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

Jonathan Thaler
jonathan.thaler@nottingham.ac.uk

May 7, 2017

Abstract

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 apply naturally to specify and implement agent-based simulations. In this thesis we fundamentally challenge this assumption by investigating how Agent-Based Models and Simulations (ABS) can be specified and implemented using pure functional methods and what the benefits are. We show how ABS can be implemented in the pure functional language Haskell and develop a basic underlying theoretical framework in Category-Theory and Type-Theory which allows to view and specify ABS from a new point-of-view. Having these tools at hand the major benefit using them is the potential for an unprecedented approach to validation & verification of an ABS. First: 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 is equals to specification thus closing the gap between specification and implementation. Second: validation becomes possible by formulating hypotheses directly in code using the EDSL and QuickCheck and then verify these hypotheses. Third: Category-Theory allows to view ABS from a high-level view thus giving a different insight and allows to specify ABM in a new way.

Contents

1 Introduction **4**
- 1.1 Background . . . 4
- 1.2 Motivation . . . 5

2 Literature Review **9**
- 2.1 Agent-Based Modelling & Simulation . . . 9
- 2.2 Fields of Application . . . 11
  - 2.2.1 Social Simulation . . . 11
  - 2.2.2 Computational Economics . . . 11
- 2.3 Alternative to object-orientation . . . 13
  - 2.3.1 The Actor Model . . . 13
  - 2.3.2 Functional Programming . . . 14
  - 2.3.3 Haskell . . . 18
  - 2.3.4 Structuring . . . 20
  - 2.3.5 Paradigm: FRP . . . 21
  - 2.3.6 EDSL . . . 24
  - 2.3.7 Category- & Type-Theory . . . 24
- 2.4 Verification & Validation of ABS . . . 25

3 Aims and Objectives **27**
- 3.1 Identifying the Gap . . . 31
  - 3.1.1 Validation . . . 32

4 Work To Date **33**
  - 4.0.1 Papers Submitted . . . 33
  - 4.0.2 Paper Drafts . . . 33
  - 4.0.3 Talks . . . 35
  - 4.0.4 Courses . . . 35

5 Conclusions **36**

6 Future Work Plan **37**
  - 6.0.1 TODOs . . . 37
  - 6.0.2 Concept of an Agent . . . 38

6.0.3 Milestones 38
6.0.4 Time-Line 39
6.0.5 ToDo 39
6.0.6 1st Year: Groundwork 39
6.0.7 2nd Year: Main Work 41
6.0.8 3rd Year: Finalizing, Publishing & Writing 42
6.0.9 4th Year: Pending Period 43
6.1 Appendix 49

Chapter 1

Introduction

1.1 Background

TODO: start with simulation in general?

Agent-based simulation and modelling (ABS) is a method for simulating the emergent behaviour of a system by modelling and simulating the interactions of its sub-parts, called agents (TODO: cite willenskys book on ABS intro, need many more citations). Examples for an ABS is simulating the spread of an epidemic throughout a population or simulating the dynamics of segregation within a city. Central to ABS is the concept of an agent who needs to be updated in regular intervals during the simulation so it can interact with other agents and its environment. We understand ABS as a method of modelling and simulating a system where the global behaviour may be unknown but the behaviour and interactions of the parts making up the system is known. Those parts, called agents, are modelled and simulated out of which then the aggregate global behaviour of the whole system emerges. So the central aspect of ABS is the concept of an agent which can be understood as a metaphor for a pro-active unit, situated in an environment, able to spawn new agents and interacting with other agents in a network of neighbours by exchange of messages [51] (TODO: more citations).

We informally assume the following about our agents [51]:

- They are uniquely addressable entities with some internal state.
- They can initiate actions on their own e.g. change their internal state, send messages, create new agents, kill themselves.
- They can react to messages they receive with actions as above.
- They can interact with an environment they are situated in.

It is important to note that we focus our understanding of ABS on a very specific kind of agents where the focus is on communicating entities with individual, localized behaviour from out of which the global behaviour of the system

emerges. This explicit distinction is necessary as to not confuse our concept of *agent* with those of the field of Multi Agent Systems (MAS). ABS and MAS influence each other in the way that the basic concept of an agent is very similar but the areas of application are fundamentally different. MAS is primarily used as an engineering approach to organize large software-systems (TODO: cite book by weiss on MAS) where ABS is primarily used for the simulation of collective behaviour of agents to gain insight into the dynamics of a system.

ABS is a method and thus always applied in a very specific domain in which phenomenon are being researched which can be mapped to collective behaviour. This implies that we need to select a specific domain in which we want to advance the methodology of ABS. For our PhD we picked the field of Agent-Based Computational Social Sciences (ACS) which is traditionally a highly interdisciplinary field [4], drawing on lots of different other areas like economics, epidemology, genetics, neurocognition, biology. Simulation in the social sciences is still a young field, having emerged in the 90s but is constantly growing and gaining significance [3]. It offers the new approach of *generative* social sciences in which one tries to generate phenomenon by e.g. constructing artificial societies to test hypotheses. This is in opposition to the classical inductive (finding patterns in empirical data) and deductive (proving theorems) sciences and thus regarded as the way of doing science [3].

TODO: write about ABS as a new tool and generative as opposed to the classical inductive and deductive sciences. major sources: TODO [5] TODO [19] TODO [20]

The most influential source on the generative social sciences can be regarded the work of Epstein and Axtell in [22] in which they indeed create an artificial society and connect observations in their simulation to phenomenon of real-world societies. They later added more research to this in [20] and most recently [21] in which Epstein tries to approach the generative social sciences from a neurocognitive approach. All three works will serve as our primary sources and use-cases in which we will apply our research and will be treated more in-depth in the literature-review section. Another field we are particularly interested in, the simulation of decentralized bilateral bartering, which belongs to the field of Agent-Based Computational Economics (ACE) [48] is covered already in the artifical society of [22]. This lucky coincidence let us approach both these very special fields together and apply them as additional use-cases for developing our new methods as presented in the next section.

1.2 Motivation

The challenges one faces when specifying and implementing an ABS are manifold:

- 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?

Epstein & Axtell explicitly advocate object-oriented programming in [22] as "a particularly natural development environment for Sugarscape specifically and artificial societies generally." and report about 20.000 lines of code which includes GUI, graphs and plotting. They implemented their Sugarscape software in Object Pascal and C where they used the former for the Agents and the latter for low-level graphics [5]. Axelrod [3] recommends Java for experienced programmers and Visual Basic for beginners. Up until now most of ABS seems to have followed this suggestion and are implemented using programming languages which follow the object-oriented imperative paradigm. In this paradigm the program consists of an implicit, global mutable state which is spread across multiple objects. Although the control-flow is explicit (with the exception of parallelism) where methods of objects are being called from other methods in objects, the data-flow is mostly implicit due to side-effects on the global mutable data. It is established knowledge that the effort to manage implicit global state which can be changed through effectful computations (side-effects) increases exponentially with the complexity of the system one implements (TODO: find a citation). This can only be held at bay through the proper usage of patterns [25] and years of working-experience (TODO: find a citation). To put it short: proper object-oriented programming *is hard* where the difficulty arises from how to split up a problem into objects and their interactions. Still if one masters the technique of object-oriented program-design and implementation, due to the implicit global mutable state bugs due to side-effects are the daily life of a programmer (TODO: find a citation). Another serious problem of object-oriented implementations is the blurring of the fundamental difference between agent and object - an agent is first of all a metaphor and *not* an object. In object-oriented programming this distinction is obviously lost as in such languages agents are implemented as objects which leads to the inherent problem that one automatically reasons about agents in a way as they were objects - agents have indeed become objects in this case. TODO: what is the problem behind this?

