PhD Thesis

The Pure Functional Programming Paradigm In Agent-Based Simulation

Jonathan Thaler (4276122)
jonathan.thaler@nottingham.ac.uk

supervised by
Dr. Peer-Olaf Siebers
Dr. Thorsten Altenkirch

December 26, 2018

Abstract

This thesis shows how to implement Agent-Based Simulations (ABS) using the *pure* functional programming paradigm and what the benefits and drawbacks are when doing so. As language of choice, Haskell is used due to its modern nature, increasing use in real-world applications and *pure* nature. The thesis presents various implementation techniques to ABS and then discusses concurrency and parallelism and verification and validation in ABS in a pure functional setting. Additionally the thesis briefly discusses the use of dependent types in ABS, to close the gap between specification and implementation - something the presented implementation techniques don't focus on. Finally a case-study is presented which tries to bring together the insights of the previous chapters by replicating an agent-based model both in pure and dependently typed functional programming. The agent-based model which was selected was much discussed in ABS communities as it claimed to have solved a fundamental problem of economics but it was then found that the implementation had a number of bugs which shed doubt on the validity and correctness of the results. The thesis' case study investigates whether this failure could have happened in pure and dependent functional programming and is a further test to see of how much value functional programming is to ABS.

Contents

1 Introduction **5**
1.1 Publications . . . 6
1.2 Contributions . . . 7
1.3 Thesis structure . . . 8

2 Literature Review **9**

3 Methodology **10**
3.1 Agent-Based Simulation . . . 10
3.1.1 Traditional approaches . . . 10
3.1.2 Verification & Validation . . . 10
3.2 Pure functional programming . . . 10
3.2.1 Functional Reactive Programming . . . 10
3.2.2 Monadic Stream Functions . . . 11
3.3 Dependent Types . . . 11

4 Pure Functional ABS **12**
4.1 Time-Driven ABS . . . 12
4.2 Event-Driven ABS . . . 12
4.2.1 Synchronised Agent-Interactions . . . 12

5 Parallelism and Concurrency **13**
5.1 Adding Parallelism . . . 13
5.2 Adding Concurrency with STM . . . 13

6 Verification & Correctness **14**
6.1 Using the Type-System . . . 14
6.2 Reasoning . . . 14
6.3 Unit-Testing . . . 14
6.4 Property-Based Testing . . . 14

7 Dependent Types **15**

8 The Gintis Case-Study **16**
8.1 A pure functional implementation 16
8.2 Exploiting property-based tests 16
8.3 A dependently typed implementation 16
8.4 Discussion 16

9 Discussion **17**
9.1 Generalising Research 17
9.1.1 Simulation in general 17
9.1.2 System Dynamics 17
9.1.3 Discrete Event Simulation 18
9.1.4 Recursive Simulation 18
9.1.5 Multi Agent Systems 18

10 Conclusions **19**
10.1 Further Research 19

Appendices **22**

Acknowledgements

Peer-Olaf Siebers Supervisor Thorsten Altenkirch Supervisor FP Group, for always being open to discussions, keen to help. Martin Handley and James Hey for working through my thesis and their feedback. Ivan Perez for always having an open ear for questions, valuable discussions with him and his contributions. Julie Greensmith for the valuable discussions and pointing me into right directions at important stages of the phd.

Chapter 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 [3] 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* [11] which still holds up today.

This thesis challenges that metaphor and explores ways of approaching ABS with the functional programming paradigm using the language Haskell. It is the first one to do so on a *systematical* level and develops a foundation by presenting fundamental concepts and advanced features to show how to leverage the benefits of it [7, 6] to become available when implementing ABS functionally. By doing this, the thesis both shows *how* to implement ABS purely functional and *why* it is of benefit of doing so, what the drawbacks are and also when a pure functional approach should *not* be used.

