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

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

II. Agent-Based Simulation Defined

A formal view

Acknowledgments

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

References