In [4] Axelrod reports the vulnerability of ABS to misunderstanding. Due to informal specifications of models and change-requests among members of a research-team bugs are very likely to be introduced. He also reported how difficult it was to reproduce the work of [2] which took the team four months which was due to inconsistencies between the original code and the published paper. The consequence is that counter-intuitive simulation results can lead to weeks of checking whether the code matches the model and is bug-free as reported in [3]. The same problem was reported in [34] which tried to reproduce the work of Gintis [27]. In his work Gintis claimed to have found a mechanism in bilateral decentralized exchange which resulted in walrasian general equilibrium without

the neo-classical approach of a tatonement process through a central auctioneer. This was a major break-through for economics as the theory of walrasian general equilibrium is non-constructive as it only postulates the properties of the equilibrium [14] but does not explain the process and dynamics through which this equilibrium can be reached or constructed - Gintis seemed to have found just this process. Ionescu et al. [34] failed and were only able to solve the problem by directly contacting Gintis which provided the code - the definitive formal reference. It was found that there was a bug in the code which led to the "revolutionary" results which were seriously damaged through this error. They also reported ambiguity between the informal model description in Gintis paper and the actual implementation. This lead to a research in a functional framework for agent-based models of exchange as described in [10] which tried to give a very formal functional specification of the model which comes very close to an implementation in Haskell. This was investigated more in-depth in the thesis by [23] who got access to Gintis code of [27]. They found that the code didn't follow good object-oriented design principles (all was public, code duplication) and - in accordance with [34] - discovered a number of bugs serious enough to invalidate the results. This reporting seems to confirm the above observations that proper object-oriented programming is hard and if not carefully done introduces bugs. The author of this text can report the same when implementing [22]. Although the work tries to be much more clearer in specifying the rules how the agents behave, when implementing them still some minor inconsistencies and ambiguities show up due to an informal specification. The fundamental problems of these reports can be subsumed under the term of verification which is the checking whether the implementation matches the specification. Informal specifications in natural language or listings of steps of behaviour will notoriously introduce inconsistencies and ambiguities which result in wrong implementations - wrong in the way that the *intended* specification does not match the *actual* implementation. To find out whether this is the case one needs to verify the model-specification against the code. Verification is a required approach for software systems which needs to meet high standards e.g. Medicine, Power Plant (TODO: cite) but has so far not been undertaken for ABS (TODO: is this really the case?, look into literature)

As ABS is almost always used for scientific research, producing often break-through scientific results as pointed out in [4], these ABS need to be *free of bugs*, *verified against their specification*, *validated against hypotheses* and ultimately be *reproducible*. One of the biggest challenges in ABS is the one of validation. In this process one needs to connect the results/dynamics of the simulation to initial hypotheses e.g. *are the emergent properties the ones anticipated? if it is completely different why?*. It is important to understand that we always *must have* a hypothesis regarding the outcome of the simulation, otherwise we leave the path of scientific discovery. We must admit that sometimes it is extremely hard to anticipate *emergent patterns* but still there must be *some* hypothesis regarding the dynamics of the simulation otherwise it is just guesswork.

In the concluding remarks of [4] Axelrod explicitly mentions that the ABS community should converge both on standards for testing the robustness of

ABS and on its tools. However as presented above, we can draw the conclusion that there seem to be some problems the way ABS is done so far. We don't say that the current state-of-the-art is flawed, which it is not as proved by influential models which are perfectly sound, but that it always contains some inherent danger of embarrassing failure. These observations lead us to posing two fundamental directions which are the basis of the motivation of our thesis.

1. **Alternative to object-orientation** - Is there an alternative to the established object-oriented approaching ABS which offers an explicit data-flow, reduces the bugs due to global mutable state and does not lead to the blurring of agent and object?

2. **Verification & Validation of ABS** - Is there a way for formal specification which is still readable and does not fall back to pure mathematics? Is there a way to formally specify hypotheses about the simulation which are then checked automatically against the results?

As will become evident from the literature-review we advocate pure functional programming in Haskell and the type-theoretic and category-theoretic foundations as a solution to the questions posed. The usage of pure functional programming in ABS is also a strong motivation by itself as it hasn't been researched yet and deserves a thorough treatment on its own. Surprisingly there exist hardly any attempts on implementing ABS in pure functional programming as will become clear in the literature-research. Maybe this can also be seen as a hint that ABS lacks a level of formalism which we hope to repair with our thesis.

So put short the motivations is a twofold direction, referring to each other in a circular way. First, pure functional programming has not been researched for implementing and specifying ABS so far. Second, the current state-of-the-art seems to be susceptible to flaws and bugs due to the lack of powerful verification. Combining both issues forms the very basic motivation of our thesis: use pure functional programming and its underlying theoretical framework to develop new methods for specifying, implementing, verifying and validating ABS to create simulations which are more reliable, reproducible and communicatable.

Chapter 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

Agent Models are NOT specific to any programming language implementation but should in theory be implementable in all languages which support the required primitives of the model or which allows the primitives of the model to be mapped to primitives of the language. Of course this says nothing about how well a language is suited to implement a given agent model and how readable and natural the mapping and implementation is.

Start from wooldridge 2.6 (look at the original papers which inspired the 2.6 chapter) and weiss book and the original papers those chapter is based upon. Look into denotational semantics of actor model. Look also in functional models of czesar ionescu. why: this is the major contribution of my thesis and is new knowledge. Must find intuitive, original and creative approach.
From functional agent models (e.g. Wooldridge) to implementation of ACE in Haskell (this should go into research-proposal introduction)

TODO: [**?**]

TODO: [**?**]

wooldridge, will clinger, hewitt

TODO: baas: emergence, hierarchies and hyperstructures TODO: [6]

This method was selected because Scala is an object-oriented functional programming language and has a powerful library included which implements the actor-model. Because actors and agents are closely related this is an obvious method to follow.

The Actor-Model, a model of concurrency, has been around since the paper [**?**] in 1973. It was a major influence in designing the concept of Agents and

although there are important differences between Actors and Agents there are huge similarities thus the idea to use actors to build agent-based simulations comes quite natural. Although there are papers around using the actor model as basis for their ABMS unfortunately no proper theoretical treatment of using the actor-model in implementing agent-based simulations has been done so far. This paper looks into how the more theoretical foundations of the suitability of actor-model to ABMS and what the upsides and downsides of using it are.

`http://www.grids.ac.uk/Complex/ABMS/`

[**?**] describes in chapter 3.3 a naive clone of NetLogo in the Erlang programming language where each agent was represented as an Erlang process. The author claims the 1:1 mapping between agent and process to "be inherently wrong" because when recursively sending messages (e.g. A to B to A) it will deadlock as A is already awaiting Bs answer. Of course this is one of the problems when adopting Erlang/Scala with Akka/the Actor Model for implementing agents *but it is inherently short-sighted to discharge the actor-model approach just because recursive messaging leads to a deadlock*. It is not a problem of the actor-model but merely a very problem with the communication protocol which needs to be more sophisticated than [**?**] described. The hypothesis is that the communication protocol will be in fact *very highly application-specific* thus leading to non-reusable agents (across domains, they should but be re-usable within domains e.g. market-simulations) as they only understand the domain-specific protocol. This is definitely NOT a drawback but can't be solved otherwise as in the end (the content of the) communication can be understand to be the very domain of the simulation and is thus not generalizable. Of course specific patterns will show up like "multi-step handshakes" but they are again then specifically applied to the concrete domain.

[35] discuss using functional programming for discrete event simulation (DES) and mention the paradigm of Functional Reactive Programming (FRP) to be very suitable to DES. We were aware of the existence of this paradigm and have experimented with it using the library Yampa, but decided to leave that topic to a side and really keep our implementation clear and very basic.

The amount of research on using the pure functional paradigm using Haskell in the field of ABS has been moderate so far. Though there exist a few papers which look into Haskell and ABS [16], [46], [35] they focus primarily on how to specify agents. A library for Discrete Event Simulation (DES) and System Dynamics (SD) in Haskell called *Aivika 3* is described in [44]. It also comes with very basic features for ABS but only allows to specify simple state-based agents with timed transitions. This papers is investigating Haskell in a different way by looking into its suitability in implementing update-strategies in ABS, something not looked at in the ABS community so far, presenting an original novelty.

There already exists research using the Actor Model [1] for ABS in the context of Erlang [49], [17], [18], [43] but we feel that they barely scratched the surface. We want to renew the interest in this direction of research by incorporating Scala with using the Actor-library in our research because we will show that one update-strategy maps directly to the Actor Model.

2.2 Fields of Application

2.2.1 Social Simulation

look closer at the contents of the 3 major books: SugarScape, Generative, Agent_Zero

The SugarScape model [22] 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 [30] TODO [40]

2.2.2 Computational Economics

[48] 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: [48], [27], [28], [24], [10] - Artificial agent-based markets: [38], [15] - Agent-Based Market Design: [39], [12]

market-microstructure: [**?**], [8]

Basics of Economics [11], [36]

look into computable economics book: `http://www.e-elgar.com/shop/computable-economics`

TODO: the reading should pull out the essence of what types of ACE there are and what features each type has (continuous/discrete time, complex agent communication, equilibriua, networks amongst agents,...)

