Pure functional programming in Agent-Based Simulation

Jonathan Thaler

University of Nottingham, Ningbo, China

AIOP Seminar 2019

Research Questions

- **How** can we implement Agent-Based Simulation in (pure) functional programming using Haskell?
- **What** are the benefits and drawbacks?

Hypotheses

- Arrive at deeper understanding of the computational aspects of ABS due to functional programmings focus on structure in computing.
- Less bugs, more likely to be correct: stronger guarantees at compile time
- Better support for concurrency
- Natural fit for randomized property-based testing
- Step toward dependently typed ABS

Agent-Based Simulation (ABS)

Example

Simulate *the spread of an infectious disease in a city.*
What are the ***dynamics*** *(peak, duration of disease)?*

1. Start with population → Agents
2. Situated in City → Environment
3. Interacting with each other → Local interactions
4. Creating dynamics → Emergent system behaviour
5. Therefore ABS → Bottom-up approach

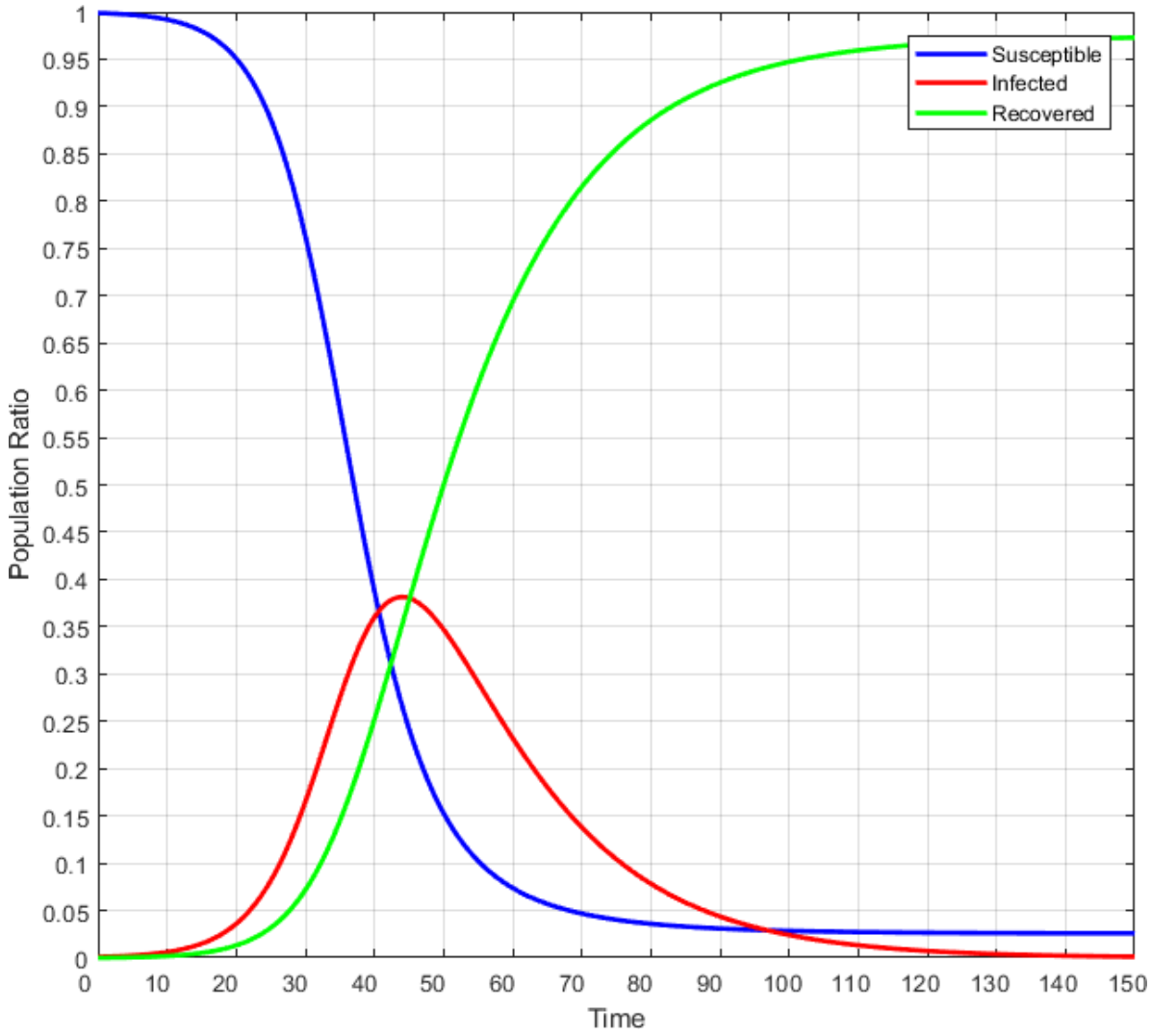
SIR Model

- Population size $N = 1,000$
- Contact rate $\beta = 5$
- Infection probability $\gamma = 0.05$
- Illness duration $\delta = 15$
- 1 initially infected agent

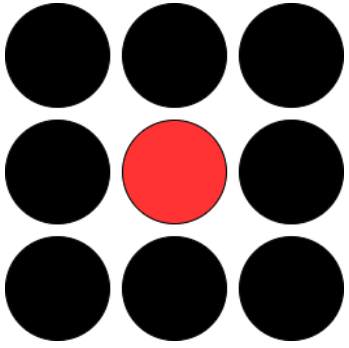
System Dynamics

Top-Down, formalised using Differential Equations, give rise to dynamics.