This thesis claims that the agent-based simulation community needs functional programming because of its *scientific computing* nature where results need to be reproducible and correct while simulations should be able to massively scale-up as well. This thesis will show that by using functional programming for implementing ABS it is easy to add parallelism and concurrency, the resulting simulations are easy to test and verify, suitable to apply new testing methods like property-based testing, guaranteed to be reproducible already at compile-time, have fewer potential sources of bugs and thus can raise the level of confidence in the correctness of an implementation to a new level.

In our research we are using the *pure* functional programming language Haskell. The paper of [6] gives a comprehensive overview over the history of the language, how it developed and its features and is very interesting to read and get accustomed to the background of the language. The main points why we decided to go for Haskell are:

- Rich Feature-Set - it has all fundamental concepts of the pure functional programming paradigm included. Further, Haskell has influenced a large number of languages, underlining its importance and influence in programming language design.

- Real-World applications - the strength of Haskell has been proven through a vast amount of highly diverse real-world applications [6], is applicable to a number of real-world problems [12] and has a large number of libraries available [1].

- Modern - Haskell is constantly evolving through its community and adapting to keep up with the fast changing field of computer science. Further, the community is the main source of high-quality libraries.

- Purity - Haskell is a *pure* functional language and in our research it is absolutely paramount, that we focus on *pure* functional ABS, which avoids any IO type under all circumstances (exceptions are when doing concurrency but there we restrict most of the concepts to STM).

- It is as closest to pure functional programming, as in the lambda-calculus, as we want to get. Other languages are often a mix of paradigms and soften some criteria / are not strictly functional and have different purposes. Also Haskell is very strong rooted in Academia and lots of knowledge is available, especially at Nottingham, Lisp / Scheme was considered because it was the very first functional programming language but deemed to be not modern enough with lack of sufficient libraries. Also it would have given the Erlang was considered in prototyping and allows to map the messaging concept of ABS nicely to a concurrent language but was ultimately rejected due to its main focus on concurrency and not being purely functional. Scala was considered as well and has been used in the research on the Art Of Iterating paper but is not purely functional and can be also impure.

TODO

- Main Argument: Defining the problem, motivation, aim and scope of the Ph.D.

- Hypotheses: Precisely stating the hypotheses which will form the points of reference for the whole research.

1.1 Publications

Throughout the course of the Ph.D. four (4) papers were published:

1. The Art Of Iterating - Update Strategies in Agent-Based Simulation [14] - This paper derives the 4 different update-strategies and their properties

[1] `https://wiki.haskell.org/Applications_and_libraries`

possible in time-driven ABS and discusses them from a programming-paradigm agnostic point of view. It is the first paper which makes the very basics of update-semantics clear on a conceptual level and is necessary to understand the options one has when implementing time-driven ABS purely functional.

2. Pure Functional Epidemics [13] - Using an agent-based SIR model, this paper establishes in technical detail *how* to implement time-driven ABS in Haskell using non-monadic FRP with Yampa and monadic FRP with Dunai. It outlines benefits and drawbacks and also touches on important points which were out of scope and lack of space in this paper but which will be addressed in the Methodology chapter of this thesis.

3. A Tale Of Lock-Free Agents (TODO cite) - This paper is the first to discuss the use of Software Transactional Memory (STM) for implementing concurrent ABS both on a conceptual and on a technical level. It presents two case-studies, with the agent-based SIR model as the first and the famous SugarScape being the second one. In both case-studies it compares performance of STM and lock-based implementations in Haskell and object-oriented implementations of established languages. Although STM is now not unique to Haskell any more, this paper shows why Haskell is particularly well suited for the use of STM and is the only language which can overcome the central problem of how to prevent persistent side-effects in retry-semantics. It does not go into technical details of functional programming as it is written for a simulation Journal.

4. Towards Pure Functional Agent-Based Simulation (TODO cite) - This paper summarizes the main benefits of using pure functional programming as in Haskell to implement ABS and discusses on a conceptual level how to implement it and also what potential drawbacks are and where the use of a functional approach is not encouraged. It is written as a conceptual / review paper, which tries to "sell" pure functional programming to the agent-based community without too much technical detail and parlance where it refers to the important technical literature from where an interested reader can start.