NOTE: I REALLY need to work out what is special in ACE? what is the unique property of ACE AS compared to other ABM/S? Conjecture: equilibrium of dynamics is the central aspect. `http://www2.econ.iastate.edu/tesfatsi/ace.htm`

[**?**] Agent-based modeling and economic theory: where do we stand? - Ballot, Mandel, Vignes

[**?**] Agent-based Computational Economics. A Short Introduction - Richiardi

[**?**] Agent-based computational economics: a constructive approach to economic theory - tesfatsion

[**?**] Introduction to computer science and economic theory - blume, easley, kleinberg
[**?**] agent-based computational economics - tesfatsion

The book [**?**] is a critique of classic economics with the triple of rational agents, "average" inidividuum, equilibrium theory. Although it does not mention ACE it can be seen as an important introduction to the approach of ACE as it introduces many important concepts and views dominant in ACE. Also ACE can be seen as an approach of tackling the problems introduced in this book:

page 6: "the view of economy is much closer to that of social insects than to the traditional view of how economies function."
page 7: "... main argument that it is the interaction between individuals that is at the heart of the explanation of many macroeconomic phenomena..."
page 15: "problem of equilibrium is information"
page 21: "the theme of this book will be that the very fact individuals interact with each other causes aggregate behaviour to be different from that of individuals"

TODO: [**?**] TODO: [**?**] TODO: how economists can get a life tesfatsion

"[...] computational modelling of economic processes (including whole economies) as open-ended dynamic systems of interacting agents." Leigh Tesfatsion

TODO: look into the models of agents dominant in ACE. They seem to be more of reactive, continuous nature

depending on the model ACE

properties

- Discrete entities with own goals and behaviour
- Not necessarily own thread of control
- Capable to adapt
- Capable to modify their behaviour
- Proactive behaviour: actions depending on motivations generated from their internal state

same as in classic ABMs: decentralised: there is no place where global system behaviour (system dynamics) is defined. Instead individual agents interact with each other and their environment to produce complex collective behaviour patterns.

also central to ACE: emergent properties. They show up in the form of equilibria

spatial / geo-spatial aspects not as dominant as in other fields of ABMs. TODO: is this really true? more important: networks between agents

what are goals in ACE? what are behaviour in ACE?

behaviour and intelligence is not the main focus
they can be seen as a continuous transformation process

2.3 Alternative to object-orientation

TODO: this is already too specific, we need first to derive why haskell: 1st because its not been done so far and because it solves a lot of problems

The pure functional programming language Haskell offers all the required features. Together with the concept of functional reactive programming (FRP) we will show that ABS becomes very well possible to be implemented in a functional language. The declarative nature of Haskell allows us to implement an embedded domain-specific language (EDSL) for ABS based on the FRP paradigm.

2.3.1 The Actor Model

The Actor-Model, a model of concurrency, has been around since the paper [29] in 1973. It was a major influence in designing the concept of Agents and although there are important differences between Actors and Agents there are huge similarities thus the idea to use actors to build agent-based simulations comes quite natural. Although there are papers around using the actor model as basis for their ABMS unfortunately no proper theoretical treatment of using the actor-model in implementing agent-based simulations has been done so far. This paper looks into how the more theoretical foundations of the suitability of actor-model to ABMS and what the upsides and downsides of using it are.

`http://www.grids.ac.uk/Complex/ABMS/`

1. [?]
2. [?]
3. [?]
4. [?]
5. [?]
6. [?]
7. [?]
8. [?]

upside: extreme huge number of agnts possible due to distributed and parallel technology downside: depends on system & hardware: scheduler, system time, systime resolution (not very nice for scientific computation), much more complicated, debugging difficult due to concurrency, no global notion of time appart from systemtime, thus always runs in real-time, but there is no global

notion of time in the actor model anyway, no EDSL full of technical details, no determinism, no reasoning

Agents more a high-level concept, Actors low level, technical concurrency primitives

This makes simulations very difficult and also due to concurrency implementing a sync conversation among agents is very cumbersome. I have already experience with the Actor Model when implementing a small version of my Master-Thesis Simulation in Erlang which uses the Actor Model as well. For a continuous simulation it was actually not that bad but the problem there was that between a round-trip between 2 agents other messages could have already interfered - this was a problem when agents trade with each other, so one has to implement synchronized trading where only messages from the current agent one trades with are allowed otherwise budget constraints could be violated. Thus I think Erlang/Akka/Actor Model is better suited for distributed high-tolerance concurrent/parallel systems instead for simulations. Note: this is definitely a major point I have to argue in my thesis: why I am rejecting the actor model.

AKKA: thus my prediction is: akka/actor model is very well suited to simulations which 1. dont rely on global time 2. dont have multi-step conversations: interactions among agents which are only question-answer. TODO: find some classical simulation model which satisfies these criterias.

how can we simulate global time? how can we implement multistep conversations (by futures)?

The real problem seems to be concurrency but i feel we can simulate concurrency by synchronizing to continuous time. computations are carried out after another but because time is explicitly modelled they happen logically at the same time. these rules hold: an agent cannot be in two conversations at the same time, the agent can be in only one or none conversation at a given time t.

What if time is of no importance and only the continuous dynamics are of interest?

To put it another way: real concurrency (with threads) makes time implicit which is what one does NOT want in simulation. Maybe FRP is the way to go because it allows to explicitly model continuous and discrete time, but I have to get into FRP first to make a proper judgement about its suitability.

2.3.2 Functional Programming

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

the specification language should not be too technical, its focus should be on non-technical expressiveness. The question is: can we abstract away the technicalities and still translate it directly to haskell (more or less)? If not, can be adjust our Haskell implementation to come closer to our specification language? Thus it is a two-fold approach: both languages need to come closer to each other if we want to close the gap

TODO: [40], [30] it is still not exactly clear =¿ we present a formal specification using ABS

it is not trivial to reproduce the results as there is only very informal descriptions in [40]. [30] give a few more details but also stay quite informal. Thus here we represent a pure functional formulation of the original program which makes it formally exactly clear how the simulation should work. we then look how it can be translated to an ABM specification

Object-oriented (OO) programming is the current state-of-the-art method used in implementing ABM/S due to the natural way of mapping concepts and models of ABM/S to an OO-language. Although this dominance in the field we claim that OO has also its serious drawbacks:

- Mutable State is distributed over multiple objects which is often very difficult to understand, track and control.
- Inheritance is a dangerous thing if not used properly and with care because it introduces very strong dependencies which cannot be changed during runtime any-more.
- Objects don't compose very well due to their internal (mutable) state (note that we are aware that there is the concept of immutable objects which are becoming more and more popular but that does not solve the fundamental problem.
- It is (nearly) impossible to reason about programs.

We claim that these drawbacks are non-existent in pure functional programming like Haskell due to the nature of the functional approach. To give an introduction into functional programming is out of scope of this paper but we refer to the classical paper of [31] which is a great paper explaining to non-functional programmers what the significance of functional programming is and helping functional programmers putting functional languages to maximum use by showing the real power and advantages of functional languages. The main conclusion of this classical paper is that *modularity*, which is the key to successful programming, can be achieved best using higher-order functions and lazy evaluation provided in functional languages like Haskell. [31] argues that the ability to divide problems into sub-problems depends on the ability to glue the sub-problems together which depends strongly on the programming-language and [31] argues that in this ability functional languages are superior to structured programming.

[46] present an EDSL for Haskell allowing to specify Agents using the BDI model. We don't go there, thats not our intention.

[42] and [50] present a domain-specific language for developing functional reactive agent-based simulations. This language called FRABJOUS is very human readable and easily understandable by domain-experts. It is not directly implemented in FRP/Haskell/Yampa but is compiled to Haskell/Yampa code which they claim is also readable. This is the direction we want to head but

we don't want this intermediate step but look for how a most simple domain-specific language embedded in Haskell would look like. We also don't touch upon FRP and Yampa yet but leave this to further research for another paper of ours.

[37] TODO

TODO: cite julie greensmith paper on haskell

The amount of research on using the pure functional paradigm using Haskell in the field of ABS has been moderate so far. Most of the papers look into how agents can be specified using the belief-desire-intention paradigm [16], [46], [35]. A library for Discrete Event Simulation (DES) and System Dynamics (SD) in Haskell called *Aivika 3* is described in [44]. It comes with very basic features for ABS but only allows to specify simple state-based agents with timed transitions. [35] discuss using functional programming for discrete event simulation (DES) and mention the paradigm of FRP to be very suitable to DES.

