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 24, 2018

Abstract

TODO

TODO: from the 2nd annual review i got the feedback that i need to come up with a more coherent thesis structure which tells a story: i need to make myself clear what story i want to tell with my phd and what research and publications i still need to do for that (identify chapters and to which extent they are already finished). Also I need to come up with a precise publication plan, focusing on writing papers early than doing research for a too long time and starting an early thesis writing. This is because any unpublished research is lost and every published research is a contribution to knowledge. "A good PhD is asking more questions than it answers."

Contents

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

TODOS

- Do the Gintis-Case Study
- Investigate property-based testing more
- Recursive Schelling Segregation
- Linear and Dependent Types with Idris 2: more general ideas / hints / research on how it is applicable to ABS

PURE functional programming: make PURE very clear, this is what we are after

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 [**?**] 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* [**?**] 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 [**?**, **?**] 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. The established object-oriented approaches need considerably high effort and might even fail to deliver these objectives due to its conceptually

different approach to computing. In contrast, 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.

TODO: make it very very precise and clear why haskell? (also make it very clear that haskell is rising and has proven itself valuable in the industry, and has influenced a large number of programming languages)

This chapter is the introduction to the thesis ("why do we need functional programming in ABS?") and motivates it and describes the aim and scope of the Ph.D. Further it states the hypotheses and contributions.

- 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.
- Contributions: Precisely list the contribution to knowledge this Ph.D. makes and list all papers which were written (and published) during this Ph.D.

TODO: understand Gintis failure (read gintis paper and the masterthesis)

Argument & Story of my Thesis with the components (publications) i have so far and which i still plan to do

dependent types will be put to rest unless there is really some time left (which i doubt) and will be considered in the final thesis but only on a conceptual level 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. Dependent Types sections will be written very late, after the finished research has been incorporated, and if at that point i have come to the conclusion that it is too unfinished / vague / not substantial enough then i will simply skip this part (or if it is too long already and the contribution is enough already withouth dependent types).

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 [**?**] - This paper derives the 4 different update-strategies and their properties 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 [**?**] - This paper establishes in technical detail *how* to implement 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. Hands Off My Property (TODO cite) - This paper is the first to apply property-based testing, as in Haskells QuickCheck and SmallCheck libraries, to test and verify ABS implementations. It discusses the problem mainly on a conceptual level without going too much into detail as it is targeted towards a simulation Journal / Conference. Also it argues that by simply using Haskell as implementation language one gets automatically rid of a large class of run-time bugs which can occur in the established languages. Further it shows that the use of Haskell makes it easier to test and verify ABS because code tends to be more modular and thus easier tested and reasoned about, which is particularly well supported in Haskell due to the strong static type system and explicitness / lack of side-effects.

There has been unpublished work as well:

1. 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.

2. The Equilibrium - Totality Correspondence (TODO cite) - This extended abstract is the first which conceptually investigates the use of dependent types in ABS, where it investigates the connection between the equilibrium of an agent-based model and a (compiler-checked) total implementation of such a simulation using the dependently typed language Idris. It hypothesise that dependent types *might* be the future of ABS due to its constructive nature and stronger guarantees at compile time but that it is far too early for this development to arrive in the mainstream.

3. Implementing Correct-By-Construction System Dynamics (A functional pearl) - This short paper shows how to implement a System Dynamics model in Haskell using FRP with the Yampa library. The code is developed step-by-step and is regarded as correct-by-construction due to its declarative nature it matches the specification.

1.2 Contributions

1. This thesis is the first to *systematically* investigate the use of the functional programming paradigm, as in Haskell, to Agent-Based Simulation, 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 STM in 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 [**?**], 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- and hypothesis 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

focus on strong narrative thus all chapters are interconnected and tell the story
leave dependent types for further research but write an additional short chapter on it which uses 2nd year report

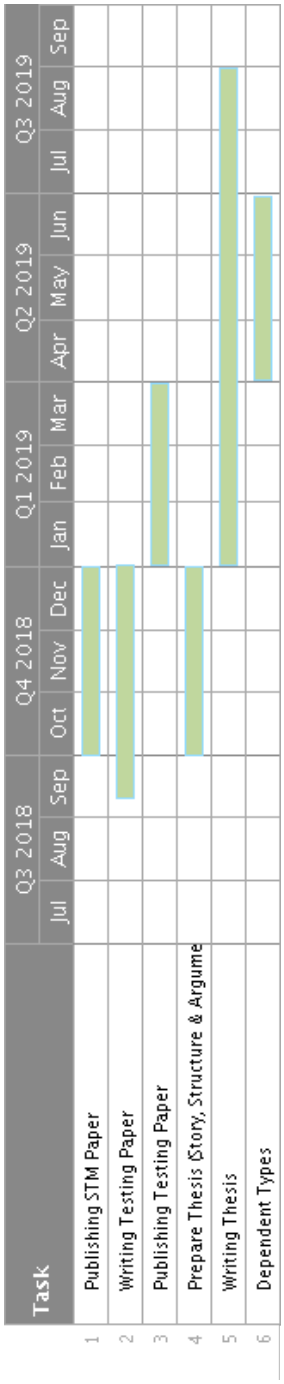

Figure 1.1: Project Plan 3rd Year

Chapter 2

Literature Review

This chapter discusses background and related work by presenting the relevant literature following 1st & 2nd annual report, and all papers 75%

