Towards pure functional Agent-Based Simulation in Haskell

Jonathan Thaler
School of Computer Science
University of Nottingham
jonathan.thaler@nottingham.ac.uk

Peer-Olaf Siebers
School of Computer Science
University of Nottingham
peer-olaf.siebers@nottingham.ac.uk

Abstract

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 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 FrABS through the example of the SIR model, discuss implications, benefits and best practices.

Index Terms

Haskell, Functional Programming, Verification

I. Introduction

A. Costy bugs due to language features

[ ] knight capital glitch [ ] mars lander [ ] moon landing [ ] ? [ ] ethereum & blockchain technology

- The 3 major benefits of the approach I claim 1. code == spec 2. can rule out serious class of bugs 3. we can perform reasoning about the simulation in code need to be metricated: e.g. this is really only possible in Haskell and not in Java. This needs thorough thinking about which metrics are used, how they can be aquired, how they can be compared,...

- I NEED TO SHOW HOW I CAN MAKE HASKELL RELEVANT IN THE FIELD OF ABS -¿ as far as I know so far no reasoning has been done in the way I intend to do it in the field of ABS. My hypothesis is that it is really only possible in Haskell due to its explicit side-effects, type-system, declarative style,... -¿ TODO: need to check if this is really unique to haskell -¿ the functional-reactive approach seems to bring a new view to ABS with an embedded language for explicit time-semantics. Together with parallel/sequential updating this allows implementing System-Dynamics and agents which rely on continuous time-semantics e.g. SIR-Agents. Maybe i invented a hybrid between SD and ABS? Also what about time-traveling? The problem is that this is not really clear as i hypothesize that is completely novel approach to ABS - again I need to check this! -¿ TODO: is this really unique to functional reactive? E.g. what about Repast, NetLogo, AnyLogic, other Java-Frameworks? -¿ maybe i have to admit that its not as unique as thought

In General i need to show that - Haskells general benefits & drawbacks over other Languages in the Field of ABS (e.g. Java, NetLogo, Repast) e.g. declarative style, reasoning, explicit about side-effects, performance, difficult to reason about performance, space-leaks difficult. So this focuses on the general comparison between the established technologies of ABS and Haskell but not yet on Haskells suitability in comparison to these other technologies. Here we talk about reasoning, side-effects, performance IN GENERAL TERMS, NOT SPECIFIC TO ABS. We need to distinguish between -¿ general technicalities e.g. lambda-calculus (denotational formalism) or turing-machine (operational formalism) foundations, declarative style, lazy-evaluation allows to split the producer from the consumer, explicit about side-effects, not possible for in-order updates,... -¿ and in what they result e.g. fewer lines of code, ruling out of bugs, reasoning, lower performance, difficult to reason about space-time

- Haskells suitability to implement ABS in comparison to other languages and technologies in the Field. Here the focus is on general problems in ABS and how they can and are solved using Haskell e.g. send message, changing environment, handling of time, replications, parallelism/concurrency,...

- Why using Haskell in ABS - do the general benefits / drawbacks apply equally well? Are there unique advantages? Can we do things in Haskell which are not possible in other technologies or just very hard? E.g. the hybrid-approach I created with FRP: how unique is it e.g. can other technologies easily implement it as well? Other potential advantages: recursive simulation. Here we DO NOT concentrate on general technicalities but see how they apply when using it for ABS and if they create a unique benefit for Haskell in ABS.

i need to show that different programming languages and paradigms have different power and are differently well suited to specific problems: the ultimate claim i need to show is that haskell is more powerful than java or C++ - the question is if this also makes it superior in applying it to problems: being more powerful, can all problems of java be solved better in haskell as well? this is i believe not the case e.g. gui- or game- programming. the question then is: what is the power of a programming language? can we measure it?

so what i need to show is how well haskell and its power are suited for implementing ABS. does the fact that haskell is much more powerful than existing technologies in ABS lead to the point that it is better suited for ABS? in fact it is power vs. better suited

B. The power of a language

[ ] more expressive: we can express complex problems more directly and with less overhead. note that this is domain-specifix: the mechanisms of a language allow to create abstractions which solve the domain-specific problem. the better these mechanisms support one in this task, the more powerful the language is in the given domain. now we end up by defining what "better" support means [ ] one could in pronciple do system programming in haskell by provoding bindings to code written in c and / or assembly but when the program is dominated by calls to these bindings then one could as well work directly in these lower languages and saves one from the overhead of the bindings [ ] but very often a domain consists of multiple subdomains. [ ] my hypothesis is that haskell is not well suited for domains which are dominated by managing and manipulating a global mutable state through side-effects / effectful computations. examples are gui-programming and computer games (state spread accross GPU and cpu, user input,...). this does not mean that it is not possible to implememt these things in haskell (it has been done with some sucess) but that the solution becomes too complex at some point. [ ] conciesness [ ] low ceremony [ ] susceptibility to bugs [ ] verbosity [ ] reasoning about performance [ ] reasoning about space requirements

C. Measuring a language

Define scientific measures: e.g. Lines Of Code (show relation to Bugs & Defects, which is an objective measure: http://www.stevemcconnel
https://softwareengineering.stackexchange.com/questions/185660/is-the-average-number-of-bugs-per-loc-the-same-for-different-programmi
Book: Code Complete, https://www.mayerdan.com/ruby/2012/11/11/bugs-per-line-of-code-ratio), also experience reports by companies which show that Haskell has huge benefits when applied to the same domain of a previous implementation of a different language, post on stack overflow / research gate / reddit, read experience reports from http://cufp.org/2015/ Also need to show the problem of operational reasoning as opposed to denotational reasoning

D. The Abstraction Hierarchy

1st: Functional vs. Object Oriented 2nd: Haskell vs. Java 3rd: FrABS vs. Repast

II. Background

- do not introduce SIR model in that length, also don't discuss SD and ABS, only minimal definition of what we understand as ABS, ignore definition of SD completely - good introduction to pure functional programming in Haskell: this is VERY difficult as it is a VAST topic where one can get lost quickly. focus on the central concepts: no assignment, recursion, pattern matching, static type-system with higher-kinded polymorphism - focus on the benefits of the pure functional approach -¿ program looks very much like a specification -¿ can rule out bugs at compile time -¿ can guarantee reproducibility at compile time -¿ 2 update-strategies without the need of different -¿ testing using quickcheck, testing = writing program spec -¿ reasoning: TODO

A. Functional Reactive Programming

FRP is a paradigm for programming hybrid systems which combine continuous and discrete components. Time is explicitly modelled: there is a continuous and synchronous time flow. There have been many attempts to implement FRP in libraries which each has its benefits and deficits. The very first functional reactive language was Fran, a domain specific language for graphics and animation. At Yale FAL, Frob, Fvision and Fruit were developed. The ideas of them all have then culminated in Yampa, the most recent FRP library [1]. The essence of FRP with Yampa is that one describes the system in terms of signal functions in a declarative manner using the EDSL of Yampa. During execution the top level signal functions will then be evaluated and return new signal functions which act as continuations. A major design goal for FRP is to free the programmer from 'presentation' details by providing the ability to think in terms of 'modeling'. It is common that an FRP program is concise enough to also serve as a specification for the problem it solves [2].