[46] present an EDSL for Haskell allowing to specify Agents using the BDI model. TODO: We don't go there, thats not our intention.

[42] and [50] present a domain-specific language for developing functional reactive agent-based simulations. This language called FRABJOUS is very human readable and easily understandable by domain-experts. It is not directly implemented in FRP/Haskell/Yampa but is compiled to Haskell/Yampa code which they claim is also readable. This is the direction we want to head but we don't want this intermediate step but look for how a most simple domain-specific language embedded in Haskell would look like. We also don't touch upon FRP and Yampa yet but leave this to further research for another paper of ours.

[37] TODO

TODO: cite julie greensmith paper on haskell

We don't focus on BDI or similar but want to rely much more on low-level basic messaging. We can also draw strong relations to Hoare's Communicating Sequential Processes (CSP), Milner's Calculus of Communicating Systems (CCS) and Pi-Calculus. By mapping the EDSL to CSP/CCS/Pi-Calculus we achieve to be able to algebraic reasoning in our EDSL. TODO: hasn't Agha done something similar in connecting Actors to the Pi-Calculus?

[9] constructs two frameworks: an agent-modelling framework and a DES framework, both written in Haskell. They put special emphasis on parallel and concurrency in their work. The author develops two programs: HLogo which is a clone of the NetLogo agent-modelling framework and HDES, a framework for discrete event simulation - where in both implementations is the very strong emphasis on parallelism. Here only the HLogo implementation is of interest as it is directly related to agent-based simulation. In this implementation the author claims to have implemented an EDSL which tries to be close to the language used for modelling in NetLogo (Logo) "which lifts certain restrictions of the original NetLogo implementation". Also the aim was to be "faster in most circumstances than NetLogo" and "utilizes many processor cores to speedup the execution of Agent Based Models". The author implements a primitive model of concurrent agents which implements a non-blocking concurrent execution of

agents which report their results back to the calling agent in a non-blocking manner. The author mentions that a big issue of the implementation is that repeated runs with same inputs could lead to different results due to random event-orderings happening because of synchronization. The problem is that the author does not give a remedy for that and just accepts it as a fact. Of course it would be very difficult, if not impossible, to introduce determinism in an inherently concurrent execution model of agents which may be the reason the author does not even try. Unfortunately the example implementation the author uses for benchmarking is a very simplistic model: the basic pattern is that agent A sends to agent B and thats it - no complex interactions. Of course this lends itself very good to parallel/concurrent execution and does not need a sophisticated communication protocol. The work lacks a proper treatment of the agent-model presented with its advantages and disadvantages and is too sketchy although the author admits that is is just a proof of concept.

Tim Sweeney, CTO of Epic Games gave an invited talk about how "future programming languages could help us write better code" by "supplying stronger typing, reduce run-time failures; and the need for pervasive concurrency support, both implicit and explicit, to effectively exploit the several forms of parallelism present in games and graphics." [47]. Although the fields of games and agent-based simulations seem to be very different in the end, they have also very important similarities: both are simulations which perform numerical computations and update objects - in games they are called "game-objects" and in abm they are called agents but they are in fact the same thing - in a loop either concurrently or sequential. His key-points were:

- Dependent types as the remedy of most of the run-time failures.
- Parallelism for numerical computation: these are pure functional algorithms, operate locally on mutable state. Haskell ST, STRef solution enables encapsulating local heaps and mutability within referentially transparent code.
- Updating game-objects (agents) concurrently using STM: update all objects concurrently in arbitrary order, with each update wrapped in atomic block - depends on collisions if performance goes up.

TODO: discuss [42] TODO: discuss [50] TODO: discuss [46] TODO: discuss [35] TODO: discuss [16] TODO: discuss [44]

TODO: check out the internet for Actors/Agents i Haskell, but havn't found anything promising

`http://haskell-distributed.github.io/wiki.html` looks good but too big and not well suited for simulations `https://code.google.com/archive/p/haskellactor/` makes heavy use of IORef and running in IO-Monad, something we deliberately want to avoid to keep the ability to reason about the program. TODO: `https://github.com/fizruk/free-agent` look into

2.3.3 Haskell

NOTE: this chapter should be the very first implementation chapter as the approach of Haskell lays the very foundations for the functional approach by introducing the basic functional concepts

So far the literature on agent-based modelling & simulation (ABM/S) hasn't focused much on models for functional agents and is lacking a proper treatment of implementing agents in pure-functional languages like Haskell. This paper looks into how agents can be specified functionally and then be implemented properly in the pure functional language Haskell. The functional agent-model is inspired by wooldridge 2.6. The programming paradigm used to implement the agents in Haskell is functional reactive programming (FRP) where the Yampa framework will be used. The paper will show that specifying and implementing agents in a pure functional language like Haskell has many advantages over classical object-oriented, concurrent ones but needs also more careful considerations to work properly.

TODO: read [7]

The state-of-the-art approach to implementing Agents are object-oriented methods and programming as the metaphor of an Agent as presented above lends itself very naturally to object-orientation (OO). The author of this thesis claims that OO in the hands of inexperienced or ignorant programmers is dangerous, leading to bugs and hardly maintainable and extensible code. The reason for this is that OO provides very powerful techniques of organising and structuring programs through Classes, Type Hierarchies and Objects, which, when misused, lead to the above mentioned problems. Also major problems, which experts face as well as beginners are 1. state is highly scattered across the program which disguises the flow of data in complex simulations and 2. objects don't compose as well as functions. The reason for this is that objects always carry around some internal state which makes it obviously much more complicated as complex dependencies can be introduced according to the internal state. All this is tackled by (pure) functional programming which abandons the concept of global state, Objects and Classes and makes data-flow explicit. This then allows to reason about correctness, termination and other properties of the program e.g. if a given function exhibits side-effects or not. Other benefits are fewer lines of code, easier maintainability and ultimately fewer bugs thus making functional programming the ideal choice for scientific computing and simulation and thus also for ACE. A very powerful feature of functional programming is Lazy evaluation. It allows to describe infinite data-structures and functions producing an infinite stream of output but which are only computed as currently needed. Thus the decision of how many is decoupled from how to (Hughes, J. (1989). Why functional programming matters. Comput. J., 32(2):98–107.). The most powerful aspect using pure functional programming however is that it allows the design of embedded domain specific languages (EDSL). In this case one develops and programs primitives e.g. types and functions in a host language (embed) in a way that they can be combined. The combination of these primitives then looks like a language specific to a given domain, in the case

of this thesis ACE. The ease of development of EDSLs in pure functional programming is also a proof of the superior extensibility and composability of pure functional languages over OO (Henderson P. (1982). Functional Geometry. Proceedings of the 1982 ACM Symposium on LISP and Functional Programming.). One of the most compelling example to utilize pure functional programming is the reporting of Hudak (Hudak P., Jones M. (1994). Haskell vs. Ada vs. C++ vs. Awk vs. ... An Experiment in Software Prototyping Productivity. Department of Computer Science, Yale University.) where in a prototyping contest of DARPA the Haskell prototype was by far the shortest with 85 lines of code. Also the Jury mistook the code as specification because the prototype did actually implement a small EDSL which is a perfect proof how close EDSL can get to and look like a specification.

Functional languages can best be characterized by their way computation works: instead of *how* something is computed, *what* is computed is described. Thus functional programming follows a declarative instead of an imperative style of programming. The key points are:

- No assignment statements - variables values can never change once given a value.
- Function calls have no side-effect and will only compute the results - this makes order of execution irrelevant, as due to the lack of side-effects the logical point in *time* when the function is calculated within the program-execution does not matter.
- higher-order functions
- lazy evaluation
- Looping is achieved using recursion, mostly through the use of the general fold or the more specific map.
- Pattern-matching

General principles

idea: can we implement a message between two agents through events? thus two states: waiting for messages, processing messages. BUT: then sending a message *will take some time*

NOTE: it is important to make a difference about whether the simulation will dynamically *add* or *remove* agents during execution. If this is not the case, a simple par-switch is possible to run ALL agent SF in parallel. If dynamically changes to the agent-population should be part of the simulation, then the dpSwitch or dpSwitchB should be used. Also it should be possible to start/stop agents: if they are inactive then they should have no running SF because would use up resources. Inactive means: doing nothing, also not awaiting something/"doing nothing in the sense that DOING something which is nothing - the best criteria to decide if an agent can be set inactive is when the event

which decides if the agents SF should be started comes from outside e.g. if the agent is just statically "living" but not changing and then another agent will "ignite" the "living" agent then this is a clear criterion for being static without a running SF.