1.2 Contributions

1. This thesis is the first to *systematically* investigate the use of the functional programming paradigm, as in Haskell, to ABS, laying out in-depth technical foundations and identifying its benefits and drawbacks. Due to the increased interested in functional concepts which were added to object-oriented languages in recent years because of its established benefits in concurrent programming, testing and software-development in general, presenting such foundational research gives this thesis significant impact.

Also it opens the way for the benefits of FP to incorporate into scientific computing, which are explored in the contributions below.

2. This thesis is the first to show the use of Software Transactional Memory (STM) to implement concurrent ABS and its potential benefit over lock-based approaches. STM is particularly strong in pure FP because of retry-semantics can be guaranteed to exclude non-repeatable persistent side-effects already at compile time. By showing how to employ STM it is possible to implement a simulation which allows massively large-scale ABS but without the low level difficulties of concurrent programming, making it easier and quicker to develop working and correct concurrent ABS models. Due to the increasing need for massively large-scale ABS in recent years [8], making this possible within a purely functional approach as well, gives this thesis substantial impact.

3. This thesis is the first to present the use of property-based testing in ABS which allows a declarative specification- testing of the implemented ABS directly in code with *automated* test-case generation. This is an addition to the established Test Driven Development process and a complementary approach to unit-testing, ultimately giving the developers an additional, powerful tool to test the implementation on a more conceptual level. This should lead to simulation software which is more likely to be correct, thus making this a significant contribution with valuable impact.

4. This thesis is the first to outline the potential use of *dependent types* to Agent-Based Simulation on a *conceptual level* to investigate its usefulness for increasing the correctness of a simulation. Dependent types can help to narrow the gap between the model specification and its implementation, reducing the potential for conceptual errors in model-to-code translation. This immediately leads to fewer number of tests required due to guarantees being expressed already at compile time. Ultimately dependent types lead to higher confidence in correctness due to formal guarantees in code, making this a unique contribution with high impact.

1.3 Thesis structure

TODO NOTE: focus on strong narrative thus all chapters are interconnected and tell the story

Chapter 2

Literature Review

This chapter discusses related work by presenting the relevant literature, following 1st & 2nd annual report, and all papers. Roughly 80% finished.

Chapter 3

Methodology

This chapter introduces the background and methodology used in the following chapters. Roughly 50% exists already.

3.1 Agent-Based Simulation

History, methodology (what is the purpose of ABS: 3rd way of doing science: exploratory, helps understand real-world phenomena), classification according to [9], ABS vs. MAS, event- vs. time-driven [10], examples: agent-based SIR, SugarScape, Gintis Bartering

3.1.1 Traditional approaches

Introduce established implementation approaches to ABS. Frameworks: NetLogo, Anylogic, Libraries: RePast, DesmoJ. Programming: Java, Python, C++. Correctness: ad-hoc, manual testing, test-driven development.

3.1.2 Verification & Validation

Introduction Verification & Validation (V & V in the context of ABS).

3.2 Pure functional programming

Definition, Haskell references,

3.2.1 Functional Reactive Programming

Short introduction to FRP (yampa), based on my pure functional epidemics paper.

3.2.2 Monadic Stream Functions

Short introduction to MSFs (dunai), based on my pure functional epidemics paper.

3.3 Dependent Types

Example, Equality as Type, Philosophical Foundations: Constructive mathematics

Chapter 4

Pure Functional ABS

Is the main chapter of the thesis to discuss *how* can we do pure functional ABS. Roughly 75% exists already.

4.1 Time-Driven ABS

Following pure functional ABS, also discuss how can we implement the 4 update-strategies of the art-iterating paper in our pure functional approach.

4.2 Event-Driven ABS

Following towards papers SugarScape implementation

4.2.1 Synchronised Agent-Interactions

Following towards papers SugarScape implementation

Chapter 5

Parallelism and Concurrency

Establish how concurrency and parallelism can be made easily available in ABS using pure functional programming. Mostly follow STM paper and add pure parallelism in ABS. Make clear that Haskell allows to distinguish between pure, deterministic parallelism and impure, non-deterministic concurrency.

