First contact: Agents meet Haskell
Simulating epidemics using Functional Reactive Programming

An Agent-Based Approach

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

Thorsten Altenkirch
School of Computer Science
University of Nottingham
thorsten.altenkirch@nottingham.ac.uk

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

Abstract

TODO: cite my own 1st paper from SSC2017: add it to citations

TODO: refine it: start with simulating epidemics and then go into ABS

Agent-Based Simulation (ABS) is a methodology in which a system is simulated in a bottom-up approach by modelling the micro interactions of its constituting parts, called agents, out of which the global macro system behaviour emerges. So far, the Haskell community hasn't been much in contact with the community of ABS due to the latter's primary focus on the object-oriented programming paradigm. This paper tries to bridge the gap between those two communities by introducing the Haskell community to the concepts of ABS. We do this by deriving an agent-based implementation for the simple SIR model from epidemiology. In our approach we leverage the basic concepts of ABS with functional reactive programming from Yampa which results in a surprisingly fresh, powerful and convenient EDSL for formulating ABS in Haskell.

Index Terms

Functional Reactive Programming, Agent-Based Simulation

I. Introduction

TODO: start with simulating epidemics and then go into ABS

Background: Follow the old 1st draft paper: 1. introduce simulating epidemics with the SIR model 2. explain the SIR model 3. explain agent-based approach Implementation: deriving implementations step-by-step 1. naive implementation without Yampa using the rand-monad advantage: first approach, it works, having random-monad is VERY convenient drawback: time is explicitly available and need to feedback all agent-states into every agent 2. naive yampa implementation advantage: time is implicit drawback: still all agent-states are visible to every agent 3. extended yampa implementation with environment and AgentIn/Out advantage: more general drawback: AgentOut is cumbersome to handle iteratively 4. dunai / bearriver advantage: can have monads AND functional reactive drawback: ?

with occasionally we can achieve extremely fine grained stochastics as opposed to draw random number of events we create only a single event or not, this allows for a much smoother curve and is a real advantage: we are treating it as a continuous system

we need deterministic behaviour under all curcumstances, thus we cannot use IO or STM. for globally mutable state we use StateT

emphasise of dataflow which is necessary because connections between agents are not fixed at compile time

2 main points from Henrik: 1.: try to stick to synchronous updates because this is how the real world works. 2.: get rid of globally shared mutable state as it complicates things extremely with reasoning

clear conceptual formal model of what agents are and then structure my implementation around it

implementing dynamic environment and transactional behaviour can be done most elegantly through STM but need monads, this could provide a perfect motivation using bearriver / dunai

idea: get rid of env in agent SF alltogether (and in agentdef,/in,/out) and use STM for read/write environment. still need some way for a proactive environment: make environment an agent itself. run all agents parallel and remove sequential updates.

Acknowledgments

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