NOTE: the route-function will be used to distribute "messages" to the agents when they are communicating with each other

NOTE: [**?**] argues that in Game-Engines (it is paraphrased in english, as the thesis was written in german): "communication among Game-Objects is always computer-game specific and must be implemented always new but the functionality of Game-objects can be built by combining independent functions and signal-functions which are fully reuse-able". Game-Objects can be understood as agents thus maybe this also holds true for agent-based simulation. [**?**] thus distinguishes between normal functions e.g. mathematical functions, signal functions which depend on output since its creation in localtime and game-object functions which output depends on inputs AND time (which is but another input).

TODO: need a mechanism to address agents: if agent A wants to send a message to agent B and agent B wants to react by answering with a message to agent A then they must have a mechanism to address each other

TODO: design general input/output data-structures

TODO: design general agent SF

TODO: don't loose STM out of sight!
Wormholes in FRP?

2.3.4 Structuring

Of course the basic pure functional primitives alone do not make a well structured functional program by themselves as the usage of classes, interfaces, objects and inheritance alone does not make a well structured object-oriented program. What is needed are *patterns* how to use the primitives available in pure functional programs to arrive at well structure programs. In object-orientation much work has been done in the 90s by the highly influential book [25] whereas in functional programming the major inventions were also done in the 90s by the invention of Monads through [**?**], [**?**] and [**?**] and beginning of the 2000s by the invention of Arrows through [**?**].

map & fmap, foldl, applicatives [33] gives a great overview and motivation for using fmap, applicatives and Monads. TODO: explain Monads

[**?**] is a great tutorial about *Arrows* which are very well suited for structuring functional programs with effects.

> Just like monads, arrow types are useful for the additional operations they support, over and above those that every arrow provides.

The main difference between Monads and Arrows are that where monadic computations are parameterized only over their output-type, Arrows computations are parameterised both over their input- and output-type thus making Arrows more general.

> In real applications an arrow often represents some kind of a process, with an input channel of type a, and an output channel of type b.

In the work [**?**] an example for the usage for Arrows is given in the field of circuit simulation. They use previously introduced streams to advance the simulation in discrete steps to calculate values of circuits thus the implementation is a form of *discrete event simulation* - which is in the direction we are heading already with ABM/S. Also the paper mentions Yampa which is introduced in the section (TODO: reference) on functional reactive programming.

Monads Arrows Continuations

2.3.5 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

FRP is a paradigm for programming hybrid systems which combine continuous and discrete components. Time is explicitly modelled: there is a continuous

and synchronous time flow.

there have been many attempts to implement FRP in frameworks which each has its own pro and contra. all started with fran, a domain specific language for graphics and animation and at yale FAL, Frob, Fvision and Fruit were developed. The ideas of them all have then culminated in Yampa which is the reason why it was chosen as the FRP framework. Also, compared to other frameworks it does not distinguish between discrete and synchronous time but leaves that to the user of the framework how the time flow should be sampled (e.g. if the sampling is discrete or continuous - of course sampling always happens at discrete times but when we speak about discrete sampling we mean that time advances in natural numbers: 1,2,3,4,... and when speaking of continuous sampling then time advances in fractions of the natural numbers where the difference between each step is a real number in the range of [0..1])

time- and space-leak: when a time-dependent computation falls behind the current time. TODO: give reason why and how this is solved through Yampa. Yampa solves this by not allowing signals as first-class values but only allowing signal functions which are signal transformers which can be viewed as a function that maps signals to signals. A signal function is of type SF which is abstract, thus it is not possible to build arbitrary signal functions. Yampa provides primitive signal functions to define more complex ones and utilizes arrows [**?**] to structure them where Yampa itself is built upon the arrows: SF is an instance of the Arrow class.

Fran, Frob and FAL made a significant distinction between continuous values and discrete signals. Yampas distinction between them is not as great. Yampas signal-functions can return an Event which makes them then to a signal-stream - the event is then similar to the Maybe type of Haskell: if the event does not signal then it is NoEvent but if it Signals it is Event with the given data. Thus the signal function always outputs something and thus care must be taken that the frequency of events should not exceed the sampling rate of the system (sampling the continuous time-flow). TODO: why? what happens if events occur more often than the sampling interval? will they disappear or will the show up every time?

switches allow to change behaviour of signal functions when an event occurs. there are multiple types of switches: immediate or delayed, once-only and recurring - all of them can be combined thus making 4 types. It is important to note that time starts with 0 and does not continue the global time when a switch occurs. TODO: why was this decided?

[**?**] give a good overview of Yampa and FRP. Quote: "The essential abstraction that our system captures is time flow". Two *semantic* domains for progress of time: continuous and discrete.

The first implementations of FRP (Fran) implemented FRP with synchronized stream processors which was also followed by [**?**]. Yampa is but using continuations inspired by Fudgets. In the stream processors approach "signals are represented as time-stamped streams, and signal functions are just functions from streams to streams", where "the Stream type can be implemented directly as (lazy) list in Haskell...":

```
type Time = Double
type SP a b = Stream a -> Stream b
newtype SF a b = SF (SP (Time, a) b)
```

Continuations on the other hand allow to freeze program-state e.g. through closures and partial applications in functions which can be continued later. This requires an indirection in the Signal-Functions which is introduced in Yampa in the following manner.

```
type DTime = Double

data SF a b =
        SF { sfTF :: DTime -> a -> (SF a b, b)
```

The implementer of Yampa call a signal function in this implementation a *transition function*. It takes the amount of time which has passed since the previous time step and the durrent input signal (a). It returns a *continuation* of type SF a b determining the behaviour of the signal function on the next step (note that exactly this is the place where how one can introduce stateful functions like integral: one just returns a new function which encloses inputs from the previous time-step) and an *output sample* of the current time-step.

When visualizing a simulation one has in fact two flows of time: the one of the user-interface which always follows real-time flow, and the one of the simulation which could be sped up or slowed down. Thus it is important to note that if I/O of the user-interface (rendering, user-input) occurs within the simulations time-frame then the user-interfaces real-time flow becomes the limiting factor. Yampa provides the function embedSync which allows to embed a signal function within another one which is then run at a given ratio of the outer SF. This allows to give the simulation its own time-flow which is independent of the user-interface.

One may be initially want to reject Yampa as being suitable for ABM/S because one is tempted to believe that due to its focus on continuous, time-changing signals, Yampa is only suitable for physical simulations modelled explicitly using mathematical formulas (integrals, differential equations,...) but that is not the case. Yampa has been used in multiple agent-based applications: [**?**] uses Yampa for implementing a robot-simulation, [**?**] implement the classical Space Invaders game using Yampa, the thesis of [**?**] shows how Yampa can be used for implementing a Game-Engine, [**?**] implemented a 3D first-

person shooter game with the style of Quake 3 in Yampa. Note that although all these applications don't focus explicitly on agents and agent-based modelling / simulation all of them inherently deal with kinds of agents which share properties of classical agents: game-entities, robots,... Other fields in which Yampa was successfully used were programming of synthesizers (TODO: cite), Network Routers, Computer Music Development and various other computer-games. This leads to the conclusion that Yampa is mature, stable and suitable to be used in functional ABM/S.
Jason Gregory (Game Engine Architecture) defines Computer-Games as "soft real-time interactive agent-based computer simulations".

To conclude: when programming systems in Haskell and Yampa one describes the system in terms of signal functions in a declarative manner (functional programming) using the EDSL of Yampa. During execution the top level signal functions will then be evaluated and return new signal functions (transition functions) which act as continuations: "every signal function in the dataflow graph returns a new continuation at every time step".

"A major design goal for FRP is to free the programmer from 'presentation' details by providing the ability to think in terms of 'modeling'. It is common that an FRP program is concise enough to also serve as a specification for the problem it solves" [**?**]. This quotation describes exactly one of the strengths using FRP in ACE

TODO: [41]

[**?**] discuss the semantic framework of FRP. Very difficult to understand and full of corollaries and theorems and proofs, have to study in depth at another time.

2.3.6 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:

[**?**] gives a wonderful way of constructing an EDSL do denotationally construct an Escher-Picture.