Yampa has been used in multiple agent-based applications: [3] uses Yampa for implementing a robot-simulation, [4] implement the classical Space Invaders game using Yampa, [5] implements a Pong-clone, the thesis of [6] shows how Yampa can be used for implementing a Game-Engine, [7] implemented a 3D first-person shooter game with the style of Quake 3 in Yampa. Note that although all these applications don't focus explicitly on agents all of them inherently deal with kinds of agents which share properties of classical agents: game-entities, robots,... Other fields in which Yampa was successfully used

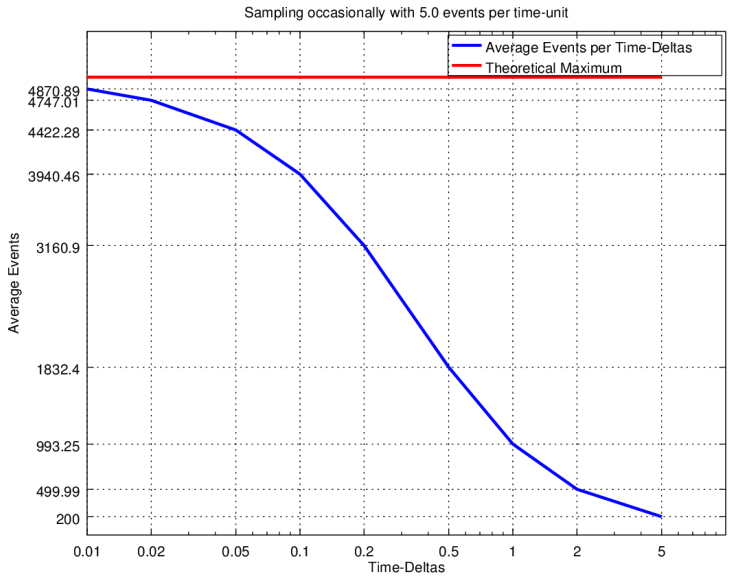


Fig. 1: Sampling the *occasional* function to visualize the influence of sampling frequencies on the event occurrence. Event-frequency of $\frac{1}{5}$ (average of 5 events per time-unit) with time-deltas of [5, 2, 1, $\frac{1}{2}$, $\frac{1}{5}$, $\frac{1}{10}$, $\frac{1}{20}$, $\frac{1}{50}$, $\frac{1}{100}$] running for 1000 time-units with 100 replications. The theoretical average is 5000 events within this time-frame.

were programming of synthesizers, network routers, computer music development and has been successfully combined with monads [8].

This leads to the conclusion that Yampa is mature, stable and suitable to be used in functional ABS. This and the reason that we have the in-house knowledge lets us focus on Yampa. Also it is out-of-scope to do a in-depth comparison of the many existing FRP libraries.

B. NetLogo

One can look at NetLogo as a functional approach to ABMS which comes with its own EDSL. Our approach differs fundamentally in the following way - untyped - no side-effects possible - no direct access to other agents, communication happens through asynchronous messages or synchronized conversations - powerful time-semantics which NetLogo completely lacks

III. Sampling the System

discuss the importance of sampling the system. When sampling the system, the correct time-delta must be selected which depends on the highest frequency which occurs in a time-reactive function in the whole system. For example in the FrSIR model we want infected agents to make on average contact with 5 other agents per time-unit, which means with a frequency of $frac15$. This functionality is built on Yampas function occasionally which behaviour we investigated under differing time-deltas with the above frequency. In this investigation we simply sampled occasionally with different time-deltas for a duration of 1000 time-units and the event-frequency of $frac15$. The results can be seen in Figure 1 and are quite striking. The plot clearly shows that occasionally needs quite high sampling frequency even for comparatively low event-frequency - this becomes of course worse for higher event-frequencies.

The other time-reactive function which occurs in the FrSIR model is the timed transition from infected to recovered which occurs on average with an exponential random-distribution after 15 time-units. This functionality is built on a custom implementation of Yampas after which creates an event after a time-out of the passed in time-duration drawn from an exponential random-distribution. Clearly this function has different semantics as although it also continuously emit events over time - NoEvent before the time was hit, and Event x after the time hit - the relevant point is that it switches to Event at some discrete point in time. This is implemented as simply adding up the time-deltas until the accumulator is GE than the previously drawn exponential time-out. We also investigated the behaviour of this function under varying time-deltas using a time-out of 15 (drawn from an exponential distribution within the function). Our approach was to sample the afterExp until an event occurs (this is one of the occasions where lazy evaluation really shines as one simply repeats the time-delta stream forever but then searches for the first occurrence of an event, which MUST occur at some point due to mathematical exponential distribution

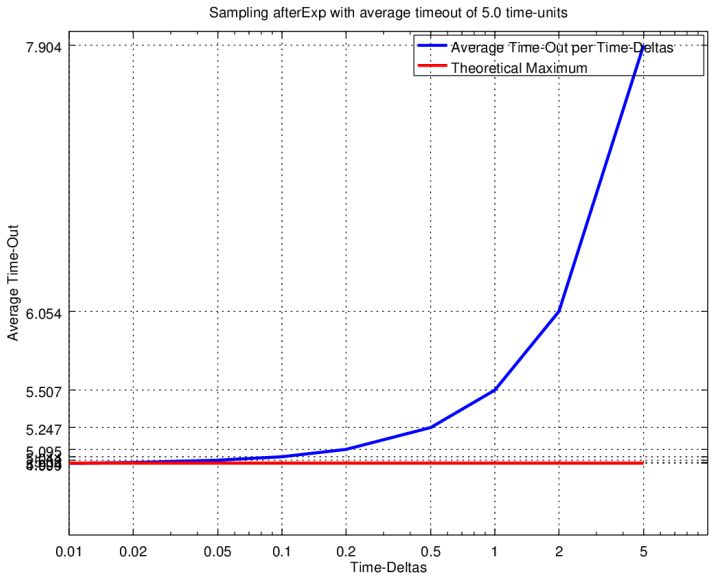


Fig. 2: Sampling the *afterExp* function to visualize the influence of sampling frequencies on the occurrence of the time-out event. Average time-out of 5 with time-deltas of [5, 2, 1, $\frac{1}{2}$, $\frac{1}{5}$, $\frac{1}{10}$, $\frac{1}{20}$, $\frac{1}{50}$, $\frac{1}{100}$] running 10,000 replications.