Chapter 3

Methodology

This chapter introduces the methodology, used in the following chapters:

1. time-driven vs. event-driven ABS 2. time-driven: start with update-strategies 3. present pure functional approach 4. how can we implement the 4 update-strategies in our approach 5. interaction between agents 6. event-driven ABS

- Defining and introducing Agent-Based Simulation (ABS) (History, ABS vs. MAS, examples, event- vs. time-driven).
- Introduce established implementation approaches to ABS (Frameworks: NetLogo, Anylogic, Libraries: RePast, DesmoJ, Programming: Java, Python, Correctness: ad-hoc, manual testing, test-driven development)
- Introduction Verification & Validation (V & V in the context of ABS).
- Introduction to functional Programming in Haskell (functions, types, recursion, algebraic data-types, higher-order functions, continuations, Define and explain side-effects and purity: monads, different types of effects, explain IO and that it is of fundamental importance to avoid it in our research).
- Introduction to dependent types (Example, Equality as Type, Philosophical Foundations: Constructive mathematics)

50%

TODO: need carefully establish why Haskell:

- Rich Feature-Set - it has all fundamental concepts of the pure functional programming paradigm of which we explain the most important below.
- Real-World applications - the strength of Haskell has been proven through a vast amount of highly diverse real-world applications [**?**], is applicable

to a number of real-world problems [**?**] 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.

- 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: need to present the established approach to ABS: object-oriented, java/python, unit-tests, test-driven, RePast, AnyLogic, NetLogo

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

Chapter 4

Pure Functional ABS

"how can we do ABS in functional programming?" art of iterating paper for basic foundations, conceptual paper for a high level description, pure functional epidemics for more details, sync communication in FRP for the trickiest part 50%

Chapter 5

Concurrency

- concurrency and parallelism -¿ STM vs. lock based -¿ pure parallelism in ABS
  STM paper 50%

Chapter 6

Verification & Correctness

Testing of functional ABS paper 10%

- correctness & verification -¿ static type system eliminates a large number run-time bugs: if we decide to rule out IO then we can guarantee

Chapter 7

Dependent Types

dependent types will be put to rest unless there is really some time left (which i doubt) and will be considered in the final thesis but only on a conceptual level 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. Dependent Types sections will be written very late, after the finished research has been incorporated, and if at that point i have come to the conclusion that it is too unfinished / vague / not substantial enough then i will simply skip this part (or if it is too long already and the contribution is enough already without dependent types).

dependent types in ABS paper, totality paper 20%

7.1 Case-Study

perform gintis case-study: apply my developed techniques of implementing ABS and testing and concurrency / parallelism to the gintis paper (and its follow ups: the ionescu paper and the masterthesis on it). The aim of this is to see: 1. we can do DES under the hood, 2. property-based tests in a different setting, 3. could pure functional programming have prevented the failure? 4. could property based tests have prevented the failure? 5. could dependent and / or types have prevented the failure?

1. do the techniques transfer to this problem? 2. does haskell prevent making that mistake which gintis made? 3. how close is our implementation to ionescus functional specification? 4. validation and verification against gintis paper using property-based testing of individual agents and the simulation as a whole. 5. being more familiar with dependent types, would they help and where do they fit in in combination with the ionescu functional specification, which mentions dependent types at the end.

after re-reading ionescu paper: too complex and out of scope, but ionescu work more directly applicable in a pure functional implementation than in e.g. c++ (that was what they used).

we base our implementation on the existing gintis code from https://people.umass.edu/gintis/ also we make use of the masterthesis on gintis work which revealed a few bugs

reasoning about termination is easier

Chapter 8

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.

8.1 Simulation in general

We already showed in the paper TODO cite PFE, 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 **??**.

8.2 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.

8.3 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

8.4 Recursive Simulation

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

TODO: implement a recursive schelling seggregation

8.5 Multi Agent Systems

The fields of Multi Agent Systems (MAS) and ABS are closely related where ABS has drawn much inspiration from MAS [**?**], [**?**]. 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 9

Discussion

This chapter re-visits the hypotheses and puts them into perspective of the contributions. re-use arguments from all papers 20%

Chapter 10

Conclusions

This chapter draws conclusions to the main hypothesis and outlines future research. re-use conclusions and further research from all papers 20%

we now know how to engineer time- and event-driven ABS with complex state both in the agent and environment, main difficulty is direct agent-interaction (see macal classification into 4 types of ABS), compile-time guaranteed reproducibility, explicit handling of complex state (read only, read/write), concurrency explicit and limited to STM, very promising concurrency but direct agent-interactions main problem (erlang as a rescue?), main drawbacks: everything is explicit, performance

10.1 Further Research

chimera a multimethod Simulation library dependent types erlang (functional language) for concurrency

References

Appendices

Datasets, lengthy code, additional proofs.