Towards pure functional agent-based simulation

Jonathan Thaler
School of Computer Science
University of Nottingham
jonathan.thaler@nottingham.ac.uk

Peer-Olaf Siebers
School of Computer Science
University of Nottingham
peer-olaf.siebers@nottingham.ac.uk

Abstract

TODO: the main punch is that our approach combines the best of the three simulation methodologies: - SD part: it can represent continuous time (as well as discrete) with continuous data-flows from agents which act at the same time (parallel update), can express the formulas directly in code, there exists also a small EDSL for expressing SD in our approach, can guarantee reproducibility and no drawing of random-numbers in our approach -¿ drawback over real SD: none known so far - DES part: it can represent discrete time with events occurring at discrete points in time which cause an instant change in the system -¿ drawback over real DES: time does not advance discretely to the next event which results of course not in the performance of a real DES system - ABS part: the entities of the system (=agents) can be heterogenous and pro-active in time and can have arbitrary neighbourhood (2d/3d discrete/continuous, network,...) -¿ drawback over classic ABS: none known so far

TODO: give examples of all 3 approaches: SD & ABS: SIR model, DES: simulation of a queuing system

TODO: describe the different approach which is necessary because of being functional - data-flows & update-strategies: sequential, parallel, concurrent, actor - how state is handled

TODO: main benefits - being explicit and polymorph about side-effects: can have 'pure' (no side-effects except state), 'random' (can draw random-numbers), 'IO' (all bets are off), STM (concurrency) agents - hybrid between SD and ABS due to continuous time AND parallel dataFlow (parallel update-strategy) - being update-strategy polymorph (TODO: this is just an asumption atm, need to prove this): 4 different update-strategies, one agent implementation - parallel update-strategy: lack of implicit side-effects makes it work without any danger of data-interference - recursive simulation - reasoning about correctness - reasoning about dynamics - testing with quickcheck much more convenient - expressivity: -¿ 1:1 mapping of SD to code: can express the SD formulas directly in code -¿ directly expressing state-charts in code

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

TODO: describing how things are treated different - time is represented using the FRP concept: Signal-Functions which are sampled at (fixed) time-deltas, the dt is never visible directly but only reflected in the code and read-only. - no method calls =¿ continuous data-flow instead - no global shared mutable environment, having different options: -¿ non-active read-only (SIR): no agent, as additional argument to each agent -¿ pro-active read-only (?): environment as agent, broadcast environment updates as data-flow -¿ non-active read/write (?): no agent, shared data as STM as additional argument to each agent -¿ pro-active read/write (Sugarscape): environment as, shared data as STM as additional argument to each agent - parallel update only, sequential is deliberately abandoned due to: -¿ reality does not behave this way -¿ if we need transactional behaviour, can use STM which is more explicit -¿ it is translates directly to a map which is very easy to reason about (sequential is basically a fold which is much more difficult to reason about) -¿ is more natural in functional programming -¿ it exists for 'transactional' reasons where we need mutual exclusive access to environment / other agents -¿ we provide a more explicit mechanism for this: Agent Transactions - state is handled using FRP: recursive arrows and continuations - still need transactions between two agents e.g. trading occurs over multiple steps (makeoffer, accept/refuse, finalize/abort) -¿ exactly define what TX means in ABS -¿ exclusive between 2 agents -¿ state-changes which occur over multiple steps and are only visible to the other agents after the TX has commited -¿ no read/write access to this state is allowed to other agents while the TX is active -¿ a TX executes in a single time-step and can have an arbitrary number of tx-steps -¿ it is easily possible using method-calls in OOP but in our pure functional approach it is not possible -¿ parallel execution is being a problem here as TX between agents are very easy with sequential -¿ an agent must be able to transact with as many other agents as it wants to in the same time-step -¿ no time passes between transactions =¿ what we need is a 'all agents transact at the same time' -¿ basically we can implement it by running the SFs of the agents involved in the TX repeatedly with dt=0 until there are no more active TXs -¿ continuations (SFs) are perfectly suited for this as we can 'rollback' easily by using the SF before the TX has started

TODO: defining agent-based simulation - look into existing literature (e.g. peers paper, any logic book,...)

TODO: need to find a formal definition on agent-based simulation - start with wooldridge book - derive a more functional approach then in my paper

TODO: it doesn't make sense for an agent to act 'always', an agents behaviour needs to have some time-dependent parameter e.g. doEvery 1.0. If this is omitted then one makes one dependent directly on the Time-Delta.