2.3.7 Category- & Type-Theory

Category-Theory [**?**] [45]

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

2.4 Verification & Validation of ABS

To do verification we need a form of formal specification which can be translated easily to the code. Being inspired by the previously mentioned work on a functional framework for agent-based models of exchange in [10] we opt for a similar direction. Having Haskell as the implementation language instead of an object-oriented one like Java allows us to build on the above proposed EDSL for ABS. Because of the declarative nature of the hypothesized EDSL it can act both as specification- and implementation-language which closes the gap between specification and implementation. This would give us a way of formally specifying the model but still in a more readable way than pure mathematics. This form of formal specification can act easily as a medium for communication between team-memberes and to the scientific audience in papers. Most important the explicit step of verification becomes obsolete as there exists no more difference between specification and implementation. The last point seems to be quite ambitious but this is a hypothesis and we will see in the course of the thesis how far we can close the gap in the end with this approach.

TODO: need to do lots of literature research still Having the EDSL from the firt two hypotheses at hand it may be possible to extend it through additional primitves which then allow to formulate hypotheses in a formal way which then can be checked automatically.

TODO: `http://jasss.soc.surrey.ac.uk/12/1/1.html` TODO: `http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4419595` TODO: `http://dspace.stir.ac.uk/handle/1893/3365#.WNjO1DsrKM8` TODO: `http://www2.econ.iastate.edu/tesfatsi/DockingSimModels.pdf` TODO: `http://www2.econ.iastate.edu/tesfatsi/empvalid.htm` TODO: `http://jasss.soc.surrey.ac.uk/10/2/8.html` TODO: `https://www.openabm.org/faq/how-validate-and-calibrate-agent-based-` TODO: `https://link.springer.com/chapter/10.1007%2F978-3-642-01109-2_10` TODO: `http://www3.nd.edu/~nom/Papers/ADS019_Xiang.pdf`

The cooperative work of [5] gives insights into validation of computational models, in a process what they call "alignment". They try to determine if two models deal with the same phenomena. For this they tried to qualtiatively reproduce the same results of [2] in the Sugarscape model of [22]. Both models are of very different nature but try to investigate the qualitatively same phenomoenon: that of cultural processes. TODO: read

model checking and reasoning in [32]

In [13] introduce *QuickCheck*, a testing-framework which allows to specifify properties and invariants of ones functions and then test them using randomly generated test-data. This is an additional tool of model-specification and in-

creases the power and strength of the verification process and more properties of a model can be expressed which are directly formulated in code through the EDSL of QuickCheck AND the EDSL of FrABS. Of course it also serves for testing (e.g. regression) and points out errors in the implementation e.g. wrong assumptions about input-data. The authors claim that the major advante of QuickCheckj is to formulate formal specifications which help in understanding a program. TODO: the question is whether it can be used for Validation as well.

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)

Chapter 3

Aims and Objectives

Deriving from these hypotheses we propose a new tool for implementing, testing, verifying and validating ABS: the one of pure functional programming and its underlying theory which we together will term *pure functional methods*. We claim that these methods are the answer to the questions posed above and will reduce or even elimiate the danger of failure considerably as outlined in the hypotheses. Of course our motivation is not only to improve verification, validation and reproduceability in ABS but also the one of discovery. So far pure functional programming was not really investigated in the context of ABS as will be shown in the literature-review so this thesis is also a proof-of-concept, an investigation how ABS can be done in a pure functional language.

The central aspect of this thesis is centred around the main question of *How Agent-Based Simulation can be done using pure functional programming and what the benefits and disadvantages are.*. 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

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

We show that the implicit assumption that an Agent is *about equal to* an object is not correct and leads to many implicit assumptions in OO implementations of an 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.

TODO: is it really valid to bind the sending of messages to the advancing of time? Downside: we cannot have method-calls as in OO, which allows agents to communicate with each other without time advancing.

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.

I noticed that it is pretty hard to convince an agent-based economics specialist who is not a computer scientist about a pure functional approach. My conjecture is that the implementation technique and method does not matter much to them because they have very little knowledge about programming and are almost always self-taught - they don't know about software-engineering, nothing about proper software-design and architecture, nothing about software-maintenance, nothing about unit-testing,... In the end they just "hack" the simulation in whatever language they are able to: C++, Visual Basic, Java or toolboxes like Netlogo. For them it is all about to *get things done somehow* and not to get things done the right way or in a beautiful way - the way and the method doesn't matter, its just a necessary evil which needs to be done. Thus if functional programming could make their lives easier, then they will definitely welcome it. But functional programming is, i think, harder to learn and harder to understand - so one needs to provide an abstraction through EDSL. So I REALLY need to come up with convincing arguments why to use pure functional approaches in ACE THEY can understand, otherwise I will be lost and not heard (not published,...).

What ACE economists care for:

- Very: Qualitative modelling with quantitative results
- Yes: Easy reproducibility
- Likely: Reasoning about convergence?
- Likely: EDSL

My contributions are: pure functional framework, functional agent-model for market-simulations, EDSL for market-simulations, qualitative / implicit modelling with quanitative results, reasoning in my framework about convergence

IDEA: could I develop non-causal modelling (models are expressed in terms of non-directed equations, modelled in signal-relations) to allow for qualitative modelling for the agent-based economists? See hybrid modelling paper of Yampa. **THIS WOULD BE A HUGE NOVEL CONTRIBUTION TO ACE ESPECIALLY WHEN COMBINED WITH AN EDSL AND PROVIDING FULL REFERENTIAL TRANSPARENCY TO KEEP THE ABILITY TO REASON ABOUT CONVERGENCE**. This should be covered in the "EDSL"-paper.

TODO: maybe i should really focus only on market models? otherwise too much?

central novelty of my PhD: model specification = runnable code. possible through EDSL. but only in specific subfield of ACE: market-models. need a functional description of the model, then translate it to model specification in

EDSL and then run it to see dynamics. But: model specification moves closer to functional programming languages.

another novelty approach: model specification through qualitative instead of quantiative approaches. is this possible?

pure functional agent-model & theory, EDSL framework in Haskell for ACE

1. Which kind of problem do we have?
2. What aim is there? Solving the problem?
3. How the aim is achieved by enumerating VERY CLEAR objectives.
4. What the impact one expects (hypothesis) and what it is (after results).

Note: It is not in the interest of the researcher to develop new economic theories but to research the use of functional methods (programming and specification) in agent-based computational economics (ACE).

NOTE: Get the reader's attention early in the introduction: motivation, significance, originality and novelty.

Methods need to be selected to implement the simulations. Special emphasis will be put on functional ones which will then be compared to established methods in the field of ABM/S and ACE.

Claim: non-programming environments are considered to be not powerful enough to capture the complexity of ACE implementations thus a programming approach to ACE will be always required.

To apply and test functional methods in ACE, four scenarios of ACE are selected and then the methods applied and compared with each other to see how each of them perform in comparison. The 4 selected scenarios represent a selection of the challenges posed in ACE: from very abstract ones to very operational ones.

Each of the selected scenarios is then implemented using the selected methods where each solution is then compared against the following criteria:

1. suitability for scientific computation
2. robustness
3. error-sources
4. testability
5. stability
6. extendability
7. size of code

8. maintainability
9. time taken for development
10. verification & correctness
11. replications & parallelism
12. EDSL

This will then allow to compare the different methods against each other and to show under which circumstances functional methods shine and when they should not be used.

3.1 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)

- 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

3.1.1 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?

Chapter 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).

4.0.1 Papers Submitted

Update-Strategies in ABS TODO: attach as appendix
A foundational paper

4.0.2 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

Papers in Progress

Recursive ABS basic work and prototype within Schelling Segregation is running. The main finding for now is that it does not increase the convergence

speed to equilibrium but can lead to extreme volatility of dynamics although the system seems to be near to complete equilibrium. In the case of a 10x10 field it was observed that although the system was nearly in its steady state - all but one agents were satisfied - the move of a single agent caused the system to become completele unstable and depart from its near-equilibrium state to a highly desequilibrium state.

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: [26] 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

Software

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

Lots of learning & prototyping for Haskell: IO, STM, Pure Functional, FRP

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

4.0.3 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

4.0.4 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)

Chapter 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.