SIR Model Dynamics

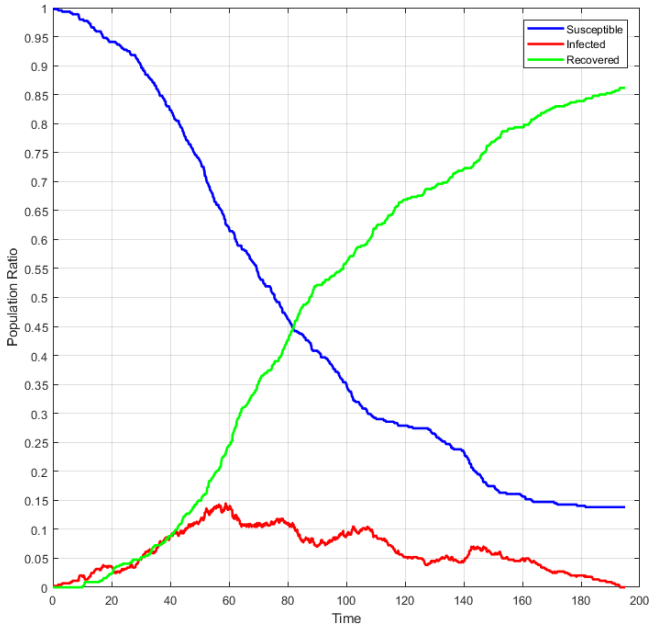


Defining Spatiality

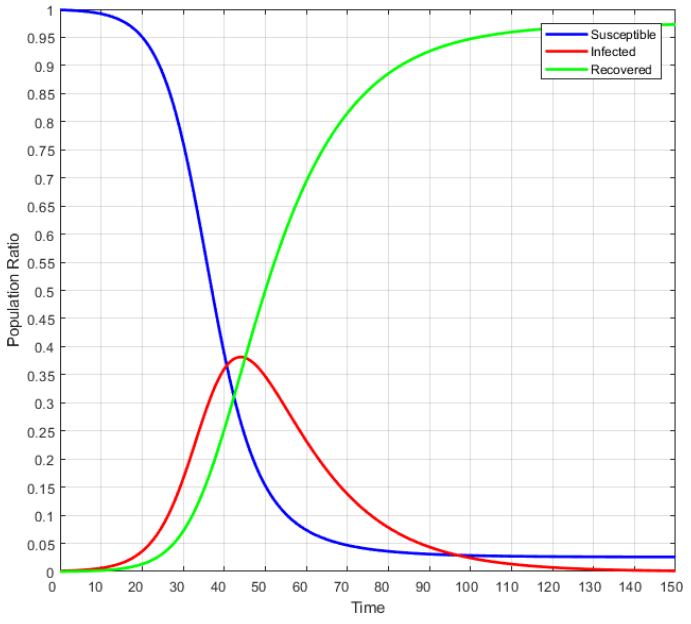


Moore Neighbourhood

Spatial Dynamics



Agent-Based



System Dynamics

Spatial Visualisation

TODO: explain monads: declare type of side-effects statically at

How to implement ABS?

Established, state-of-the-art approach in ABS

Object-Oriented Programming in Python, Java,...

We want (pure) functional programming

Purity, explicit about side-effects, declarative, reasoning, parallelism, concurrency, property-based testing,...

How can we do it?

Functional Reactive Programming

Arrowized Functional Reactive Programming (AFRP)

- Continuous- & discrete-time systems in FP
- Signal Function
- Events
- Effects like random-numbers, global state, concurrency
- *Arrowized* FRP using the *Dunai* library

Monadic Stream Functions (MSF)

Process over time

$$\mathit{SF}\ \alpha\ \beta \approx \mathit{Signal}\ \alpha \to \mathit{Signal}\ \beta$$
$$\mathit{Signal}\ \alpha \approx \mathit{Time} \to \alpha$$

Agents as Signal Functions

- Clean interface (input / output)
- Pro-activity by perceiving time

FRP combinators

Dynamic change of behaviour

```
switch :: SF inp (out, Event e)
       -> (e -> SF inp out)
       -> SF inp out
```

Stochastic event source

```
occasionally :: RandomGen g
             => g -> Time -> b -> SF a (Event b)
```

Random number stream

```
noiseR :: (RandomGen g, Random b)
       => (b, b) -> g -> SF a b
```

Infinitesimal delay (1 step)

```
iPre :: a -> SF a a
```

Conclusion

- Purity guarantees reproducibility
- Enforce and guarantee update semantics
- Performance low, applying STM to it

Thank You!