1st Year Report
Pure Functional Methods in Agent-Based Modelling & Simulation

Jonathan Thaler
jonathan.thaler@nottingham.ac.uk

April 25, 2017

Abstract

A succinct and concise summary (250 words maximum) of the report contents and presented on a single page.

So far specifying Agent-Based Models and implementing them as an Agent-Based Simulation (ABS) is done using object-oriented methods like UML and object-oriented programming languages like Java. The reason for this is that until now the concept of an agent was always understood to be very close to, if not equals to - which it is not - the concept of an object. Therefore, the reasoning goes, object-oriented methods and languages should fit themselves naturally to specify and implement agent-based simulations. In this PhD Thesis we fundamentally challenge this assumption by investigating how Agent-Based Models and Simulations can be specified and implemented using pure functional methods and programming in Haskell and what the benefits are. We will show that the implicit assumption that an Agent is *about equal to* an Object is not correct and leads to many implicit assumptions in object-oriented implementations of Agent-Based Simulation (ABS). When implementing ABS in Haskell these implicit assumptions become explicit and challenge the fundamental assumptions about ABS and Agents. We present these implicit assumption in an explicit way by approaching it through programming, type-theory and category-theory to further deepen the concepts and methods in the field of Agent-Based Modelling & Simulation. We also think that the major benefit of implementing ABS in Haskell is the potential for an unprecedented approach of formal validation & verification of an Agent-Based Model and its implementation. Due to the declarative nature of pure functional programming in Haskell it is possible to implement an EDSL for ABS which ideally results in code which looks like specification thus closing the gap between specification and implementation because the specification is already the code. For validation we want to pursue testing through QuickCheck.

In this report I discuss the research conducted to far, present the open problems together with an in-depth literature review. An outline for the research of the following 2 years is given and the aims.

1 Introduction

The central aspect of my PhD is centred around the main question of *How can Agent-Based Simulation be done using pure functional programming and what are the benefits and disadvantages of it?*. So far functional programming has not got much attention in the field of ABS and implementations always focus on the object-oriented approach. We claim, based upon the research of the first year that functional programming is very well suited for ABS and that it offers methods which are not directly possible and only very difficult to achieve with object-oriented programming.

We claim that to build large and complex agent-based simulations in functional programming is possible using the functional reactive programming (FRP) paradigm. We applied FRP to implementing ABS and developed a library in Haskell called FrABS. We implemented the quite complex model SugarScape from social simulation using FrABS and proofed by that, that applying FRP to ABS enables ABS to happen in pure functional programming.

After having shown how agent-based simulation can be done in functional programming we claim that the major benefit of using it enabled a new way of *verification & validation* in agent-based simulation.

Due to the declarative nature of pure functional programming it is an established method of implementing an EDSL to solve a given problem in a specific domain. We followed this approach in FrABS and developed an EDSL for ABS in pure functional programming. Our intention was to develop an EDSL which can be used both as specification- and implementation-language. We show this by specifying all the rules of SugarScape in our EDSL.

Due to the lack of implicit side-effects and the recursive nature of pure functional programming we claim that it is natural to apply it to a novel method we came up with: MetaABS, which allows recursive simulation.

Finally having such an EDSL at hand this will allow us to reason about the programs. This will be applied to specify and reason about the dynamics and emergent properties of decentralized bilateral trading and bartering in agent-based computational economics (ACE) and social simulations like SugarScape.

Disadvantages - although the lack of side-effects is also a benefit, it is also a weakness as all data needs to be passed in and out explicitly - indirection due to the lack of objects & method calls. - When not to use it: - if you are not familiar with functional programming - when you can solve your problem without programming in a Tool like NetLogo, AnyLogic,... - when you don't need to reason about your program

1.1 Problems of ABS in OO

- Objects don't compose - implicit state, change through effectful computations - blurring of fundamental difference between agent and object: an agent is a metaphor, it is much more than an object. an object is: a uniquely identifieable compound of functions (=methods) and data

1.2 Problems of ABS in general

Specification: how is my model specified? Verification: does my implementation really match my specification? Validation: how to connect the results to the hypothesis? are the emergent properties the ones anticipated? if it is completely different why? note: we always MUST HAVE a hypothesis regarding the outcome of the simulation, otherwise we leave the path of scientific discovery. But we must admit that sometimes it is extremely hard to anticipate *emergent patterns*. But anyway there must be *some* hypothesis regarding the dynamics of the simulation. The idea is to express hypotheses directly in the program using QuickCheck and then let the simulation verify it