It is of most importance to stress that we don't condemn the current state-of-the-art approach of object-oriented specification and implementation to ABS. We are realists and know that there are more points to consider when selecting a set of methods for developing software for an ABS than robustness, verification and validation. Almost always the popularity of an existing language and which languages the implementer knows is the driving force behind which methods and languages to choose. This means that ABS will continue to be implemented in object-oriented programming languages and many perfectly well functioning models will be created by it in the future. Although they all suffer from the same issues mentioned in the introduction this doesn't matter as they are not of central importance to most of them. Nonetheless we think our work is still essential and necessary as it may start a slow paradigm-shift and opens up the minds of the ABS community to a more functional and formal way of approaching and implementing agent-based models and simulations and recognizing the benefits one gets automatically from it by doing so.

Chapter 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!

6.0.1 TODOs

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

Category Theory

develop category theory behind FrABS: look into monads, arrows

category theory foundations (monads, arrows)

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

Verification and Validation

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

Papers

paper 2: recursive ABS: dont pursue recursive ABS further as separate paper: it could take a whole PhD by itself. proof-of-concept is ok but not groundbreaking.

maybe some really groundbreaking idea will come up in a few months, then can work on it again. incorporate it maybe in FrABS paper. definitely incorporate it in final thesis 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

Reading

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

6.0.2 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.

- what do i want to have achieved in the first year? -¿ FrABS -¿ 1st conference paper published -¿ research questions -¿ milestones -¿ clear idea what i want to do and where to go

- Paradigm shift in ABS!

- Focus on Papers + Milestones -¿ find out which conferences I want to go and when the deadlines are

- organize as agile sprints

- Conferences -¿ Multi-Agent Systems, good for verification and validation: AAMS, Deadline in November -¿ Social Simulation, good for new methods in simulation e.g. MetaBS: SSC, Deadline in March -¿ functional, for presenting functional stuff but not too advanced: TFP -¿ TODO some functional, good for FrABS & EDSL: ?

6.0.3 Milestones

TODO: add subpoints

Submitted Paper on Iteration

SSC 2017, Deadline in 31st March, 25-29th September 2017, `http://www.sim2017.com/`

Submitted Paper on Towards FrABS

TFP 2017, Deadline 5th May 2017, 19-21th June 2017, `https://www.cs.kent.ac.uk/events/tfp17/index.html`

1st Year Viva

Deadline: end of July

Submitted Paper on MetaABS

SSC 2018, Deadline in March 2018?,

Submitted Paper on EDSL

TODO Some functional conference?

Submitted Paper on Validation & Verification

AAMS 2018, Deadline in November 2018, `http://www.aamas2017.org/`

Completed PhD

Deadline September 2019

6.0.4 Time-Line

TODO: add Gantt-Chart from june 2017 on detailed overview of activity also project gantt-chart back to what i have done so far

The whole PhD lasts for 3 years, 36 Months, from October 2016 to September 2019 and thus I will structure it according to 3 years where each year will be a major milestone - which is also intended by the Computer School.

6.0.5 ToDo

1. Paper: agents' vs. agent's
2. FRP & ABS = FrABS: meeting with Henrik
3. FrABS: work on it
4. Agent_Zero genauer anschauen: winter simulation papers, unterschied zu sugarscape?
5. experiment with MetaABS in SchellingSegregation
6. Refine project-plan

6.0.6 1st Year: Groundwork

In this year I will learn basics and develop and research the methodology I will use for the main work in the 2nd year. The year will be guided by the principal question of "How can Agent-Based Simulation be done using pure functional programming?".

Important mile-stones:

31st March 2017	Finished and submit Paper
June (Mid) 2017	Finished writing 1st year report
End of June / Begin of July 2017	Oral annual report

Major Activities

- Develop FrABS library: March 2017 - May 2017
- Refine FrABS and publish as library on Hackage: July 2017 - September 2017
- Write FrABS paper: July 2017 - September 2017
- Study decentralized bartering: July 2017 - September 2017

March 2017

- experiment with MetaABS (also implement it in the EDSL) to see if it is making any sense to follow this road
- Programming: bring ABS to FRP in Yampa: FrABS, implement SugarScape
- 31st March: Finalize and submit 'Art of Iterating'-Paper to SSC 2017
- Formulate Research Questions
- Draft Project Plan

April 2017

- 9th - 13th: Midland Graduate School in Leicester
- Programming: bring ABS to FRP in Yampa: FrABS, implement SugarScape
- Refine Research Questions
- Refine Project Plan

May 2017

- Start writing 1st year report
- Programming: bring ABS to FRP in Yampa: FrABS, implement SugarScape
- Finalize Research Questions
- Finalize Project Plan

June 2017

- 4th - 18th: 2 weeks holiday on Amrum (planned last year already)
- Finalize 1st year report
- Prepare for 1st Year oral exam
- End (or beginning of July) 1st Year oral exam

July 2017

- Start writing on FrABS Paper: show how the rules of SugarScape can be formalized in FrABS =¿ specification equals code
- Study decentralized bilateral trading/bartering
- Refine FrABS library

August 2017

- Work on FrABS Paper: Formalize the Rules in the EDSL of FrABS
- Study decentralized bilateral trading/bartering
- Refine FrABS library

September 2017

- Finalize FrABS Paper
- Study decentralized bilateral trading/bartering
- Refine FrABS library: put on hackage

October 2017

2nd year starts

6.0.7 2nd Year: Main Work

Applying 1st year results, methods and experiences to work in the Research-Questions. The second year is investigating the question of "What are the benefits of doing ABS in pure functional programming?".

Important mile-stones:

October	Finished FrABS Paper
?	Submit FrABS Paper
?	Finished and submit MetaABS Paper

Major Points for research

- EDSL which is both specification- and implementation- language
- MetaABS which allows to do recursive simulation
- Reasoning about dynamics and emergent properties of ABS

October 2017

Start of 2nd year

December 2017

- 22nd December 2017 - 7th January 2018: 2 weeks xmas holidays

January 2018

- 22nd December 2017 - 7th January 2018: 2 weeks xmas holidays

April 2018

1 week holiday

August 2018

2 weeks holidays

October 2018

3rd year starts

6.0.8 3rd Year: Finalizing, Publishing & Writing

The third year serves to refine, finish and publish the research of the 2nd year (ideally a journal paper) and then to write the final thesis. To have a bit of distraction and to prevent myself to become too locked in in writing on the thesis I will also work on my optional philosophical paper and hope to at least finish them and maybe publish them - at least I want to present them to 2-3 audiences (e.g. FP Lunch) to test the reaction.

Important mile-stones:

30th September 2019	End of official PhD

October 2018

3rd year starts

October 2018 - December 2018

Finalize research of 2nd year

22nd December 2018 - 6th January 2019

2 weeks xmas holidays

January 2019 - March 2019

Publish journal paper of 2nd year, define structure of PhD Thesis

April 2019

1 week holiday

April 2019 - August 2019

Writing thesis

September 2019

Submitting final thesis, 2 weeks of holidays

October 2019

End of official PhD

6.0.9 4th Year: Pending Period

It is very hard to finish ALL within the 3rd year and it is very likely that I will enter the 4th year - pending period. I plan on spending in this period not more than till Christmas as I have no funding in this time and I want to finish within time.

Important mile-stones:

1st October 2019	Start of pending period
30th September 2020	End of pending period

October

- Prepare for viva

November

- Viva

December

- Refine Thesis (incorporate minor changes)

Bibliography

[1] Agha, G. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, MA, USA, 1986.

[2] Axelrod, R. The Convergence and Stability of Cultures: Local Convergence and Global Polarization. Working Paper, Santa Fe Institute, Mar. 1995.

[3] Axelrod, R. Advancing the Art of Simulation in the Social Sciences. In *Simulating Social Phenomena*. Springer, Berlin, Heidelberg, 1997, pp. 21–40. DOI: 10.1007/978-3-662-03366-1_2.

[4] Axelrod, R. Chapter 33 Agent-based Modeling as a Bridge Between Disciplines. In *Handbook of Computational Economics*, L. T. a. K. L. Judd, Ed., vol. 2. Elsevier, 2006, pp. 1565–1584. DOI: 10.1016/S1574-0021(05)02033-2.

[5] Axtell, R., Axelrod, R., Epstein, J. M., and Cohen, M. D. Aligning simulation models: A case study and results. *Computational & Mathematical Organization Theory 1*, 2 (Feb. 1996), 123–141.

[6] Baas, N. A., and Emmeche, C. On Emergence and Explanation. Working Paper, Santa Fe Institute, Feb. 1997.