and our parameters to it, so it will always terminate) and then see when it has occurred. We run this with 10,000 replications with different random-number seeds and average the resulting times. The results can be seen in Figure 2. The result is striking in another way: this function seems to be pretty invariant to the time-deltas, for obvious reasons: we are basically just interested in the ”after”-condition of the whole semantics whereas in occasionally we are interested in the ”repeatedly”-conditions. If we under the afterExp then we can be off by one time-delta. If we under sample occasionally we keep loosing events, the close time-delta and event-frequency are, the more we lose. Of course afterExp can also be used for very short time-outs e.g. $frac15$. We have investigated the behaviour of this function for various time-deltas as well as seen in Figure **??**. Here the result is much more striking and shows that afterExp is vulnerable to small time-outs as well as occasionally. To show that occasionally is not vulnerable to very low frequencies of e.g. one event every 5 time-steps we plotted the behaviour of this under varying time-steps in Figure 4. The result shows that for low frequencies occasionally works fine with larger time-deltas

A. *Frequency of an ABS*

TODO: can we derive a formula to calculate the optimal time-delta for a given agent-based model?

B. *Super-Sampling*

of course performance is a big issue and it decreases as time-deltas get smaller and smaller. if we could perform subsampling just for the given high-frequency function with the remaining system running in lower frequency then we could achieve substantial performance-increase.

formula of calculating-steps = steps per time-unit * time-to-run-the-simulation

The problem is that *embed* does not really help because when running a sf with it, the signal-functions time does not advance. Thus we implemented a new signal-function which allows us to super-sample another signal-function.

```
superSampling :: Int -> SF a b -> SF a [b]
```

It takes the number of super-samples *n* and the signal-function *sf* to sample and returns a new signal-function which performs the super-sampling. It does this by evaluating sf for n times with dt of $\frac{dt}{n}$ and the same input argument a for all n evaluations. At time 0 no super-sampling is done and just a single output of sf is calculated. A list of b is returned with length of n containing the result of the n evaluations of sf. If 0 or less super samples are requested exactly one is calculated.

We ran tests super-sampling both *occasionally* (Figure 5, Figure 6) and *afterExp* (Figure). They work the same way as above except that now the time-delta is fixed to 1.0 but using increasing numbers of super-samples. The results are as expected: as the number of super-samples increase, so increases the accuracy.

At first this might not seem to be a real win as we still need to calculate a big number of samples every time. The big win comes though when these super-sampled signal-functions are embedded in a larger system which could run on a comparatively

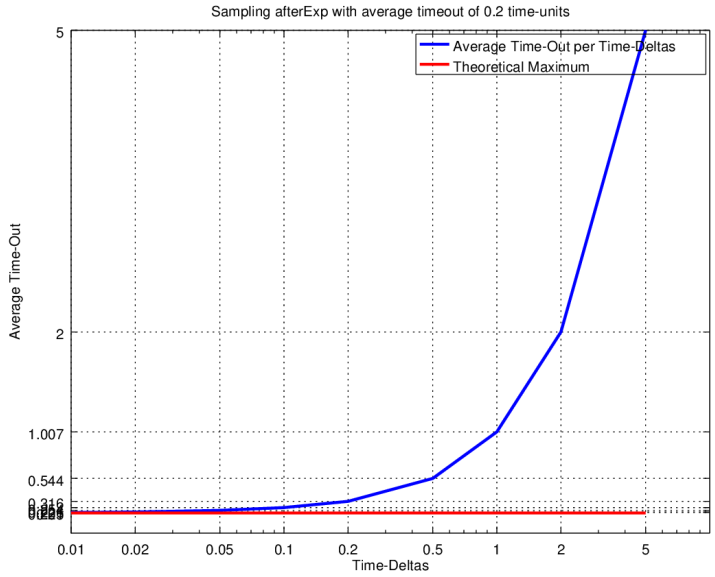


Fig. 3: Sampling the *afterExp* function to visualize the influence of sampling frequencies on the occurrence of the event. Average time-out of 0.2 with time-deltas of [5, 2, 1, $\frac{1}{2}$, $\frac{1}{5}$, $\frac{1}{10}$, $\frac{1}{20}$, $\frac{1}{50}$, $\frac{1}{100}$] running 10,000 replications.

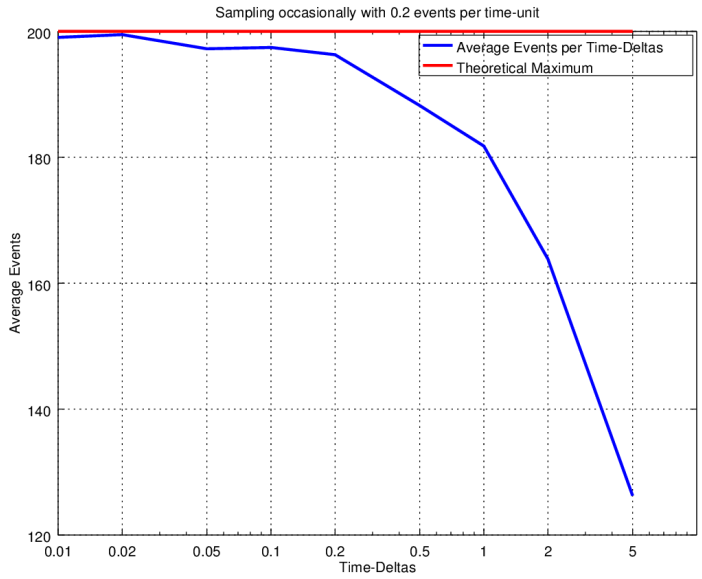


Fig. 4: Sampling the *occasional* function to visualize the influence of sampling frequencies on the event occurrence. Event-frequency of 5 (average of 0.2 events per time-unit) with time-deltas of [5, 2, 1, $\frac{1}{2}$, $\frac{1}{5}$, $\frac{1}{10}$, $\frac{1}{20}$, $\frac{1}{50}$, $\frac{1}{100}$] running for 1000 time-units with 100 replications. The theoretical average is 200 event within this time-frame.