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

Index Terms

Haskell, Functional Programming, Verification

I. Introduction

In this paper we investigate how agent-based simulation can be approach from a pure functional direction using the programming language Haskell. So far no in-depth research has been conducted on this subject because so far agent-based simulation was always thought of as being best implemented in object-oriented languages like C++, Java and Python. As will become apparent throughout this paper, a pure functional approach needs to approach various concepts of ABS very different which leads to a very different approach to ABS but makes concepts which where blurred and implicit very explicit and clear. This papers major contribution is that it reveals these implicit concept and provides a formal view on ABS and a pure functional implementation of these concepts.

II. Agent-Based Simulation Defined

Agent-Based Simulation (ABS) is a methodology to model and simulate a system where the global behaviour may be unknown but the behaviour and interactions of the parts making up the system is of knowledge. Those parts, called agents, are modelled and simulated out of which then the aggregate global behaviour of the whole system emerges. Epstein [1] identifies ABS to be especially applicable for analysing *"spatially distributed systems of heterogeneous autonomous actors with bounded information and computing capacity"*. Thus in the line of the simulation methods *Statistic* †, *Markov* ‡, *System Dynamics* §, *Discrete Event* $^{\mp}$, ABS is the most recent development and the most powerful one as it subsumes it's predecessors features and goes beyond:

- Linearity & Non-Linearity $^{\dagger\ddagger\S\mp}$ - the dynamics of the simulation can exhibit both linear and non-linear behaviour.
- Time $^{\dagger\ddagger\S\mp}$ - agents act over time, time is also the source of pro-activity.
- States $^{\ddagger\S\mp}$ - agents encapsulate some state which can be accessed and changed during the simulation.
- Feedback-Loops $^{\S\mp}$ - because agents act continuously and their actions influence each other and themselves, feedback-loops are the norm in ABS.
- Heterogeneity $^{\mp}$ - although agents can have same properties like height, sex,... the actual values can vary arbitrarily between agents.
- Interactions - agents can be modelled after interactions with an environment or other agents, making this a unique feature of ABS, not possible in the other simulation models.
- Spatiality & Networks - agents can be situated within e.g. a spatial (discrete 2d, continuous 3d,...) or network environment, making this also a unique feature of ABS, not possible in the other simulation models.

A. Deriving central concepts

Before we can approach a functional view on ABS, we need to identify the central concepts of ABS on a more technical level. Unfortunately there does not exist a commonly agreed technical definition of ABS but we can draw inspiration from the closely related field of Multi-Agent Systems (MAS). It is important to understand that MAS and ABS are two different fields where in MAS the focus is much more on technical details implementing a system of interacting intelligent agents within a highly complex environment with the focus primarily on solving AI problems.

1) Agents: The central aspect of ABS is the concept of an agent. In MAS [2], [3] agents can be informally defined as:

- They are uniquely addressable entities with some internal state over which they have full, exclusive control.
- They are situated in an environment which they can observe and act upon.
- They can interact with other agents which are situated in the same environment.
- They are pro-active which means they can initiate actions on their own e.g. change their internal state, interact with other agents, create new agents, terminate themselves, interact with the environment,...

2) Environment: The other important concept is the one of an environment. In MAS [2], [3] one distinguishes between different types of environments (based on [4]):

- Accessible vs. inaccessible - in an accessible environment an agent can obtain complete and accurate information from the environment. In ABS environments are generally implemented as being accessible.
- Deterministic vs. non-deterministic - in a deterministic environment the actions of an agent have no uncertainty and are guaranteed to have a single effect. In ABS environments are generally implemented as being deterministic.
- Static vs. dynamic - a static environment only changes due to the agents actions whereas a dynamic one has other processes which operate on it. In ABS both static and dynamic environments are common.
- Discrete vs. continuous - a discrete environment has only a fixed, finite number of states and actions whereas a continuous is potentially unlimited. In ABS both discrete and continuous environments are common.

Note that in MAS the focus is much more on the environment rather than on the agents where the environment is almost always a highly complex one and the agents may intelligently act on it. In ABS the focus is rather on the agents and their interactions where the environment plays a role but is not of central interest as it is almost always deterministic.