[7] Backus, J. Can Programming Be Liberated from the Von Neumann Style?: A Functional Style and Its Algebra of Programs. *Commun. ACM 21*, 8 (Aug. 1978), 613–641.

[8] Baker, H. K., Kiymaz, H., Alan, N. S., Bildik, R., and Schwartz, R. Market Microstructure in Emerging and Developed Markets. *Business Faculty Book Gallery* (Jan. 2013).

[9] Bezirgiannis, N. *Improving Performance of Simulation Software Using Haskell's Concurrency & Parallelism*. PhD thesis, Utrecht University - Dept. of Information and Computing Sciences, 2013.

[10] 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.

[11] Bowles, S., Edwards, R., and Roosevelt, F. *Understanding Capitalism: Competition, Command, and Change*, 3 edition ed. Oxford University Press, New York, Mar. 2005.

[12] Budish, E., Cramton, P., and Shim, J. Editor's Choice The High-Frequency Trading Arms Race: Frequent Batch Auctions as a Market Design Response. *The Quarterly Journal of Economics 130*, 4 (2015), 1547–1621.

[13] Claessen, K., and Hughes, J. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming* (New York, NY, USA, 2000), ICFP '00, ACM, pp. 268–279.

[14] Colell, A. M. *Microeconomic Theory*. Oxford University Press, 1995. Google-Books-ID: dFS2AQAACAAJ.

[15] Darley, V., and Outkin, A. V. *Nasdaq Market Simulation: Insights on a Major Market from the Science of Complex Adaptive Systems*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2007.

[16] De Jong, T. Suitability of Haskell for Multi-Agent Systems. Tech. rep., University of Twente, 2014.

[17] Di Stefano, A., and Santoro, C. Using the Erlang Language for Multi-Agent Systems Implementation. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology* (Washington, DC, USA, 2005), IAT '05, IEEE Computer Society, pp. 679–685.

[18] Di Stefano, A., and Santoro, C. eXAT: an Experimental Tool for Programming Multi-Agent Systems in Erlang. Tech. rep., 2007.

[19] Epstein, J. M. Chapter 34 Remarks on the Foundations of Agent-Based Generative Social Science. In *Handbook of Computational Economics*, L. T. a. K. L. Judd, Ed., vol. 2. Elsevier, 2006, pp. 1585–1604. DOI: 10.1016/S1574-0021(05)02034-4.

[20] Epstein, J. M. *Generative Social Science: Studies in Agent-Based Computational Modeling*. Princeton University Press, Jan. 2012. Google-Books-ID: 6jPiuMbKKJ4C.

[21] Epstein, J. M. *Agent_Zero: Toward Neurocognitive Foundations for Generative Social Science*. Princeton University Press, Feb. 2014. Google-Books-ID: VJEpAgAAQBAJ.

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

[23] Evensen, P., and Märdin, M. *An Extensible and Scalable Agent-Based Simulation of Barter Economics*. Master, Chalmers University of Technology, Göteborg, 2010.

[24] Gaffeo, E., Gatti, D. D., Desiderio, S., and Gallegati, M. Adaptive microfoundations for emergent macroeconomics. Department of Economics Working Paper 0802, Department of Economics, University of Trento, Italia, 2008.

[25] Gamma, E., Helm, R., Johnson, R., Vlissides, J., and Booch, G. *Design Patterns: Elements of Reusable Object-Oriented Software*, 1 edition ed. Addison-Wesley Professional, Oct. 1994.

[26] 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.

[27] Gintis, H. The Emergence of a Price System from Decentralized Bilateral Exchange. *Contributions in Theoretical Economics 6*, 1 (2006), 1–15.

[28] Gintis, H. The Dynamics of General Equilibrium. SSRN Scholarly Paper ID 1017117, Social Science Research Network, Rochester, NY, Sept. 2007.

[29] Hewitt, C., Bishop, P., and Steiger, R. A Universal Modular ACTOR Formalism for Artificial Intelligence. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence* (San Francisco, CA, USA, 1973), IJCAI'73, Morgan Kaufmann Publishers Inc., pp. 235–245.

[30] Huberman, B. A., and Glance, N. S. Evolutionary games and computer simulations. *Proceedings of the National Academy of Sciences 90*, 16 (Aug. 1993), 7716–7718.

[31] Hughes, J. Why Functional Programming Matters. *Comput. J. 32*, 2 (Apr. 1989), 98–107.

[32] Hutton, G. A Tutorial on the Universality and Expressiveness of Fold. *J. Funct. Program. 9*, 4 (July 1999), 355–372.

[33] Hutton, G. *Programming in Haskell*. Cambridge University Press, Cambridge, UK ; New York, Jan. 2007.

[34] Ionescu, C., and Jansson, P. Dependently-Typed Programming in Scientific Computing. In *Implementation and Application of Functional Languages* (Aug. 2012), R. Hinze, Ed., Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 140–156.

[35] Jankovic, P., and Such, O. Functional Programming and Discrete Simulation. Tech. rep., 2007.

[36] Kirman, A. *Complex Economics: Individual and Collective Rationality*. Routledge, London ; New York, NY, July 2010.

[37] Klügl, F., and Davidsson, P. AMASON: Abstract Meta-model for Agent-Based SimulatiON. In *Multiagent System Technologies: 11th German Conference, MATES 2013, Koblenz, Germany, September 16-20, 2013. Proceedings*, M. Klusch, M. Thimm, and M. Paprzycki, Eds. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 101–114. DOI: 10.1007/978-3-642-40776-5_11.

[38] MacKie-Mason, J. K., and Wellman, M. P. Chapter 28 Automated Markets and Trading Agents. In *Handbook of Computational Economics*, L. T. a. K. L. Judd, Ed., vol. 2. Elsevier, 2006, pp. 1381–1431. DOI: 10.1016/S1574-0021(05)02028-9.

[39] Marks, R. Chapter 27 Market Design Using Agent-Based Models. In *Handbook of Computational Economics*, L. T. a. K. L. Judd, Ed., vol. 2. Elsevier, 2006, pp. 1339–1380. DOI: 10.1016/S1574-0021(05)02027-7.

[40] Nowak, M. A., and May, R. M. Evolutionary games and spatial chaos. *Nature 359*, 6398 (Oct. 1992), 826–829.

[41] Perez, I., Bärenz, M., and Nilsson, H. Functional Reactive Programming, Refactored. In *Proceedings of the 9th International Symposium on Haskell* (New York, NY, USA, 2016), Haskell 2016, ACM, pp. 33–44.

[42] Schneider, O., Dutchyn, C., and Osgood, N. Towards Frabjous: A Two-level System for Functional Reactive Agent-based Epidemic Simulation. In *Proceedings of the 2Nd ACM SIGHIT International Health Informatics Symposium* (New York, NY, USA, 2012), IHI '12, ACM, pp. 785–790.

[43] Sher, G. I. *Agent-Based Modeling Using Erlang Eliminating The Conceptual Gap Between The Programming Language & ABM*. 2013.

[44] Sorokin, D. *Aivika 3: Creating a Simulation Library based on Functional Programming*. 2015.

[45] Spivak, D. I. *Category Theory for the Sciences*, 1 ed. Mit Press Ltd, Cambridge, Massachusetts, Nov. 2014.

[46] Sulzmann, M., and Lam, E. Specifying and Controlling Agents in Haskell. Tech. rep., 2007.

[47] Sweeney, T. The Next Mainstream Programming Language: A Game Developer's Perspective. In *Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (New York, NY, USA, 2006), POPL '06, ACM, pp. 269–269.

[48] Tesfatsion, L. Agent-Based Computational Economics: A Constructive Approach to Economic Theory. Handbook of Computational Economics, Elsevier, 2006.

[49] Varela, C., Abalde, C., Castro, L., and Gulías, J. On Modelling Agent Systems with Erlang. In *Proceedings of the 2004 ACM SIGPLAN Workshop on Erlang* (New York, NY, USA, 2004), ERLANG '04, ACM, pp. 65–70.

[50] Vendrov, I., Dutchyn, C., and Osgood, N. D. Frabjous: A Declarative Domain-Specific Language for Agent-Based Modeling. In *Social Computing, Behavioral-Cultural Modeling and Prediction*, W. G. Kennedy, N. Agarwal, and S. J. Yang, Eds., no. 8393 in Lecture Notes in Computer Science. Springer International Publishing, Apr. 2014, pp. 385–392.

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

6.1 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