Dependent types in Agent-Based Simulation

JONATHAN THALER, THORSTEN ALTENKIRCH, and PEER-OLAF SIEBERS, University of Nottingham, United Kingdom

TODO

TODO: future research: can we combine all three into one: ABS which can do SD and DES?

Additional Key Words and Phrases: Idris, Functional Programming, Dependent Types, Agent-Based Simulation, Discrete Event Simulation

ACM Reference Format:
Jonathan Thaler, Thorsten Altenkirch, and Peer-Olaf Siebers. 2018. Dependent types in Agent-Based Simulation. *Proc. ACM Program. Lang.* 9, 4, Article 39 (September 2018), 3 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

Independent of the programming paradigm, there exist fundamentally two approaches implementing agent-based simulation: time- and event-driven. In the time-driven approach, the simulation is stepped in fixed Δt and all agents are executed at each time-step - they act virtually in lock-step at the same time. The approach is inspired by the theory of continuous system dynamics (TODO: cite). In the event-driven approach, the system is advanced through events, generated by the agents, and the global system state changes by jumping from event to event, where the state is held constant in between. The approach is inspired by discrete event simulation (DES) (TODO: citation) which is formalized in the DEVS formalism [10].

In a preceding paper we investigated how to derive a time-driven pure functional ABS approach in Haskell (TODO: cite my paper). We came to quite satisfactory results and implemented also a number of agent-based models of various complexity (TODO: cite schelling, sugarscape, agent zero). Still we identified weaknesses due to the underlying functional reactive programming (FRP) approach. It is possible to define partial implementations which diverge during runtime, which may be difficult to determine for complex models for a programmer at compile time. Also sampling the system with fixed Δt can lead to severe performance problems when small Δt are required, as was shown in our paper. The later problem is well known in the simulation community and thus as a remedy an event-driven approach was suggested [5]. In this paper for the first time, we derive a pure functional event-driven agent-based simulation. Instead of using Haskell, which provides already libraries for DES [7], we focus on the dependently typed pure functional programming language Idris. In our previous paper we hypothesised that dependent types may offer interesting new insights and approaches to ABS but it was unclear how exactly we can make use of them, which was left for further research. In this paper we hypothesise that, as opposed to a time-driven approach, the even-driven approach is especially suited to make proper use of dependent types due

Authors' address: Jonathan Thaler, jonathan.thaler@nottingham.ac.uk; Thorsten Altenkirch, thorsten.altenkirch@nottingham.ac.uk; Peer-Olaf Siebers, University of Nottingham, 7301 Wollaton Rd, Nottingham, NG8 1BB, United Kingdom, peer-olaf.siebers@nottingham.ac.uk.

2475-1421/2018/9-ART39 $15.00
https://doi.org/0000001.0000001

to its different nature. Note that both a pure functional event-driven approach to ABS *and* the use of dependent types in ABS has so far never been investigated, which is the unique contribution of this paper.

Dependent Types are the holy grail in functional programming as they allow to express even stronger guarantees about the correctness of programs and go as far where programs and types become constructive proofs [9] which must be total by definition [8], [2], [1], [6]. Thus the next obvious step is to apply them to our pure functional approach of agent-based simulation. So far no research in applying dependent types to agent-based simulation exists at all and it is not clear whether dependent types do make sense in this setting. We explore this for the first time and ask more specifically how we can add dependent types to our pure functional approach, which conceptual implications this has for ABS and what we gain from doing so. Note that we can only scratch the surface and lay down basic ideas and leave a proper in-depth treatment of this topic for further research. We use Idris [3], [4] as language of choice as it is very close to Haskell with focus on real-world application and running programs as opposed to other languages with dependent types e.g. Agda and Coq which serve primarily as proof assistants.

Dependent Types promise the following:

(1) Types as proofs - In dependently types languages, types can depend on any values and are first-class objects themselves. TODO: make more clear
(2) Totality and termination - Constructive proofs must terminate, this means a well-typed program (which is itself a proof) is always terminating which in turn means that it must consist out of total functions. A total function is defined by [4] as: it terminates with a well-typed result or produces a non-empty finite prefix of a well-typed infinite result in finite time. Idris is turing complete but is able to check the totality of a function under some circumstances but not in general as it would imply that it can solve the halting problem. Other dependently typed languages like Agda or Coq restrict recursion to ensure totality of all their functions - this makes them non turing complete.