1.3 Functional approach to Agent-Based Modelling & Simulation

Because we left the path of OO and want to develop a completely different method we have fundamentally two problems to solve in our functional method: 1. Specifying the Agent-Based Model (ABM): Category-Theory, Type-Theory, EDSL: all this clearly overlaps with the implementation-aspect because the theory behind pure functional programming in Haskell is exactly this. This is a very strong indication that functional programming may be able to really close the gap between specification and implementation in ABS. 2. Implementing the ABM into an Agent-Based Simulation (ABS): building on FRP paradigm

1.3.1 Challenges

- how is an agent represented? - how do agents pro-actively act? - how do agents interact? - how is the environment represented? - how can agents act on the environment? - how to handle structural dynamism (creation and removal of agents)?

1.3.2 Expected benefits

1. By mapping the concepts of ABS to Category-Theory and Type-Theory we gain a deeper understanding of the deeper structure of Agents, Agent-Models and Agent-Simulations. 2. The declarative nature of pure functional programming will allow to close the gap between specification and code by designing an EDSL for ABS in Haskell building on the previously derived abstractions in Category-Theory and Type-Theory. The abstractions and the EDSL implementation will then serve as a specification tool and at the same time code. 3. The pure functional nature together with the EDSL and abstractions in Category- & Type-Theory allow for a new level of formal verification & validation using a combination of mathematical proofs in Category- & Type-Theory, algebraic reasoning in the EDSL and model-checking using Unit-Tests and QuickCheck. The expectation is that this allows us to formally specify hypotheses about expected outcomes about the dynamics (or emergent patterns) of our simulations which then can be verified.

1.4 Field of Application

Agent-Based Modelling & Simulation is a method and tool and thus always applied in a very specific domain in which phenomenon are being researched which can be mapped to ABS. For our PhD we picked the field of Agent-Based Computational Social Sciences (ACS) with slight influences from Agent-Based Computational Economics (ACE). The reason for this is that ACS was one of the first fields to adopt ABS in their research on artificial societies and is still a strong driving force behind the application of ABS. ACE is about the same age as ACS but is not yet nearly as established in Economics as ACS is in the Social Sciences (TODO: can i really back up my claims?). ACS draw upon findings of ACE in the SugarScape Model where Agents engage in bilateral decentralized bartering. The author claims that both fields are highly relevant for the future. Economists start realizing that more heterogenous models need to be studied which can only be done using ABS. Our primary field of application will be ACS but we will sometimes draw upon ACE when appropriate e.g. in SugarScape. By applying our new method to these fields we hope to bring forward a paradigm-shift which allows to better understand the concepts of ABS and to have more powerful tools for verification & validation at hand.

TODO: what are the aims of ACS and ACE?

1. Initial Use-Case: SugarScape Model as in "Growing Artificial Societies: Social Science From the Bottom Up" 2. Supplementary Material: "Generative Social Science: Studies in Agent-Based Computational Modelling" 3. Verification & Validation Use-Case: "Agent_Zero: Toward Neurocognitive Foundations for Generative Social Science"

2 Literature Review

Literature Review - trichter: mit den 3 themen beginnen und dann runterbrechen und ins detail gehen, bis der gap gefunden wurde

2.1 Agent-Based Modelling & Simulation

wooldridge, will clinger, hewitt

TODO: baas: emergence, hierarchies and hyperstructures TODO: [**?**]

2.1.1 Social Simulation

The SugarScape model [**?**] is one of the most influential models of agent-based simulation in the social sciences. The book heavily promotes object-oriented programming (note that in 1996 oop was still in its infancy and not yet very well understood by the mainstream software-engineering industry). We ask how it can be done using pure functional programming paradigm and what the benefits and limits are. We hypothesize that our solution will be shorter (original reported 20.000 LOC), can make use of EDSL thus making it much more expressive, can utilize QuickCheck for a completely new dimension of

model-checking and debugging and allows a very natural implementation of MetaABS (see Part III) due to its recursive and declarative nature.

TODO [?] TODO [?]

2.1.2 Computational Economics

[?] gives a broad overview of agent-based computational economics (ACE), gives the four primary objectives of it and discusses advantages and disadvantages. She introduces a model called *ACE Trading World* in which she shows how an artificial economy can be implemented without the *Walrasian Auctioneer* but just by agents and their interactions. She gives a detailed mathematical specification in the appendix of the paper which should allow others to implement the simulation.