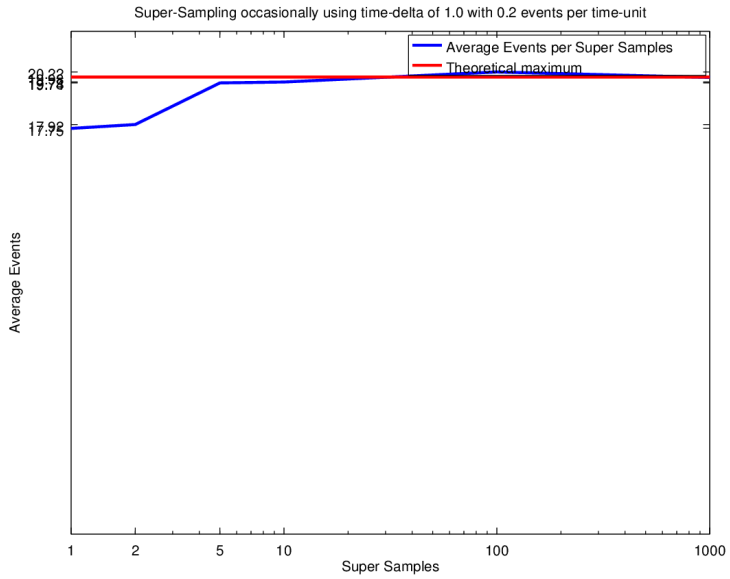


Fig. 5: Super-Sampling the *occasional* function to visualize the influence of increasing number of super-samples on the event occurrence. Event-frequency of 5 (average of 0.2 events per time-unit) with time-delta of 1, with super-samples of [1, 2, 5, 10, 100, 1000] running for 100 time-units with 100 replications. The theoretical average is 20 event within this time-frame.

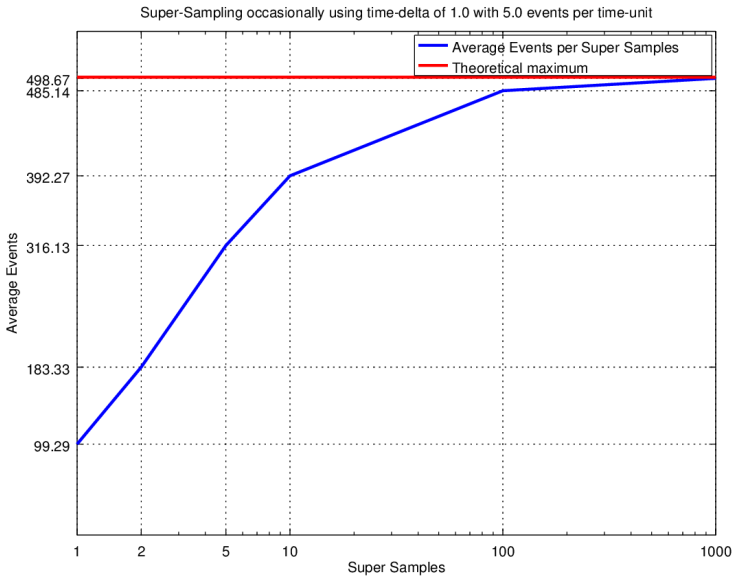


Fig. 6: Super-Sampling the *occasional* function to visualize the influence of increasing number of super-samples on the event occurrence. Event-frequency of $\frac{1}{5}$ (average of 5 events per time-unit) with time-delta of 1, with super-samples of [1, 2, 5, 10, 100, 1000] running for 100 time-units with 100 replications. The theoretical average is 20 event within this time-frame.

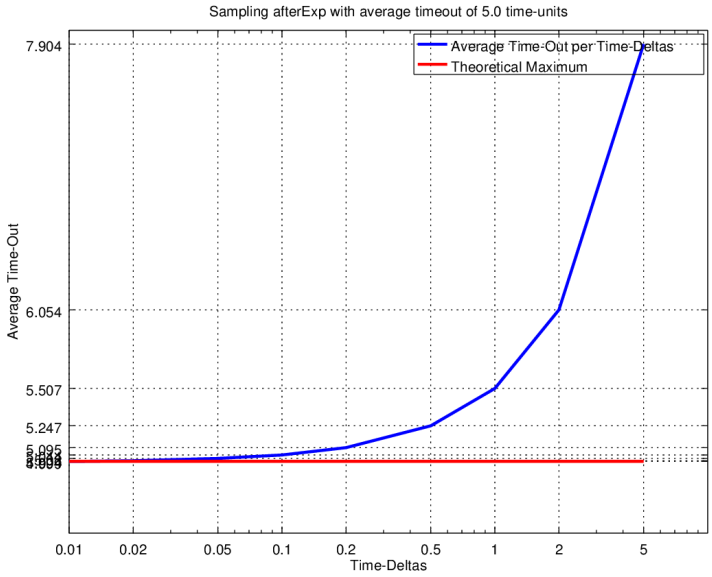


Fig. 7: Super-Sampling the *afterExp* function to visualize the influence of increasing number of super-samples on the average time-out. Average time-out of 5 with time-delta of 1, with super-samples of [1, 2, 5, 10, 100, 1000], running 10,000 replications.

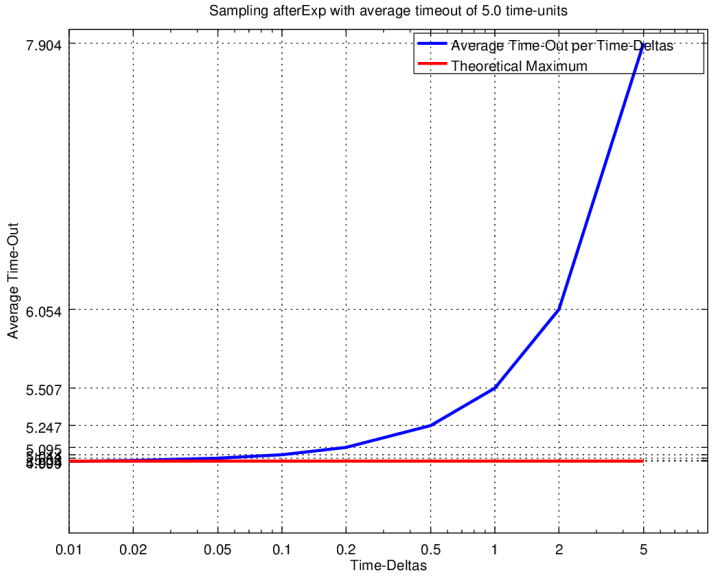


Fig. 8: Super-Sampling the *afterExp* function to visualize the influence of increasing number of super-samples on the average time-out. Average time-out of 0.2 with time-delta of 1, with super-samples of [1, 2, 5, 10, 100, 1000], running 10,000 replications.

low frequency of e.g. 1 dt. So we are then increasing the sampling-frequency just where we need it and keep the frequency low where it is not required.

