Specification equals Code: An EDSL for pure functional Agent-Based Modelling & Simulation

Jonathan Thaler

February 3, 2017

Abstract

Building upon our previous work on update-strategies in Agent-Based Modelling & Simulation (ABM/S) where we showed that Haskell is a very attractive alternative to existing object-oriented approaches we ask in this paper if the declarative power of the pure functional language can be utilized to write specifications for simple ABM/S models which can be directly translated to our Haskell implementation.

1 Introduction

the specification language should not be too technical, its focus should be on non-technical expressiveness. The question is: can we abstract away the technicalities and still translate it directly to haskell (more or less)? If not, can be adjust our Haskell implementation to come closer to our specification language? Thus it is a two-fold approach: both languages need to come closer to each other if we want to close the gap
TODO: [2], [1] it is still not exactly clear =¿ we present a formal specification using ABS
note that the difference between SEQ and PAR in Haskell is in the end a 'fold' over the agents in the case of SEQ and a 'map' in the case of PAR the conc-version and the act-version of the agent-implementations look EXACTLY the same BUT we lost the ability to step the simulation!!!
[4] present an EDSL for Haskell allowing to specify Agents using the BDI model.
TODO: [3]
TODO: [5]
We don't focus on BDI or similar but want to rely much more on low-level basic messaging. We can also draw strong relations to Hoare's Communicating Sequential Processes (CSP), Milner's Calculus of Communicating Systems (CCS) and Pi-Calculus. By mapping the EDSL to CSP/CCS/Pi-Calculus we achieve to be able to algebraic reasoning in our EDSL. TODO: hasn't Agha done something similar in connecting Actors to the Pi-Calculus?

2 Related Research

3 Problem

4 The specification Language

An Agent A is a 5-tuple ¡Aid, s, m, e, [tf]¿ Aid is the id of the agent s is the generic state of the agent m is the message-protocoll the agent understands including the messages (Dt d) and (Terminate) e is the generic environment the agent acts upon tf is a transformer-function for every message in the protocol (if an agent wants to ignore the message, it is just the identify function id)
the type of a tf depends on the semantics of the model and there are 4 of them SEQ: tf :: (A, e) -¿ (Aid, m) -¿ (A, e) PAR: tf :: (A, Global e) -¿ (Aid, m) -¿ (A, Local e) CON: tf :: (A, e) -¿ (Aid, m) -¿ STM A ACT: tf :: (A, e) -¿ (Aid, m) -¿ STM A
Local e = e Global e = Map Aid (Local e)
further the agent has the following functions at its hands sendMessage :: A -¿ Aid -¿ m -¿ A

5 Conclusion

6 Further Research

References

[1] Huberman, B. A., and Glance, N. S. Evolutionary games and computer simulations. *Proceedings of the National Academy of Sciences 90*, 16 (Aug. 1993), 7716–7718.

[2] Nowak, M. A., and May, R. M. Evolutionary games and spatial chaos. *Nature 359*, 6398 (Oct. 1992), 826–829.

[3] Schneider, O., Dutchyn, C., and Osgood, N. Towards Frabjous: A Two-level System for Functional Reactive Agent-based Epidemic Simulation. In *Proceedings of the 2Nd ACM SIGHIT International Health Informatics Symposium* (New York, NY, USA, 2012), IHI '12, ACM, pp. 785–790.

[4] Sulzmann, M., and Lam, E. Specifying and Controlling Agents in Haskell. Tech. rep., 2007.

[5] Vendrov, I., Dutchyn, C., and Osgood, N. D. Frabjous: A Declarative Domain-Specific Language for Agent-Based Modeling. In *Social Computing, Behavioral-Cultural Modeling and Prediction*, W. G. Kennedy, N. Agarwal, and S. J. Yang, Eds., no. 8393 in Lecture Notes in Computer Science. Springer International Publishing, Apr. 2014, pp. 385–392. DOI: 10.1007/978-3-319-05579-4_47.