- Artificial agent-based economies: [?], [?], [?], [?], [?] - Artificial agent-based markets: [?], [?] - Agent-Based Market Design: [?], [?]

market-microstructure: [?], [?]

Basics of Economics [?], [?]

2.2 Verification & Validation

model checking and reasoning by quickcheck: [?], [?]

Verification/Reasoning ist einer der größten Pluspunkte von rein funktionaler Programmierung, da durch den deklarativen Stil und das Fehlen von Sideeffects und Globalen Daten equational/algebraic/inductive Reasoning betrieben werden kann. Hier habe ich noch garnichts dazu gemacht, aber sollte mit den oben genannten Ideen sicherlich interessant werden - ein interessantes Paper von Graham Hutton (für den ich übrigens dieses Semester ein Tutor in seiner Haskell-Laborübung bin) gibt interessante Richtungen für Reasoning vor: http://dl.acm.org/citation.cfm?id=968579

[ ] deadlock: when messages need to be exchanged but mutual waiting [ ] silence: no more message exchange [ ] protocoll: ensure happens before / sequences (like necessary for 2D prisoner dilemma)

2.3 Functional Programming

all FRP, quickcheck, arrows, monads, wadler, hughes

2.3.1 EDSL

EDSL steht für Embedded Domain Specific Language d.h. man implementiert in Haskell eine Art von 'Spezifikations-Sprache' für eine spezielle Domain (z.b. ABM/S), die - dank der rein funktionalen, deklarativen Natur von Haskell - auch gleichzeitig Haskell Code ist - der Unterschied zwischen Spezifikation und Implementierung verschwindet dann (idealserweise). In diese Richtung arbeite ich erst seit kurzem, durch die Umsetzung von ABS/M mit Yampa. Yampa ist ebenfalls eine EDSL um funktional-reaktive Systeme zu beschreiben/implementieren, ich

werde auf dieser EDSL aufsetzen und sie um ABS/M erweitern - so zumindest der Plan. Dann habe ich die theoretische Grundlage von FRP, auf die ich dann auch theorie von ABS/M (z.b. Actor Semantics) setzen kann und somit zum nächsten Punkt komme:

2.3.2 General principles

2.3.3 Structuring

Monads Arrows Continuations

2.3.4 Paradigm: FRP

TODO: why Yampa? There are lots of other FRP-libraries for Haskell. Reason: in-house knowledge (Nilsson, Perez), start with *some* FRP-library to get familiar with the concept and see if FRP is applicable to ABS. TODO: short overview over other FRP-libraries but leave a in-depth evaluation for further-research out of the scope of the PhD as Yampa seems to be suitable. One exception: the extension of Yampa to Dunai to be able to do FRP in Monads, something which will be definitely useful for a better and clearer structuring of the implementation. TODO: Push vs. Pull

TODO: describe FRP

TODO: 1st year report Ivan: "FPR tries to shift the direction of data-flow, from message passing onto data dependency. This helps reason about what things are over time, as opposed to how changes propagate". QUESTION: Message-passing is an essential concept in ABS, thus is then FRP still the right way to do ABS or DO WE HAVE TO LOOK AT MESSAGE PASSING IN A DIFFERENT WAY IN FRP, TO VIEW AND MODEL IT AS DATA-DEPENDENCY? HOW CAN THIS BE DONE? BUT: agent-relations in interactions are NEVER FIXED and always completely dynamic, forming a network. The question is: is there a mechanism in which we have explicit data-dependency but which is dynamic like message-passing but does not try to fake method-calls? maybe the conversations come very close

2.4 Category- & Type-Theory

Category-Theory [**?**] [**?**]

include paper on arrows my hughes apply category theory to agent-based simulation: how can a ABS system itself be represented in category theory and can we represent models in this category theory as well? ADOM: Agent Domain of Monads: https://www.haskell.org/communities/11-2006/html/report.html develop category theory behind FrABS: look into monads, arrows

2.5 Identifying the Gap

- Functional programming in this area exists but only scratches the surface and focus only on implementing agent-behaviour frameworks like BDI. An in-

depth treatise of Agent-Based Modelling and implementing an Agent-Based Simulation in a pure functional language has so far never been attempted.

- There basically exists no approach to Agent-Based Modelling & Simulation in terms of Category-Theory and Type-Theory

- Verification is an issue in ABS as they are very often described in natural language and supplemented with a few formulas. This leads to implementation-errors, e.g. Gintis Bartering-Paper, and results become hard to reproduce. Such errors become a threatening problem when simulation-results are used in decision making e.g. economics, policy-making, ...

