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

Jonathan THALER

January 22, 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

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