TODO: report results of using it in FrSIR

IV. Reasoning

[ ] reasoning: sd is always reproducible because all runs outside the io moand. this means that potential RNG seeds are always the same and do not depend on outside states e.g. time [ ] for the agents this means that repeated runs with the same seed are guaranteed to result in the same dynamics as there are again no possibilities to introduce e.g. random seeds or values from the outside which may change between runs like time, files,...

-¿ spatial and network behaviour is EXACTLY the same except selection of neighbours =¿ what can we reason about it regarding the dynamics?

TODO: can we formally show that the SIR approximates the SD model?

-¿ my emulation of SD using ABS is really an implementation of the SD model and follows it - they are equivalent -¿ my ABS implementation is the same as / equivalent to the SD emulation =¿ thus if i can show that my SD emulation is equlas to the SD model =¿ AND that the ABS implementation is the same as the SD emulation =¿ THEN the ABS implementation is an SD implementation, and we have shown this in code for the first time in ABS

i need to get a deep understanding in writing correct code and reasoning about correctness in Haskell - look into papers: https://wiki.haskell.org/Research_papers/Testing_and_correctness https://www.reddit.com/r/haskell/comments/4tagq3/examples_of_realworld_haskell_usage_where/ https://stackoverflow.com/questions/4077970/can-haskell-functions-be-proved-model-checked-verified-with-correctne

V. Testing

TODO: property-based and unit testing of a model

TODO: modular testing of agents

TODO: reasoning about dynamics in code would allow us to cut substantial calculations: can make assumptions about dynamics without actually running it. is it even possible?

[9]

-¿ testing replies: an infected agent always replies with contact infected, a recovered never replies, a susceptible never replies -¿ testing contacts: a susceptible contacts occaaionally, an infected contacts occasionally, a recovered never contacts. -¿ testing: an infected agent recovering after (same as falling ball?) -¿ testing: a susceptible agent gets infected after infect contact -¿ testing: can we measure the occasional distribution to verify?

VI. Time-Traveling

[10]

[ ] time in FrABS: when 0dt then still actions can occur when not relying on time semantics [ ] what about time-travel in abms for introspection during running it? this is much easier in FrABS

VII. Results

A. System Dynamics SIR Simulation

TODO: SIR dynamics using SD with FrABS, using 1.0, 0.5, 0.1 and 0.01 sampling

B. Agent-based SIR Simulation

TODO: SIR dynamics using ABS with FrABS, using 1.0, 0.5, 0.1 and 0.05 sampling with 1000 agents

TODO: need to do the same thing as AnyLogic. maybe the problem is sampling as well as replications as number of agents, which leaves us in a pretty bad shape performance wise. -¿ how many agents? are 1000 enough? -¿ how many replications? are 32 enough? -¿ what is the sampling interval? 0.1 enough or do we have to go below? 1000 agents with 16 replications and 0.05 dt seems to deliver pretty good results need enough initially infected agents, otherwise wouldn't work

VIII. Discussion

advantages: - no side-effects within agents leads to much safer code - edsl for time-semantics - declarative style: agent-implementation looks like a model-specification - reasoning and verification - sequential and parallel - powerful time-semantics - arrowized programming is optional and only required when utilizing yampas time-semantics. if the model does not rely on time-semantics, it can use monadic-programming by building on the existing monadic functions in the EDSL which allow to run in the State-Monad which simplifies things very much - when to use yampas arrowized programing: time-semantics, simple state-chart agents - when not using yampas facilities: in all the other cases e.g. SugarScape is such a case as it proceeds in unit time-steps and all agents act in every time-step - can implement System Dynamics building on Yampas facilities with total ease - get replications for free without having to worry about side-effects and can even run them in parallel without

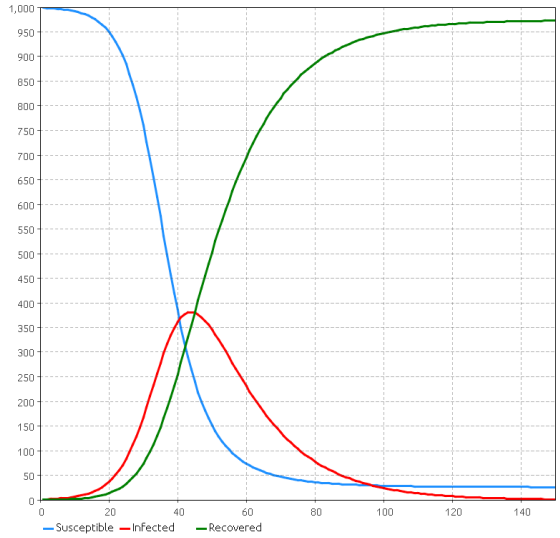
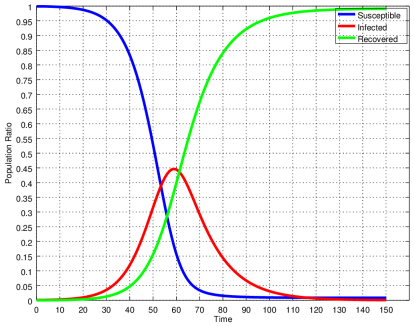
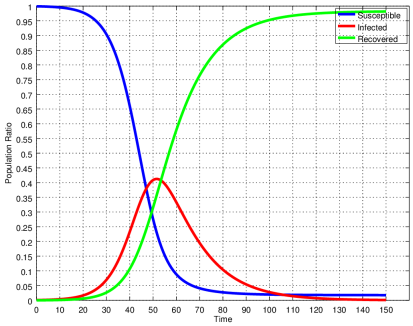


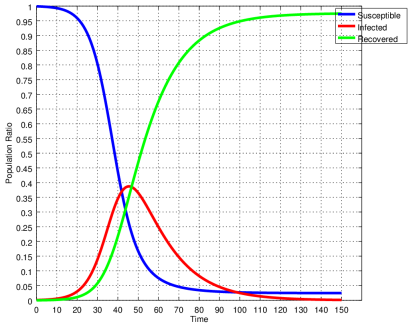
Fig. 9: Dynamics of the SIR compartment model using the System Dynamics approach generated with generated with AnyLogic Personal Learning Edition 8.1.0. Population Size N = 1000, contact rate $\beta = 1/5$, infectivity $\gamma = 0.05$, illness duration $\delta = 15$ with initially 1 infected agent, run for 150 time-units.