- Validation is basically an untouched topic in ABS: models are formulated, a few hypotheses are formulated, the model is implemented and run, then the results are checked against the hypotheses. What the field of ABS needs is an in-depth discussion on how to rigorously validate a model. Validation is of course only as strong as the verification part: if the implementation is wrong anyway then we can not rely on anything (from false comes nothing)

3 Aims and Objectives

- developing a category- & type-theoretical view on Agent-Based Modelling & Simulation which will -¿ 1. give a deeper insight into the structure of agents, agent-models and agent-based simulation -¿ 2. serves as the basis for the pure functional implementation -¿ 3. serves as a high-level specification tool for agent-models

- implementing a library called FrABS based upon the FRP paradigm which allows to specify Agent-Based Models in an EDSL and run them

- Verification: closing the gap between specification and implementation through the category- & type-theoretical view and the EDSL

- Validation: formalizing hypotheses and reasoning about dynamics and expected outcomes of the simulation

Define 5 general research questions for each Research-Context

- 2 related to FP
- 1 related to integration of FP to ABM/S
- 2 related to ABM/S

3.1 Validation

Semantics for FrABS and our EDSL to reason about the results: are they reasonable? do they match the theory? if yes why? if not why not? - Can we define semantics for the EDSL to do reasoning about ABS in general? - How can we reason about ABS in general in pure functional programming? - dynamics - emergent properties - deadlocks - silence (no messages/agent-agent communication and interaction) - define semantics of FrABS based on semantics of FRP and Actors - what is emergence in ABS and how can we reason about it? -

identify emergent properties: equilibrium, behaviour on macroscale not defined on micro, chaos,... - can we anticipate emergent properties / dynamics just by looking at the code and reason about it? - can emergence in ABS be formalized? - hypothesis: it may be possible through functional programming because of its dual nature of declarative EDSL which awakens to a process during computation - what is the relation between emergence and computation? we need change over time (=computation) for emergence

- Can we reason about the dynamics and equilibria of agent-based models of decentralized bilateral trading & bartering?

4 Work To Date

TODO: Describe the research work carried out during this stage of the PhD and the outcomes. A literature review must be included. Then, as appropriate according to the PhD project, this section can also include theoretical and/or experimental methods, presentation and discussion of results, etc. In the case that papers have been submitted or published within the year of the review, this section can be shorter and focused on discussing the outcomes from those papers within the wider context of the PhD programme of study (papers to be included in the appendix).

4.1 Papers Submitted

Update-Strategies in ABS TODO: attach as appendix

A foundational paper

4.2 Paper Drafts

Programming Paradigms and ABS TODO: attach as appendix

In this work I investigated the suitability of three fundamentally different programming pardigms to implement an ABS. The paradigms I looked at was object-oriented using Java, pure functional using Haskell and multi-paradigm functional using Scala with the Actors library. It is important to note that at this point I didn't use FRP as underlying paradigm in Haskell TODO: would this have changed my final conclusion on its suitability?

STM: the really unique thing which is ONLY possible in pure functional programming is composition of concurrency. TODO: cite Tim Sweeny Actors in Scala

4.2.1 Papers in Progress

Recursive ABS Give each Agent the ability to run the simulation locally from its point of view do anticipate its actions and change them in the future thus introducing a meta-level in the simulation, from which the method derives its name.

- TODO: i have only the idea but am lacking a theory or hypothesis for its use

- meta need a kind of decision error measure to distinguish between various meta-simulations. also we need a mechanism to sample the decision space =¿ it can be considered to be an optimization technique.

Problems

- Definition of a recursive, declarative description of the Model.
- Perfect information about other agents is not realistic and runs counter to agent-based simulation (especially in social sciences) thus an Agent needs to be able to have local, noisy representations of the other agents.
- Local representation of other agents could be captured by Hidden Markov Models: observe what other agents do but have hidden interpretation of their internal state - these internal state-representations can be different between the local and the global version whereas the agent learns to represent the global version as best as possible locally.
- Infinite regress is theoretically possible but not on computers, we need to terminate at some point

Interpretation: It can be regarded as a Model of Free Will in ABS, which allows learning in an ABS environment in a new way - look on the section of interpretation. Application: hypothesis: allows to model social and psychological phenomena like free will. Mostly in social sciences, maybe also in economics. Investigate SugarScape, PrisonersDilemma and ACE Trading World

TODO: question: what is the meaning of an entity running simulations? it strongly depends on the context: in ACE it may be search for optimization behaviour, in Social Simulation it may be interpreted as a kind of free will

