

Pure Functional Epidemics

An Agent-Based Approach

Jonathan Thaler Thorsten Altenkirch Peer-Olaf Siebers

University of Nottingham, United Kingdom

IFL 2018

Research Question(s)

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

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

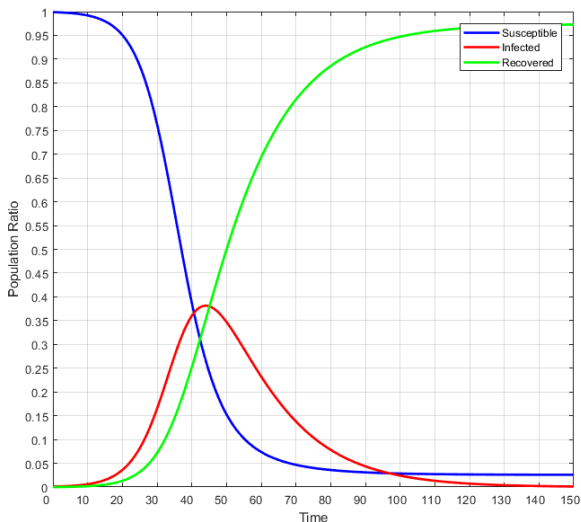


- Population size $N = 1,000$
- Contact rate $\beta = 0.2$
- Infection probability $\gamma = 0.05$
- Illness duration $\delta = 15$
- 1 initially infected agent

System Dynamics

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

SIR Model Dynamics



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

Functional Reactive Programming (FRP)

- Continuous- & discrete-time systems in functional programming
- Signal Function (SF)
⇒ process over time, maps signal to signal
- Events (deterministic & stochastic)
- Random-number streams
- *Arrowized* FRP using the *Yampa* library

First Steps

```
1  data SIRState = Susceptible | Infected | Recovered
2
3  type SIRAgent = SF [SIRState] SIRState
4
5  sirAgent :: RandomGen g => g -> SIRState -> SIRAgent
6  sirAgent g Susceptible = susceptibleAgent g
7  sirAgent g Infected    = infectedAgent g
8  sirAgent _ Recovered   = recoveredAgent
9
10 recoveredAgent :: SIRAgent
11 recoveredAgent = arr (const Recovered)
```


Susceptible Agent

```

1  susceptibleAgent :: RandomGen g => g -> SIRAgent
2  susceptibleAgent g = switch (susceptible g) (const (infectedAgent g))
3      where
4          susceptible :: RandomGen g => g -> SF [SIRState] (SIRState, Event ())
5          susceptible g = proc as -> do
6              makeContact <- occasionally g (1 / contactRate) () -< ()
7              if isEvent makeContact
8                  then (do
9                      -- draw random element from the list
10                     a <- drawRandomElemSF g -< as
11                     case a of
12                         Infected -> do
13                             -- returns True with given probability
14                             i <- randomBoolSF g infectivity -< ()
15                             if i
16                                 then returnA -< (Infected, Event ())
17                                 else returnA -< (Susceptible, NoEvent)
18                             -> returnA -< (Susceptible, NoEvent))
19                     else returnA -< (Susceptible, NoEvent)

```

Infected Agent

```
1  infectedAgent :: RandomGen g => g -> SIRAgent
2  infectedAgent g = switch infected (const recoveredAgent)
3    where
4      infected :: SF [SIRState] (SIRState, Event ())
5      infected = proc _ -> do
6        recEvt <- occasionally g illnessDuration () -< ()
7        let a = event Infected (const Recovered) recEvt
8        returnA -< (a, recEvt)
```

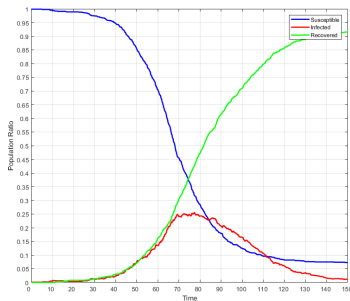
Running the Simulation

```

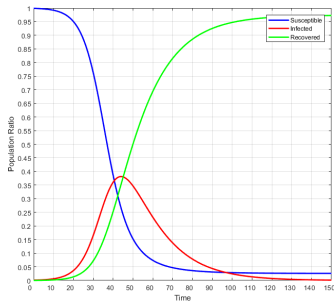
1  runSimulation :: RandomGen g => g -> Time -> DTime -> [SIRState] -> [[SIRState]]
2  runSimulation g t dt as = embed (stepSimulation sfs as) ((), dts)
3  where
4      steps      = floor (t / dt)
5      dts        = replicate steps (dt, Nothing)
6      n          = length as
7      (rngs, _)  = rngSplits g n [] -- unique rngs for each agent
8      sfs        = zipWith sirAgent rngs as
9
10 stepSimulation :: [SIRAgent] -> [SIRState] -> SF () [SIRState]
11 stepSimulation sfs as =
12     dpSwitch
13     -- feeding the agent states to each SF
14     (\_ sfs' -> (map (\sf -> (as, sf)) sfs'))
15     -- the signal functions
16     sfs
17     -- switching event, ignored at t = 0
18     (switchingEvt >>> notYet)
19     -- recursively switch back into stepSimulation
20     stepSimulation
21 where
22     switchingEvt :: SF ((), [SIRState]) (Event [SIRState])
23     switchingEvt = arr (\ (_, newAs) -> Event newAs)