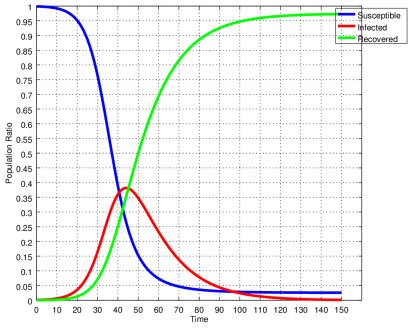
(a) Δ 1.0



(b) Δ 0.5



(c) Δ 0.1



(d) Δ 0.01

Fig. 10: Dynamics of the SIR compartment model using the System Dynamics approach generated with generated with FrABS with the same model parameters as in Figure 9 (population Size N = 1000, contact rate $\beta = 1/5$, infectivity $\gamma = 0.05$, illness duration $\delta = 15$ with initially 1 infected agent, run for 150 time-units.).

headaches - cant mess around with time because delta-time is hidden from you (intentional design-decision by Yampa). this would be only very difficult and cumbersome to achieve in an object-oriented approach. TODO: experiment with it in Java - how could we actually implement this? I think it is impossible: may only achieve this through complicated application of patterns and inheritance but then has the problem of how to update the dt and more important how to deal with functions like integral which accumulates a value through closures and continuations. We could do this in OO by having a general base-class e.g. ContinuousTime which provides functions like updateDt and integrate, but we could only accumulate a single integral value. - reproducibility statically guaranteed - cannot mess around with dt - code == specification - rule out serious class of bugs - different time-sampling leads to different results e.g. in wildfire & SIR but not in Prisoners Dilemma. why? probabilistic time-sampling? - reasoning about equivalence between SD and ABS implementation in the same framework - recursive implementations

- we can statically guarantee the reproducibility of the simulation because: no side effects possible within the agents which would result in differences between same runs (e.g. file access, networking, threading), also timedeltas are fixed and do not depend on rendering performance or userinput

disadvantages: - performance is low - reasoning about performance is very difficult - very steep learning curve for non-functional programmers - learning a new EDSL - think ABMS different: when to use async messages, when to use sync conversations

[ ] important: increasing sampling freqzency and increasing number of steps so that the same number of simulation steps are executed should lead to same results. but it doesnt. why? [ ] hypothesis: if time-semantics are involved then event ordering becomes relevant for emergent patterns. there are no tine semantics in heroes and cowards but in the prisoners dilemma [ ] can we implement different types of agents interacting with each other in the same simulation ? with different behaviour funcs, digferent state? yes, also not possible in NetLogo to my knowledge. but they must have the same messages, emvironment

[ ] Hypothesis: we can combine with FrABS agent-based simulation and system dynamics

ACKNOWLEDGMENTS

The authors would like to thank I. Perez, H. Nilsson, J. Greensmith and S. Venkatesan for constructive comments and valuable discussions.

REFERENCES

[1] H. Nilsson, A. Courtney, and J. Peterson, "Functional Reactive Programming, Continued," in *Proceedings of the 2002 ACM SIGPLAN Workshop on Haskell*, ser. Haskell '02. New York, NY, USA: ACM, 2002, pp. 51–64. [Online]. Available: http://doi.acm.org/10.1145/581690.581695

[2] Z. Wan and P. Hudak, "Functional Reactive Programming from First Principles," in *Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation*, ser. PLDI '00. New York, NY, USA: ACM, 2000, pp. 242–252. [Online]. Available: http://doi.acm.org/10.1145/349299.349331

[3] P. Hudak, A. Courtney, H. Nilsson, and J. Peterson, "Arrows, Robots, and Functional Reactive Programming," in *Advanced Functional Programming*, ser. Lecture Notes in Computer Science, J. Jeuring and S. L. P. Jones, Eds. Springer Berlin Heidelberg, 2003, no. 2638, pp. 159–187, dOI: 10.1007/978-3-540-44833-4_6. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-540-44833-4_6

[4] A. Courtney, H. Nilsson, and J. Peterson, "The Yampa Arcade," in *Proceedings of the 2003 ACM SIGPLAN Workshop on Haskell*, ser. Haskell '03. New York, NY, USA: ACM, 2003, pp. 7–18. [Online]. Available: http://doi.acm.org/10.1145/871895.871897

[5] H. Nilsson and I. Perez, "Declarative Game Programming: Distilled Tutorial," in *Proceedings of the 16th International Symposium on Principles and Practice of Declarative Programming*, ser. PPDP '14. New York, NY, USA: ACM, 2014, pp. 159–160. [Online]. Available: http://doi.acm.org/10.1145/2643135.2643160

[6] G. Meisinger, "Game-Engine-Architektur mit funktional-reaktiver Programmierung in Haskell/Yampa," Master, Fachhochschule Obersterreich - Fakultt fr Informatik, Kommunikation und Medien (Campus Hagenberg), Austria, 2010.

[7] C. Mun Hon, "Functional Programming and 3d Games," Ph.D. dissertation, University of New South Wales, Sydney, Australia, 2005.

[8] I. Perez, M. Brenz, and H. Nilsson, "Functional Reactive Programming, Refactored," in *Proceedings of the 9th International Symposium on Haskell*, ser. Haskell 2016. New York, NY, USA: ACM, 2016, pp. 33–44. [Online]. Available: http://doi.acm.org/10.1145/2976002.2976010

[9] I. Perez and H. Nilsson, "Testing and Debugging Functional Reactive Programming," *Proc. ACM Program. Lang.*, vol. 1, no. ICFP, pp. 2:1–2:27, Aug. 2017. [Online]. Available: http://doi.acm.org/10.1145/3110246

[10] I. Perez, "Back to the Future: Time Travel in FRP," in *Proceedings of the 10th ACM SIGPLAN International Symposium on Haskell*, ser. Haskell 2017. New York, NY, USA: ACM, 2017, pp. 105–116. [Online]. Available: http://doi.acm.org/10.1145/3122955.3122957

[11] J. M. Epstein and R. Axtell, *Growing Artificial Societies: Social Science from the Bottom Up*. Washington, DC, USA: The Brookings Institution, 1996.

[12] J. M. Epstein, *Agent_Zero: Toward Neurocognitive Foundations for Generative Social Science*. Princeton University Press, Feb. 2014, google-Books-ID: VJEpAgAAQBAJ.

[13] T. Schelling, "Dynamic models of segregation," *Journal of Mathematical Sociology*, vol. 1, 1971.

