Towards pure functional ABS Part I

The suitability of Haskell for implementing ABS

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*—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 being its most prominent representative. In this paper we examine the suitability of the pure functional programming language Haskell for implementing ABS. We approach the problem from a general direction and look into the features and power of Haskell, compare it with existing technologies in ABS and see what potential benefits and drawbacks using Haskell in ABS are. The findings are directly applied in the second part, published as a separate paper, where we introduce a general-purpose library implemented in Haskell for implementing all kinds of ABS.**

***Index Terms*—Agent-Based Simulation, Haskell, Functional Programming**

I. Introduction

- The 3 major benefits of the approach I claim 1. code == spec 2. can rule out serious class of bugs 3. we can perform reasoning about the simulation in code need to be metricated: e.g. this is really only possible in Haskell and not in Java. This needs thorough thinking about which metrics are used, how they can be aquired, how they can be compared,...

- I NEED TO SHOW HOW I CAN MAKE HASKELL RELEVANT IN THE FIELD OF ABS -¿ as far as I know so far no reasoning has been done in the way I intend to do it in the field of ABS. My hypothesis is that it is really only possible in Haskell due to its explicit side-effects, type-system, declarative style,... -¿ TODO: need to check if this is really unique to haskell -¿ the functional-reactive approach seems to bring a new view to ABS with an embedded language for explicit time-semantics. Together with parallel/sequential updating this allows implementing System-Dynamics and agents which rely on continuous time-semantics e.g. SIR-Agents. Maybe i invented a hybrid between SD and ABS? Also what about time-traveling? The problem is that this is not really clear as i hypothesize that is completely novel approach to ABS - again I need to check this! -¿ TODO: is this really unique to functional reactive? E.g. what about Repast, NetLogo, AnyLogic, other Java-Frameworks? -¿ maybe i have to admit that its not as unique as thought

In General i need to show that - Haskells general benefits & drawbacks over other Languages in the Field of ABS (e.g. Java, NetLogo, Repast) e.g. declarative style, reasoning, explicit about side-effects, performance, difficult to reason about performance, space-leaks difficult. So this focuses on the general comparison between the established technologies of ABS and Haskell but not yet on Haskells suitability in comparison to these other technologies. Here we talk about reasoning, side-effects, performance IN GENERAL TERMS, NOT SPECIFIC TO ABS. We need to distinguish between -¿ general technicalities e.g. lambda-calculus (denotational formalism) or turing-machine (operational formalism) foundations, declarative style, lazy-evaluation allows to split the producer from the consumer, explicit about side-effects, not possible for in-order updates,... -¿ and in what they result e.g. fewer lines of code, ruling out of bugs, reasoning, lower performance, difficult to reason about space-time

- Haskells suitability to implement ABS in comparison to other languages and technologies in the Field. Here the focus is on general problems in ABS and how they can and are solved using Haskell e.g. send message, changing environment, handling of time, replications, parallelism/concurrency,...

- Why using Haskell in ABS - do the general benefits / drawbacks apply equally well? Are there unique advantages? Can we do things in Haskell which are not possible in other technologies or just very hard? E.g. the hybrid-approach I created with FRP: how unique is it e.g. can other technologies easily implement it as well? Other potential advantages: recursive simulation. Here we DO NOT concentrate on general technicalities but see how they apply when using it for ABS and if they create a unique benefit for Haskell in ABS.

i need to show that different programming languages and paradigms have different power and are differently well suited to specific problems: the ultimate claim i need to show is that haskell is more powerful than java or C++ - the question is if this also makes it superior in applying it to problems: being more powerful, can all problems of java be solved better in haskell as well? this is i believe not the case e.g. gui- or game-programming. the question then is: what is the power of a programming language? can we measure it?

so what i need to show is how well haskell and its power are suited for implementing ABS. does the fact that haskell is much more powerful than existing technologies in ABS lead to the point that it is better suited for ABS? in fact it is power vs. better suited

A. The power of a language

[ ] more expressive: we can express complex problems more directly and with less overhead. note that this is domain-specifix: the mechanisms of a language allow to create abstractions which solve the domain-specific problem. the better these mechanisms support one in this task, the more powerful the language is in the given domain. now we end up by defining what ”better” support means [ ] one could in pronciple do system programming in haskell by provoding bindings to code written in c and / or assembly but when the program is dominated by calls to these bindings then one could as well work directly in these lower languages and saves one from the overhead of the bindings [ ] but very often a domain consists of multiple subdomains. [ ] my hypothesis is that haskell is not well suited for domains which are dominated by managing and manipulating a global mutable state through side-effects / effectful computations. examples are gui-programming and computer games (state spread accross GPU and cpu, user input,...). this does not mean that it is not possible to implememt these things in haskell (it has been done with some sucess) but that the solution becomes too complex at some point. [ ] conciesness [ ] low ceremony [ ] susceptibility to bugs [ ] verbosity [ ] reasoning about performance [ ] reasoning about space requirements

B. Measuring a language

Define scientific measures: e.g. Lines Of Code (show relation to Bugs & Defects, which is an objective measure: http://www.stevemcconnell.com/est.htm, https://softwareengineering.stackexchange.com/questions/185660/is-the-average-number-of-bugs-per-loc-the-same-for-different-programming-languag, Book: Code Complete, https://www.mayerdan.com/ruby/2012/11/11/bugs-per-line-of-code-ratio), also experience reports by companies which show that Haskell has huge benefits when applied to the same domain of a previous implementation of a different language, post on stack overflow / research gate / reddit, read experience reports from http://cufp.org/2015/ Also need to show the problem of operational reasoning as opposed to denotational reasoning

References