About 50% finished.

5.1 Adding Parallelism

Discusses where there is potential for adding parallelism: using data-parallel data-structures for the environment so cells can be updated in parallel, in time-driven ABS agents can be updated in parallel using parMap because they all act conceptually at the same time (and if they don't run in monadic code). 0% finished.

5.2 Adding Concurrency with STM

This is a shorter recap of the STM paper, 100% finished.

Chapter 6

Verification & Correctness

Exploring ways in which pure functional ABS can be of benefit to verification & validation and increasing correctness of an ABS implementation.

Roughly 50%

6.1 Using the Type-System

Static type system eliminates a large number run-time bugs.

6.2 Reasoning

TODO: can we apply equational reasoning? Can we (informally) reason about various properties e.g. termination?

6.3 Unit-Testing

Follow unit-testing of the whole simulation as prototyped for towards paper.

6.4 Property-Based Testing

Follow property-based testing as prototyped for towards paper.

Chapter 7

Dependent Types

The pure functional implementation techniques have a number of technical benefits but don't help as much in closing the gap between specification and implementation as one is used from functional programming in general. Therefore we take a step back and abstract from these highly complex implementation techniques and move towards dependent types. Follow [1] and [2].

Conceptually discuss how dependent types can be made of use in ABS without going into lot of technical detail because: 1. i didn't do enough research on it and 2. dependent types seem to be nearly out of focus of the thesis.

Chapter 8

The Gintis Case-Study

Apply my developed techniques to the Gintis paper (and its follow ups: the Ionescu paper [2] and a Masterthesis [4] on it). The aim of this study is to see:

1. Do the techniques transfer to this problem and model?
2. Could pure functional programming have prevented the bugs which Gintis made?
3. Would property-based tests have been of any help to preven the bugs?
4. Could dependent and / or types have prevented the bugs which Gintis made?
5. How close is our (dependently typed) implementation to Ionescus functional specification?
6. When having Cezar Ionescu as external examiner, this chapter will be of great influence as it deals heavily with his work.

Not yet started, need to implement it but there exists code for it already (gintis and java implementations)

8.1 A pure functional implementation

8.2 Exploiting property-based tests

8.3 A dependently typed implementation

8.4 Discussion

Chapter 9

Discussion

This chapter re-visits the aim, objective and hypotheses of the introduction and puts them into perspective with the contributions. About 20% exists already.

9.1 Generalising Research

We hypothesize that our research can be transferred to other related fields as well, which puts our contributions into a much broader perspective, giving it more impact than restricting it just to the very narrow field of Agent-Based Simulation. Although we don't have the time to back up our claims with in-depth research, we argue that our findings might be applicable to the following fields at least on a conceptual level.

9.1.1 Simulation in general

We already showed in the paper [13], that purity in a simulation leads to repeatability which is of utmost importance in scientific computation. These insights are easily transferable to simulation software in general and might be of huge benefit there. Also my approach to dependent types in ABS might be applicable to simulations in general due to the correspondence between equilibrium & totality, in use for hypotheses formulation and specifications formulation as pointed out in section **??**.

9.1.2 System Dynamics

discuss pure functional system dynamics - correct by construction: benefits: strictly deterministic already at compile time, encode equations directly in code =¿ correct by construction. Can serve as backend implementation of visual SD packages.

9.1.3 Discrete Event Simulation

pure functional DES easily possible with my developed synchronous messaging ABS DES in FP: we doing it in gintis study, PDES, should be conceptually easil possible using STM, optimistic approach should be conceptually easier to implement due to persistent data-structures and controlled side-effects

9.1.4 Recursive Simulation

Inspired by [5], add ideas about recursive simulation described in 1st year report and "paper". functional programming maps naturally here due to its inherently recursive nature and controlled side-effects which makes it easier to construct correct recursive simulations. recursive simulation should be conceptually easier to implememt and more likely to be correct due to recursive Nature of haskell itself, lack of sideeffeccts and mutable data

9.1.5 Multi Agent Systems

The fields of Multi Agent Systems (MAS) and ABS are closely related where ABS has drawn much inspiration from MAS [16], [15]. It is important to understand that MAS and ABS are two different fields where in MAS the focus is more on technical details, implementing a system of interacting intelligent agents within a highly complex environment with the focus on solving AI problems.