```

Dynamics $\Delta t = 0.1$

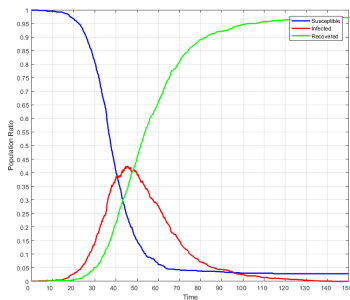


(a) Agent-Based approach

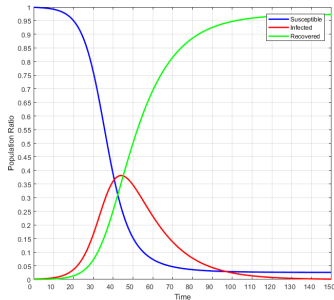


(b) System Dynamics

Dynamics $\Delta t = 0.01$



(a) Agent-Based approach



(b) System Dynamics

Reflection

- Time
- Agents
- Feedback
- Stochasticity
- Deterministic

Where is the Environment?

Adding an Environment

- Running SFs *'parallel'*
⇒ multiple copies of environment
- State Monad elegant solution

Problem

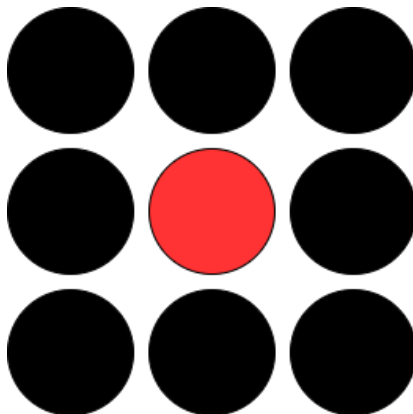
Yampa not monadic

Solution

Monadic Stream Functions (MSFs)

⇒ Signal Functions with monadic context

The environment: Moore neighbourhood



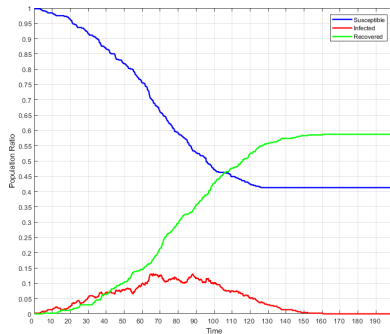
Re-defining Types

```

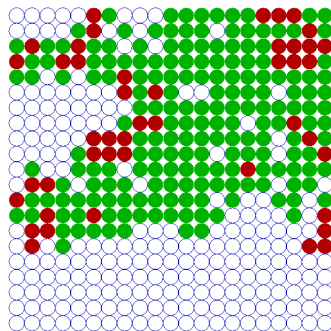
1  type Disc2dCoord = (Int, Int)
2  type SIREnv      = Array Disc2dCoord SIRState
3
4  type SIRMonad g = StateT SIREnv (Rand g)
5  type SIRAgent g = SF (SIRMonad g) () ()
6
7  neighbours :: SIREnv -> Disc2dCoord -> Disc2dCoord -> [Disc2dCoord] -> [SIRState]
8
9  moore :: [Disc2dCoord]
10 moore = [ topLeftDelta,    topDelta,    topRightDelta,
11           leftDelta,      rightDelta,
12           bottomLeftDelta, bottomDelta, bottomRightDelta ]
13
14 topLeftDelta :: Disc2dCoord
15 topLeftDelta = (-1, -1)
16
17 topDelta :: Disc2dCoord
18 topDelta = ( 0, -1)
19 ...

```


Dynamics with environment



(a) Dynamics over $t = 200$



(b) Visualisation at $t = 100$

Conclusion

- Purity guarantees reproducibility at compile time
- Performance
- Agent-Identity a bit lost
- Agent-Interaction is main difficulty

Future Work

Property-based testing for verification in ABS

Might offer huge benefits e.g. formalising hypotheses

Dependent Types in ABS (using Idris)

- Safe environment access, state-transitions, agent-interactions
- Equilibrium of Model & Totality of Implementation

Thank You!