[14] M. A. Nowak and R. M. May, "Evolutionary games and spatial chaos," *Nature*, vol. 359, no. 6398, pp. 826–829, Oct. 1992. [Online]. Available: http://www.nature.com/nature/journal/v359/n6398/abs/359826a0.html

[15] B. A. Huberman and N. S. Glance, "Evolutionary games and computer simulations," *Proceedings of the National Academy of Sciences*, vol. 90, no. 16, pp. 7716–7718, Aug. 1993. [Online]. Available: http://www.pnas.org/content/90/16/7716

[16] U. Wilensky and W. Rand, *An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NETLogo*. MIT Press, 2015. [Online]. Available: https://www.amazon.co.uk/Introduction-Agent-Based-Modeling-Natural-Engineered/dp/0262731894

[17] T. Breuer, M. Jandaka, M. Summer, and H.-J. Vollbrecht, "Endogenous leverage and asset pricing in double auctions," *Journal of Economic Dynamics and Control*, vol. 53, no. C, pp. 144–160, 2015. [Online]. Available: http://econpapers.repec.org/article/eeedyncon/v_3a53_3ay_3a2015_3ai_3ac_3ap_3a144-160.htm

[18] J. B. Gilmer, Jr. and F. J. Sullivan, "Recursive Simulation to Aid Models of Decision Making," in *Proceedings of the 32Nd Conference on Winter Simulation*, ser. WSC '00. San Diego, CA, USA: Society for Computer Simulation International, 2000, pp. 958–963. [Online]. Available: http://dl.acm.org/citation.cfm?id=510378.510515

[19] N. Bostrom, “Are We Living in a Computer Simulation?” *The Philosophical Quarterly*, vol. 53, no. 211, pp. 243–255, 2003. [Online]. Available: http://dx.doi.org/10.1111/1467-9213.00309

[20] E. Steinhart, “Theological Implications of the Simulation Argument,” *Ars Disputandi: The Online Journal for Philosophy of Religion*, vol. 10, pp. 23–37, 2010.

APPENDIX A
EXAMPLES

In this appendix we give a list of all the examples we have implemented and discuss implementation details relevant [1]. The examples were implemented as use-cases to drive the development of *FrABS* and to give code samples of known models which show how to use this new approach. Note that we do not give an explanation of each model as this would be out of scope of this paper but instead give the major references from which an understanding of the model can be obtained.

We distinguish between the following attributes

- Implementation - Which style was used? Either Pure, Monadic or Reactive. Examples could have been implemented in all of them.
- Yampa Time-Semantics - Does the implemented model make use of Yampas time-semantics e.g. occasional, after,...? Yes / No.
- Update-Strategy - Which update-strategy is required for the given example? It is either Sequential or Parallel or both. In the case of Sequential Agents may be shuffled or not.
- Environment - Which kind of environment is used in the given example? Possibilities are 2D/3D Discrete/Continuous or Network. In case of a Parallel Update-Strategy, collapsing may become necessary, depending on the semantics of the model. Also it is noted if the environment has behaviour. Note that an implementation may also have no environment which is noted as None. Although every model implemented in *FrABs* needs to set up some environment, it is not required to use it in the implementation.
- Recursive - Is this implementation making use of the recursive features of *FrABS* Yes/No (only available in sequential updating)?
- Conversations - Is this implementation making use of the conversations features of *FrABS* Yes/No (only available in sequential updating)?

A. Sugarscape

This is a full implementation of the famous Sugarscape model as described by Epstein & Axtell in their book [11]. The model description itself has no real time-semantics, the agents act in every time-step. Only the environment may change its behaviour after a given number of steps but this is easily expressed without time-semantics as described in the model by Epstein & Axtell [2].

Implementation	Pure, Monadic
Yampa Time-Semantics	No
Update-Strategy	Sequential, shuffling
Environment	2D Discrete, behaviour
Recursive	No
Conversations	Yes

B. Agent_Zero

This is an implementation of the *Parable 1* from the book of Epstein [12].

Implementation	Pure, Monadic
Yampa Time-Semantics	No
Update-Strategy	Parallel, Sequential, shuffling
Environment	2D Discrete, behaviour, collapsing
Recursive	No
Conversations	No

C. Schelling Segregation

This is an implementation of [13] with extended agent-behaviour which allows to study dynamics of different optimization behaviour: local or global, nearest/random, increasing/binary/future. This is also the only 'real' model in which the recursive features were applied [3].

[1] The examples are freely available under https://github.com/thalerjonathan/phd/tree/master/coding/libraries/frABS/examples

[2] Note that this implementation has about 2600 lines of code which - although it includes both a pure and monadic implementation - is significant lower than e.g. the Java-implementation http://sugarscape.sourceforge.net/ with about 6000. Of course it is difficult to compare such measures as we do not include FrABS itself into our measure.

[3] The example of Recursive ABS is just a plain how-to example without any real deeper implications.

Implementation	Pure
Yampa Time-Semantics	No
Update-Strategy	Sequential, shuffling
Environment	2D Discrete
Recursive	Yes (optional)
Conversations	No

D. Prisoners Dilemma

This is an implementation of the Prisoners Dilemma on a 2D Grid as discussed in the papers of [14], [15] and TODO: cite my own paper on update-strategies.

TODO: implement

E. Heroes & Cowards

This is an implementation of the Heroes & Cowards Game as introduced in [16] and discussed more in depth in TODO: cite my own paper on update-strategies.

TODO: implement

F. SIRS

This is an early, non-reactive implementation of a spatial version of the SIRS compartment model found in epidemiology. Note that although the SIRS model itself includes time-semantics, in this implementation no use of Yampas facilities were made. Timed transitions and making contact was implemented directly into the model which results in contacts being made on every iteration, independent of the sampling time. Also in this sample only the infected agents make contact with others, which is not quite correct when wanting to approximate the System Dynamics model (see below). It is primarily included as a comparison to the later implementations (Fr*SIRS) of the same model which make full use of *FrABS* and to see the huge differences the usage of Yampas time-semantics can make.

Implementation	Pure, Monadic
Yampa Time-Semantics	No
Update-Strategy	Parallel, Sequential with shuffling
Environment	2D Discrete
Recursive	No
Conversations	No

G. Reactive SIRS

This is the reactive implementations of both 2D spatial and network (complete graph, Erdos-Renyi and Barbasi-Albert) versions of the SIRS compartment model. Unlike SIRS these examples make full use of the time-semantics provided by Yampa and show the real strength provided by *FrABS*.

