The Agent's new Cloths
Towards functional Agent-Based Simulation

[Anonymised review copy]

Abstract: TODO: implement MSF SugarScape TODO: implement STM Sugarscape

TODO: parallelism for free because all isolated e.g. running multiple replications or parameter-variations

TODO: it is paramount not to write against the established approach but for the functional approach. not to try to come up with arguments AGAINST the object-oriented approach but IN FAVOUR for the functional approach. In the end: dont tell the people that what they do sucks and that i am the saviour with my new method but: that i have a new method which might be of interest as it has a few nice advantages.

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.

Keywords: Agent-Based Simulation, Functional Programming, Haskell

Introduction

1.1 The traditional approach to Agent-Based Simulation (ABS) has so far always been object-oriented techniques, due to the influence of the seminal work of Epstein et al Epstein & Axtell (1996) in which the authors claim "[..] object-oriented programming to be a particularly natural development environment for Sugarscape specifically and artificial societies generally [..]" (p. 179). This work established the metaphor in the ABS community, that *agents map naturally to objects* (North & Macal 2007) which still holds up today.

1.2 In this paper we challenge this metaphor and explore ways of approaching ABS using the functional programming paradigm as in the language Haskell. By doing this we expect to leverage the benefits of it (Hudak et al. 2007) to become available when implementing ABS functionally: expressing *what* a system is instead of *how* it works through declarative code, being explicit about the interactions of the program with the real world, explicit data-centric programming resulting in less sources of bugs and a strong static type system making type-errors at run-time obsolete.

1.3 We show that these functional concepts result in an approach to implementing ABS where it is harder to make mistakes and allow to implement simulations which are guaranteed to be reproducible, have less sources of bugs, are easier to verify and thus more likely to be correct which is of paramount importance in high-impact scientific computing.

1.4 As a use-case throughout the paper we employ the well known SugarScape model (Epstein & Axtell 1996) to demonstrate our case-studies, because it can be seen as one of the most influential models in ABS and it laid the foundations of object-oriented implementation of agent-based models.

1.5 The aim of this paper is show *how* to implement ABS in functional programming as in Haskell and *why* it is of benefit of doing so. Further, we give the reader a good understanding of what functional programming is, what the challenges are in applying it to ABS and how we solve these in our approach.

1.6 The paper makes the following contributions:

- It is the first to *systematically* introduce the functional programming paradigm, as in Haskell, to ABS, identifying its benefits, difficulties and drawbacks.
- We show how functional ABS can be scaled up to massively large-scale without the problems of low level concurrent programming using Software Transactional Memory (STM). Although there exist STM implementations in non-functional languages like Java and Python, due to the nature of Haskells type-system, the use of STM has unique benefits in this setting.
- Further we introduce a powerful and very expressive approach to testing ABS implementations using property-based testing. This allows a much more powerful way of expressing tests, shifting from unit-testing towards specification-based testing. Although property-based testing has been brought to non-functional languages like Java and Python as well, it has its origins in Haskell and it is here where it truly shines.

References

Epstein, J. M. & Axtell, R. (1996). *Growing Artificial Societies: Social Science from the Bottom Up*. Washington, DC, USA: The Brookings Institution

Hudak, P., Hughes, J., Peyton Jones, S. & Wadler, P. (2007). A History of Haskell: Being Lazy with Class. In *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*, HOPL III, (pp. 12–1–12–55). New York, NY, USA: ACM. doi:10.1145/1238844.1238856
URL `http://dx.doi.org/10.1145/1238844.1238856`

North, M. J. & Macal, C. M. (2007). *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation*. Oxford University Press, USA. Google-Books-ID: gRATDAAAQBAJ