Ionesus talk on dependently typed programming in scientific computing https://www.pik-potsdam.de/members/ionescu/cezar-ifl2012-slides.pdf Ionescus talk on Increasingly Correct Scientific Computing https://www.cicm-conference.org/2012/slides/CezarIonescu.pdf Ionescus talk on Economic Equilibria in Type Theory https://www.pik-potsdam.de/members/ionescu/cezar-types11-slides.pdf Ionescus talk on Dependently-Typed Programming in Economic Modelling https://www.pik-potsdam.de/members/ionescu/ee-tt.pdf

2 DEPENDENTLY TYPED SIR

A SIR model enters a steady state as soon as there are no more infected agents. Thus we can informally argue that a SIR model must always terminate as:

(1) Only infected agents can infect susceptible agents.
(2) Eventually after a finite time every infected agent will recover.
(3) There is no way to move from the consuming *recovered* state back into the *infected* state [1].

Thus a SIR model must enter a steady state after a finite amount of time. Using the DE presented in the introduction on the SD model we can show that the system enters a stable state in finite time (TODO: are there any references?):

TODO: show that it must terminate through the SD formulas. also calculate the t for when I(t)= 0

This result gives us the confidence, that the agent-based approach will terminate, given it is really a correct implementation of the SD model. Still this does not proof that the agent-based approach

[1]There exists an extended SIR model, called SIRS which adds a cycle to the state-machine by introducing a transition from recovered to susceptible but we don't consider that here.

itself will terminate and so far no proof of the totality of it was given. Dependent Types and Idris ability for totality and termination checking should theoretically allow us to proof that an agent-based SIR implementation terminates after finite time: if an implementation of the agent-based SIR model in Idris is total it is a proof by construction. Note that such an implementation should not run for a limited virtual time but run unrestricted of the time and the simulation should terminate as soon as there are no more infected agents. We hypothesize that it should be possible due to the nature of the state transitions where there are no cycles and that all infected agents will eventually reach the recovered state. Abandoning the FRP approach and starting fresh, the question is how we implement a *total* agent-based SIR model in Idris. Note that in the SIR model an agent is in the end just a state-machine thus the model consists of communicating / interacting state-machines. In the book [4] the author discusses using dependent types for implementing type-safe state-machines, so we investigate if and how we can apply this to our model. We face the following questions: how can we be total? can we even be total when drawing random-numbers? Also a fundamental question we need to solve then is how we represent time: can we get both the time-semantics of the FRP approach of Haskell AND the type-dependent expressivity or will there be a trade-off between the two?

TODO: implement sir with state-machine approach from Idris. an idea would be to let infected agents generate infection- actions: the more infected agents the more infection-actions => zero infected agents mean zero infection actions. this list can then be reduced?

can we also emulate SD in Idris and formulate positive/negative feedback loops in types?

REFERENCES

[1] Thorsten Altenkirch, Nils Anders Danielsson, Andres LÃűh, and Nicolas Oury. 2010. Pi_Sigma: Dependent Types Without the Sugar. In *Proceedings of the 10th International Conference on Functional and Logic Programming (FLOPS'10).* Springer-Verlag, Berlin, Heidelberg, 40–55. https://doi.org/10.1007/978-3-642-12251-4_5

[2] Thorsten Altenkirch, Conor Mcbride, and James Mckinna. 2005. Why dependent types matter. In *In preparation, http://www.e-pig.org/downloads/ydtm.pdf.*

[3] Edwin Brady. 2013. Idris, a general-purpose dependently typed programming language: Design and implementation. *Journal of Functional Programming* 23, 05 (2013), 552–593. https://doi.org/10.1017/S095679681300018X

[4] Edwin Brady. 2017. *Type-Driven Development with Idris.* Manning Publications Company. Google-Books-ID: eWzEjwEACAAJ.

[5] Ruth Meyer. 2014. Event-Driven Multi-agent Simulation. In *Multi-Agent-Based Simulation XV (Lecture Notes in Computer Science).* Springer, Cham, 3–16. https://doi.org/10.1007/978-3-319-14627-0_1

[6] The Univalent Foundations Program. 2013. *Homotopy Type Theory: Univalent Foundations of Mathematics.* https://homotopytypetheory.org/book, Institute for Advanced Study.

[7] David Sorokin. 2015. *Aivika 3: Creating a Simulation Library based on Functional Programming.*

[8] Simon Thompson. 1991. *Type Theory and Functional Programming.* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

[9] Philip Wadler. 2015. Propositions As Types. *Commun. ACM* 58, 12 (Nov. 2015), 75–84. https://doi.org/10.1145/2699407

[10] Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim. 2000. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems.* Academic Press. Google-Books-ID: REzmYOQmHuQC.

Received March 2018; revised March 2018; accepted March 2018