Implementation	Reactive
Yampa Time-Semantics	Yes
Update-Strategy	Parallel
Environment	2D Discrete, Network
Recursive	No
Conversations	No

H. System Dynamics SIR

This is an emulation of the System Dynamics model of the SIR compartment model in epidemiology. It was implemented as a proof-of-concept to show that *FrABS* is able to implement even System Dynamic models because of its continuous-time and time-semantic features. Connections between stocks & flows are hardcoded, after all System Dynamics completely lacks the concept of spatial- or network-effects. Note that describing the implementation as Reactive may seem not appropriate as in System Dynamics we are not dealing with any events or reactions to it - it is all about a continuous flow between stocks. In this case we wanted to express with Reactive that it is implemented using the Arrowized notion of Yampa which is required when one wants to use Yampas time-semantics anyway.

Implementation	Reactive
Yampa Time-Semantics	Yes
Update-Strategy	Parallel
Environment	None
Recursive	No
Conversations	No

I. WildFire

This is an implementation of a very simple Wildfire model inspired by an example from AnyLogic™with the same name.

Implementation	Reactive
Yampa Time-Semantics	Yes
Update-Strategy	Parallel
Environment	2D Discrete
Recursive	No
Conversations	No

J. Double Auction

This is a basic implementation of a double-auction process of a model described by [17]. This model is not relying on any environment at the moment but could make use of networks in the future for matching offers.

Implementation	Pure, Monadic
Yampa Time-Semantics	No
Update-Strategy	Parallel
Environment	None
Recursive	No
Conversations	No

K. Policy Effects

This is an implementation of a model inspired by Uri Wilensky [4]: "Imagine a room full of 100 people with 100 dollars each. With every tick of the clock, every person with money gives a dollar to one randomly chosen other person. After some time progresses, how will the money be distributed?"

Implementation	Monadic
Yampa Time-Semantics	No
Update-Strategy	Parallel
Environment	Network
Recursive	No
Conversations	No

L. Proof of concepts

1) Recursive ABS: This example shows the very basics of how to implement a recursive ABS using *FrABS*. Note that recursive features only work within the sequential strategy.

Implementation	Pure
Yampa Time-Semantics	No
Update-Strategy	Sequential
Environment	None
Recursive	Yes
Conversations	No

2) Conversation: This example shows the very basics of how to implement conversations in *FrABS*. Note that conversations only work within the sequential strategy.

Implementation	Pure
Yampa Time-Semantics	No
Update-Strategy	Sequential
Environment	None
Recursive	No
Conversations	Yes

[4]http://www.decisionsciencenews.com/2017/06/19/counterintuitive-problem-everyone-room-keeps-giving-dollars-random-others-youll-never-guess-happens-next/

APPENDIX B
RECURSIVE AGENT-BASED SIMULATION

The idea for this paper arose from my idea of *anticipating agents*, which can project their actions in the future. Because this paper is not as polished as the draft for programming paradigms, we opted not to include it as an appendix and only give its basic ideas and results for the experiments conducted so far. Note that we were not able to find any research regarding recursive ABS [5]. In Recursive ABS agents are able to halt time and 'play through' an arbitrary number of actions, compare their outcome and then to resume time and continue with a specifically chosen action e.g. the best performing or the one in which they haven't died. More precisely, what we want is to give an agent the ability to run the simulation recursively a number of times where the this number is not determined initially but can depend on the outcome of the recursive simulation. So Recursive ABS gives each Agent the ability to run the simulation locally from its point of view to anticipate its actions in the future and change them in the present. We investigate the famous Schelling Segregation [13] and endow our agents with the ability to project their actions into the future by recursively running simulations. Based on the outcome of the recursions they are then able to determine whether their move increases their utility in the future or not. The main finding for now is that it does not increase the convergence speed to equilibrium but can lead to extreme volatility of dynamics although the system seems to be near to complete equilibrium. In the case of a 10x10 field it was observed that although the system was nearly in its steady state - all but one agent were satisfied - the move of a single agent caused the system to become completely unstable and depart from its near-equilibrium state to a highly volatile and unstable state.

This approach of course rises a few questions and issues. The main problem of our approach is that, depending on ones viewpoint, it is violating the principles of locality of information and limit of computing power. To recursively run the simulation the agent which initiates the recursion is feeding in all the states of the other agents and calculates the outcome of potentially multiple of its own steps, each potentially multiple recursion-layers deep and each recursion-layer multiple time-steps long. Both requires that each agent has perfect information about the complete simulation *and* can compute these 3-dimensional recursions, which scale exponentially. In the social sciences where agents are often designed to have only very local information and perform low-cost computations it is very difficult or impossible to motivate the usage of recursive simulations - it simply does not match the assumptions of the real world, the social sciences want to model. In general simulations, where it is much more commonly accepted to assume perfect information and potentially infinite amount of computing power this approach is easily motivated by a constructive argument: it is possible to build, thus we build it. Another fundamental question regards the meaning and epistemology behind an entity running simulations. Of course, this strongly depends on the context: in ACE it may be understood as a search for optimizing behaviour, in Social Simulation it may be interpreted as a kind of free will: the agent who is initiating the recursion can be seen as 'knowing' that it is running inside a simulation, thus in this context free will is seen as being able to anticipate ones actions and change them. When talking about recursion it is always the question of the depth of the recursion and because as we are running on computers we need to terminate at some point. Accelerating Turing machines (also known as Zeno Machine) are theoretically able to calculate an infinite regress but this raises again epistemological questions and can be seen as having religious character as discussed e.g. in Tiplers Omega Point, Bostroms simulation argument [19] and its theological implications [20]. So the ultimate question this research leaves is what the outcome would be when running a recursive ABS on a Zeno Machine/Accelerated Turing Machine? [6]

At the moment this idea lies dormant as the intention was just to develop it far enough to give a proof-of-concept and see some results. Having achieved this we arrived at the conclusion, that the results are not really ground-breaking. This stems from the fact that Schelling segregation is not the best model to demonstrate this technique and that we are thus lacking the right model in which recursive ABS is the real killer-feature. Also to pursue this direction further and treat it in-depth, would require much more time and give the PhD a complete different spin. Still it is useful in supporting our move towards pure functional ABS as we are convinced that recursion is comparably easy to implement because the language is built on it and due to the lack of side-effects [7].

TODO: add the dynamics of recursive schelling segregation

[5] We found a paper on recursive simulation in general [18] which focuses on military simulation implemented in C++. Its main findings are that deterministic models seem to benefit significantly from using recursions of the simulation for the decision making process and that when using stochastic models this benefit seems to be lost.

[6] Anyway this would mean we have infinite amount of computing power - I am sure that in this case we don't worry the slightest about recursive ABS any more.

[7] Actually implementing it was *really hard* but we wouldn't dare to implement this into an object-oriented language or into an object-oriented ABS framework.