy polymorph (TODO: this is just an asumption atm, need to prove this): 4 different update-strategies, one agent implementation

- parallel update-strategy: lack of implicit side-effects makes it work without any danger of data-interference

- recursive simulation - reasoning about correctness - reasoning about dynamics - testing with quickcheck much more convenient - expressivity: -> 1:1 mapping of SD to code: can express the SD formulas directly in code -> directly expressing state-charts in code

2.2 Disadvantages

2.2.1 Performance. Performance is currently no where near imperative object-oriented implementations. The reason for this is that we don't have in-place updates of data-structures and make no use of references. This results in lots of copying which is simply not necessary in the imperative languages with implicit effects. Also it is much more difficult to reason about time and space in our approach. Thus we see performance clearly as the main drawback of the functional approach and the only real advantage of the imperative approach over our.

2.2.2 Steep learning curve. Our approach is quite advanced in three ways. First it builds on the already quite involved FRP paradigm. Second it forces one to think properly of time-semantics of the model, how to sample it, how small Δt should be and whether one needs super-sampling or not and if yes how many samples one should take. Third it requires to think about agent-interaction and update-strategies, whether one should use conversations which forces one to run sequentially or if one can run agents in parallel and use normal messaging which incurs a time-delay which in turn would need to be considered when setting the Δt.

2.3 Future Work

TODO: what we need to show / future work - can we do DES? e.g. single queue with multiple servers? also specialist vs. generalist - reasoning about correctness: implement Gintis & Ionescous papers - reasoning about dynamics: implement Gintis & Ionescous papers

ACKNOWLEDGMENTS

The authors would like to thank J. Greensmith for constructive feedback, comments and valuable discussions.

REFERENCES

[1] Joshua M. Epstein and Robert Axtell. 1996. *Growing Artificial Societies: Social Science from the Bottom Up.* The Brookings Institution, Washington, DC, USA.

[2] Paul Hudak, John Hughes, Simon Peyton Jones, and Philip Wadler. 2007. A History of Haskell: Being Lazy with Class. In *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages (HOPL III).* ACM, New York, NY, USA, 12–1–12–55. https://doi.org/10.1145/1238844.1238856

[3] Michael J. North and Charles M. Macal. 2007. *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation.* Oxford University Press, USA. Google-Books-ID: gRATDAAAQBAJ.

[4] Ivan Perez, Manuel Baerenz, and Henrik Nilsson. 2016. Functional Reactive Programming, Refactored. In *Proceedings of the 9th International Symposium on Haskell (Haskell 2016).* ACM, New York, NY, USA, 33–44. https://doi.org/10.1145/2976002.2976010

[5] Zhanyong Wan and Paul Hudak. 2000. Functional Reactive Programming from First Principles. In *Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation (PLDI '00).* ACM, New York, NY, USA, 242–252. https://doi.org/10.1145/349299.349331

Received May 2018