Because in both fields, the concept of interacting agents is of fundamental importance, we expect our research also to be applicable in parts to the field of MAS. Especially the work on dependent types should be very useful there because MAS is very interested in correctness, verification and formally reasoning about a system and their agents, to show that a system follows a formal specifications.

Chapter 10

Conclusions

This chapter concludes the whole thesis and outlines future research. Roughly 20% exists already.

10.1 Further Research

1. generalise concepts explored into a pure functional ABS library in Haskell (called chimera), 2. dependent types and linear types are the next big step

References

[1] Botta, N., Mandel, A., and Ionescu, C. Time in discrete agent-based models of socio-economic systems. Documents de travail du Centre d'Economie de la Sorbonne 10076, Universit Panthon-Sorbonne (Paris 1), Centre d'Economie de la Sorbonne, 2010.

[2] Botta, N., Mandel, A., Ionescu, C., Hofmann, M., Lincke, D., Schupp, S., and Jaeger, C. A functional framework for agent-based models of exchange. *Applied Mathematics and Computation 218*, 8 (Dec. 2011), 4025–4040.

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

[4] Evensen, P., and Mrdin, M. An Extensible and Scalable Agent-Based Simulation of Barter Economics. Master's thesis, Chalmers University of Technology, Gteborg, 2010.

[5] Gilmer, Jr., J. B., and Sullivan, F. J. Recursive Simulation to Aid Models of Decision Making. In *Proceedings of the 32Nd Conference on Winter Simulation* (San Diego, CA, USA, 2000), WSC '00, Society for Computer Simulation International, pp. 958–963.

[6] Hudak, P., Hughes, J., Peyton Jones, S., and Wadler, P. A History of Haskell: Being Lazy with Class. In *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages* (New York, NY, USA, 2007), HOPL III, ACM, pp. 12–1–12–55.

[7] Hudak, P., and Jones, M. Haskell vs. Ada vs. C++ vs. Awk vs. ... An Experiment in Software Prototyping Productivity. Research Report YALEU/DCS/RR-1049, Department of Computer Science, Yale University, New Haven, CT, Oct. 1994.

[8] Lysenko, M., and D'Souza, R. M. A Framework for Megascale Agent Based Model Simulations on Graphics Processing Units. *Journal of Artificial Societies and Social Simulation 11*, 4 (2008), 10.

[9] Macal, C. M. Everything you need to know about agent-based modelling and simulation. *Journal of Simulation 10*, 2 (May 2016), 144–156.

[10] Meyer, R. Event-Driven Multi-agent Simulation. In *Multi-Agent-Based Simulation XV* (May 2014), Lecture Notes in Computer Science, Springer, Cham, pp. 3–16.

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

[12] O'Sullivan, B., Goerzen, J., and Stewart, D. *Real World Haskell*, 1st ed. O'Reilly Media, Inc., 2008.

[13] Thaler, J., Altenkirch, T., and Siebers, P.-O. Pure Functional Epidemics - An Agent-Based Approach. In *International Symposium on Implementation and Application of Functional Languages* (Lowell, Massachusettes, Aug. 2019).

[14] Thaler, J., and Siebers, P.-O. The Art Of Iterating: Update-Strategies in Agent-Based Simulation.

[15] Weiss, G. *Multiagent Systems*. MIT Press, Mar. 2013. Google-Books-ID: WY36AQAAQBAJ.

[16] Wooldridge, M. *An Introduction to MultiAgent Systems*, 2nd ed. Wiley Publishing, 2009.

Appendices

Datasets, lengthy code, additional proofs.