Functional programming in Agent-Based Simulation and Modelling

Jonathan Thaler

December 1, 2016

Abstract

Agent-Based Modelling and Simulation (ABM/S) is still a young discipline and the dominant approach to it is object-oriented computation. This thesis goes into the opposite direction and asks how ABM/S can be mapped to and implemented using pure functional computation and what one gains from doing so. To the best knowledge of the author, so far no proper treatment of ABM/S in this field exists but a few papers which only scratch the surface. The author argues that approaching ABM/S from a pure functional direction offers a wealth of new powerful tools and methods. The most obvious one is that when using pure functional computation reasoning about the correctness and about total and partial correctness of the simulation becomes possible. Also pure functional approaches allow the design of an embedded domain specific language (EDSL) in which then the models can be formulated by domain-experts. The strongest point in using EDSL is that ideally the distinction between specification and implementation disappears: the model specification is then already the code of the simulation-program. This allows to rule out a serious class of errors where specification and implementation does not match, which is especially a big problem in scientific computing.
In this paper we look at the very simple social-simulation of *Heroes & Cowards* invented by TODO: cite to study the new methods mentioned above and to highlight its potential use and its limitations as opposed to object-oriented methods.

Introduction

1 Reasoning

Allowing to reason about a program is one of the most interesting and powerful features of a Haskell-program. Just by looking at the types one can show that there is no randomness in the simulation *after* the random initialization, which is not slightest possible in the case of a Java, Scala, ReLogo or NetLogo solution. Things we can reason about just by looking at types:

- Concurrency involved?

- Randomness involved?
- IO with the system (e.g. user-input, read/write to file(s),...) involved?

This all boils down to the question of whether there are *side-effects* included in the simulation or not.

1.1 Reasoning about termination

What about reasoning about the termination? Is this possible in Haskell? Is it possible by types alone? My hypothesis is that the types are an important hint but are not able to give a clear hint about termination and thus we we need a closer look at the implementation. In dependently-typed programming languages like Agda this should be then possible and the program is then also a proof that the program itself terminates.

1.2 Debugging

Because functions compose easier than classes & objects (TODO: we need hard claims here, look for literature supporting this thesis or proof it by myself) it is also much easier to debug *parts* of the implementation e.g. the rendering of the agents without any changes to the system as a whole - just the main-loop has do be adopted. Then it is very easy to calculate e.g. only one iteration and to freeze the result or to manually create agents instead of randomly create initial ones.

2 World

The coordinates calculated by the agents are *virtual* ones ranging between 0.0 and 1.0. This prevents us from knowing the rendering-resolution and polluting code which has nothing to do with rendering with these implementation-details. Also this simulation could run without rendering-output or any rendering-frontend thus sticking to virtual coordinates is also very useful regarding this (but then again: what is the use of this simulation without any visual output=

- Clipping: *calculated* coordinates are clipped at 0.0 and 1.0
- Wraparound: *calculated* coordinates are wrapped around to 0.0 when reaching 1.0. Will lead pursuing friends to change direction apruptly when wrapping around.
- Remainder: *calculated* coordinates are never clipped but in rendering only the remainder after the comma is taken. Of course this then always requires a visual output to observe the effects. This will result in an effect like the classic asteroid game where agents their friends but wont change direction when they seem to clamp around like in the wraparound version. Problem: we need to adjust the orientation as well otherwise it would look strange!

References