Research Questions

1. How does deep regression influence the dynamics of a system? Hypothesis: TODO
2. How do the dynamics of a system change when using perfect information or learning local information? Hypothesis: TODO
3. Is a hidden markov model suitable for the local learning? Hypothesis: TODO
4. How can MetaABS best be implemented? Hypothesis: implementing a MetaABS EDSL in a pure functional language like Haskell, should be best suited due to its inherent recursive, declarative nature, which should allow a direct mapping of features of this paradigm to the specification of the meta-model

- functional programming perfect. standard toolkits (anylogic, netlogo, repast) are not capable of doing this - extend my existing EDSL for functional reactive agent-based simulation & modelling (FrABS/M) with recursive functionality

Related Research: TODO: [**?**] cite paper of recursive simulation: [ ] military simulation, [ ] not explicitly abs, [ ] implemented in c++, [ ] deterministic models seem to benefit significantly from using recursions of the simulation for the decision making process. when using stochastic models this benefit seems to be lost

Functional Reactive ABS

4.2.2 Software

Lots of prototyping: Heroes & Cowards, SIRS & Schelling Segregation in Java, Haskell and Scala Parallelism and Concurrency in Haskell

FrABS: SugarScape Model as use-case no.1. TODO: available on github

4.3 Talks

presenting the ideas of my Update-Strategies paper at the IMA - seminar day presenting my FrABS ideas to the FP-Lab Group at the FPLunch

4.4 Courses

- Computer Science PGR Introductory Seminar 5 Dates - Attended Midland Graduate School 2017 from 9-13 April in Leicester. Attended courses on Denotational Semantics, Naïve Type Theory and Testing with Theorem Provers. - Graduate School: -¿ Nature of the doctorate and the supervision process, 15th November 2016 (9:30 - 12:00) -¿ Presentation skills for researchers (all disciplines), 27th Jan 2017 (9:30 - 15:30) -¿ Planning your research, 20th Feb 2017 (9:30 - 13:00) -¿ Getting into the habit of writing, 23th Feb 2017 (9:30 - 12:30) - Tradition of Critique Lecture series, Monday 29th September - Monday 8th December (18:00 - 20:00)

5 Conclusions

TODO: Provide a succinct account of the conclusions from the report, stating clearly the research questions that have been identified during this stage of the PhD and the progress so far towards addressing those questions.

6 Future Work Plan

TODO: A future work plan that is consistent with the progress to date, stating clearly the research question(s) to be addressed during the next year of the PhD.

TODO: gantt chart!

6.1 TODOs

out of this i will build the gantt chart for the next 12 months+

6.1.1 Category Theory

develop category theory behind FrABS: look into monads, arrows
category theory foundations (monads, arrows)

6.1.2 Implementation and Software-Engineering

implement chapter 4 of sugarscape implement chapter 5 of sugarscape use monadic or arrowized programming for structuribg the siftware implement schelling segregation in recursiveABS and report results

FrABS: SugarScape 1st prototype: pure-functional implemented, no category-theory/type-theory applied 2nd prototype: category-theory/type-theory applied: clean monadic / arrowized programming applied

Agent_Zero 1st prototype: implemented the book, based upon FrABS

6.1.3 Verification and Validation

look into QuickCheck to test and verificate FrABS. start with SIRS (quickcheck, isabelle, agda?), recursive simulation

6.1.4 Papers

paper 2: recursive ABS paper 3: FrABS - Towards pure functional programming in ABS paper 4: Towards category theory in ABS paper 5: verification and validation in ABS with pure functional programming

6.1.5 Reading

read "Writing For Computer Science" read "Agent_Zero" read "category theory for the sciences"

6.2 Concept of an Agent

an agent is not an object but when implementing ABS in oo then it is tempting to treat an agent like that. when implementing it in a pure functional language like haskell, this temptation cannot arise which creates a different view on agents.

7 Appendix

Material that is complementary to the main body of the report can be included in an appendix. For externally sponsored students, if a report has been submitted to the sponsor during the year of the review, the report should be included

in the appendix (a copy of the report can be supplied by the PGR coordinator). The appendix should include a list of training courses (including dates, duration, etc.) taken by the student during the year and other relevant research activities such as given seminars, attendance and presentations to conferences, etc. The appendix could also include material that is supplementary to the main body of the report such as: description of data sets, detailed experimental results, papers that have been submitted or published, etc.

TODO: programming-paradigms directly as PDF TODO: update-strategies directly as PDF TODO: recursive-ABS directly as PDF IF OK