B. Deriving a formal view

In order to explore how we can implement an ABS in a pure functional way we need a sufficiently formal view on it. This will help us expressing the concepts in Haskell as formal, mathematical specifications translate easily into functional programming. There exists formalisations of MAS [2] but unfortunately they are not very helpful in our context as its formalization is tailored much more towards optimizing, intelligent and reasoning behaviour of agents within a highly complex and uncertain environment. What we need for ABS is a more agent-oriented approach:

1) An ABS is a simulation over time in which time is advanced either in discrete or continuous time-steps where discrete means advancing by a natural number time-delta and continuous by a real-valued time-delta. So we have a potentially infinite stream of time-steps starting at t=0 advancing by some fixed time-delta.
2) At each time-step all agents are allowed to act which is the source of their proactivity because it allows them to initiate actions on their own. Of course such actions are always time-dependent - be it explicitly like executing actions *after* a specific time, or be it implicit like executing actions every time-delta - but this is the only way of implementing proactivity in a computer system.
3) In each step an agent should be able to read/write the environment. TODO: orderings? when are changes visible?
4) In each step an agent should be able to interact with other agents through communication.
5) In each step an agent should be able to update its internal state.
6) Depending on its type, the environment must also be allowed to act in each time-step.
7) In general we can thus see an agent to exhibit both time-dependent and reactive behaviour: it can act continuously or discretely, depending on how the time is advanced and exhibit reactive behaviour which means it can react to changing environment or agents.
8) The interactions between agents their update-state and environment forms a feedback as the state of time ti forms the input state on which to act at time-step ti+1

III. A FUNCTIONAL VIEW

Due to the fundamentally different approaches of pure Functional Programming (pure FP) an ABS needs to be implemented fundamentally different as well compared to traditional object-oriented approaches (OO). We face the following challenges:

1) How can we represent an Agent? In OO the obvious approach is to map an agent directly onto an object which encapsulates data and provides methods which implement the agents actions. Obviously we don't have objects in pure FP thus we need to find a different approach to represent the agents actions and to encapsulate its state.
2) How can we represent state in an Agent? In the classic OO approach one represents the state of an Agent explicitly in mutable member variables of the object which implements the Agent. As already mentioned we don't have objects in pure FP and state is immutable which leaves us with the very tricky question how to represent state of an Agent which can be actually updated.
3) How can we implement proactivity of an Agent? In the classic OO approach one would either expose the current time-delta in a mutable variable and implement time-dependent functions or ignore it at all and assume agents act on every step. At first this seems to be not a big deal in pure FP but when considering that it is yet unclear how to represent Agents and their state, which is directly related to time-dependent and reactive behaviour it raises the question how we can implement time-varying and reactive behaviour in a purely functional way.
4) How can we implement the agent-agent interaction? In the classic OO approach Agents can directly invoke other Agents methods which makes direct Agent interaction *very* easy. Again this is obviously not possible in pure FP as we don't have objects with methods and mutable state inside.
5) How can we represent an environment and its various types? In the classic OO approach an environment is almost always a mutable object which can be easily made dynamic by implementing a method which changes its state and then calling it every step as well. In pure FP we struggle with this for the same reasons we face when deciding how to represent an Agent, its state and proactivity.
6) How can we implement the agent-environment interaction? In the classic OO approach agents simply have access to the environment either through global mechanisms (e.g. Singleton or simply global variable) or passed as parameter to a method and call methods which change the environment. Again we don't have this in pure FP as we don't have objects and globally mutable state.
7) How can we step the simulation? In the classic OO approach agents are run one after another (with being optionally shuffled before to uniformly distribute the ordering) which ensures mutual exclusive access in the agent-agent and agent-environment interactions. Obviously in pure FP we cannot iteratively mutate a global state.

A. Agent representation, state and proactivity

Viewing agent-agent interaction as simple method calls leads to a few dangerous assumptions. - it takes no time - it has a synchronous and transactional character - an agent gives up control over its data / actions or at least there is always the danger that it exposes too much of its interface and implementation details. - agents equals objects, which is definitely NOT true. Agents

signal functions in pure FP everything is a function.

data-flow synchronous agent transactions

B. Environment representation and interaction

pro-active vs. inactive read-only vs. read/write

- non-active read-only (SIR): no agent, as additional argument to each agent - pro-active read-only (?): environment as agent, broadcast environment updates as data-flow - non-active read/write (?): no agent, shared data as STM as additional argument to each agent - pro-active read/write (Sugarscape): environment as, shared data as STM as additional argument to each agent

C. Stepping the simulation

ACKNOWLEDGMENTS

The authors would like to thank I. Perez, H. Nilsson, J. Greensmith for constructive comments and valuable discussions.

REFERENCES

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

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

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

[4] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010.