Pure functional programming in Agent-Based Simulation

Jonathan Thaler

University of Nottingham, Ningbo, China

AIOP Seminar 2019

The Metaphor

- "[..] object-oriented programming is a particularly natural development environment for Sugarscape specifically and artificial societies generally [..]" (Epstein et al 1996)
- "agents map naturally to objects" (North et al 2007)

Outline

- What is Agent-Based Simulation (ABS)?
- What is *pure* Functional Programming (FP)?
- How can we do ABS + FP?
- ABS + FP = ?
- Conclusions

What is 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

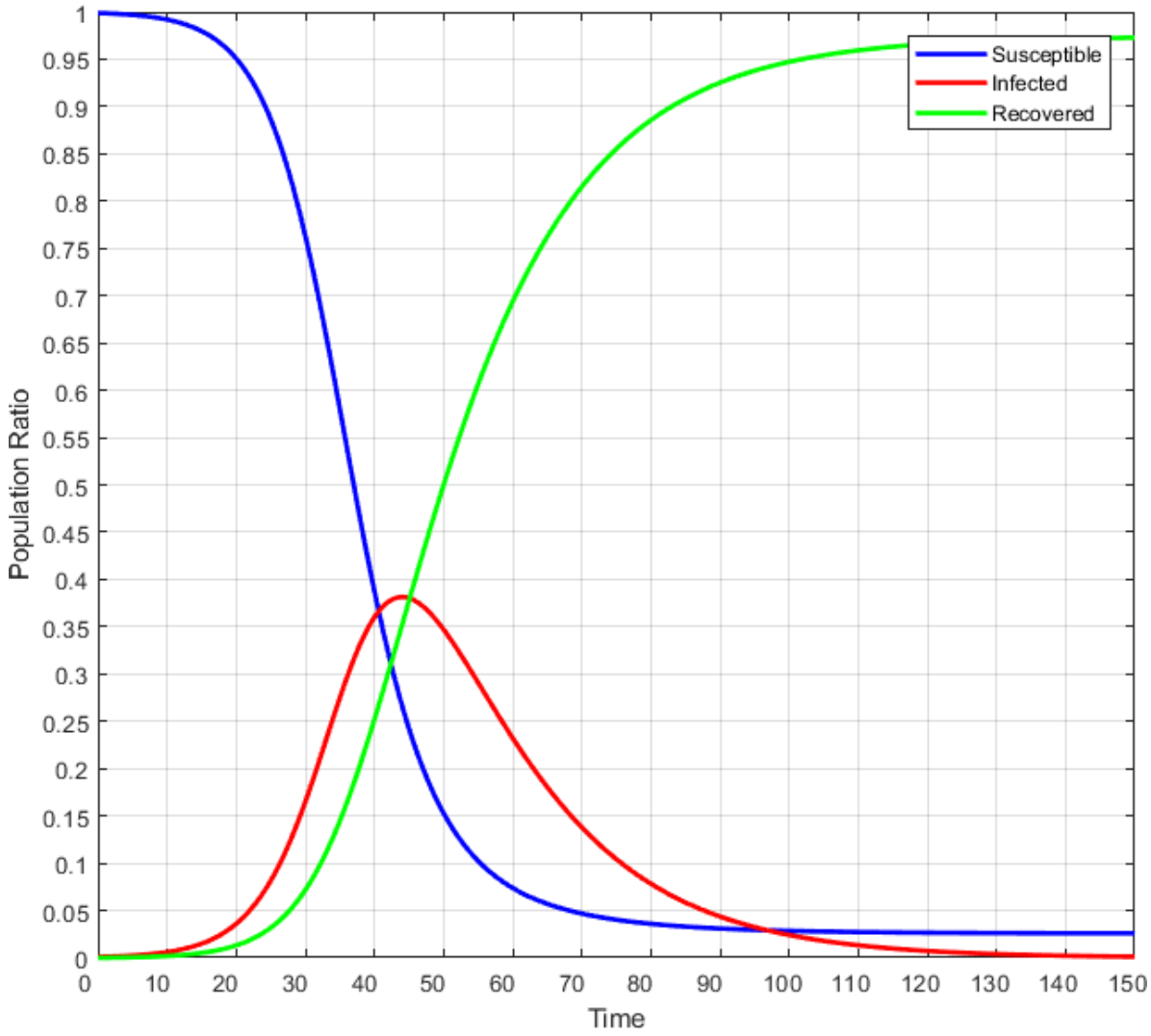
SIR Model

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

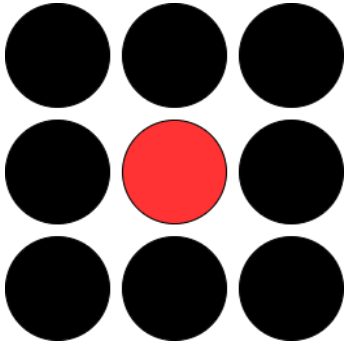
System Dynamics

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

SIR Model Dynamics

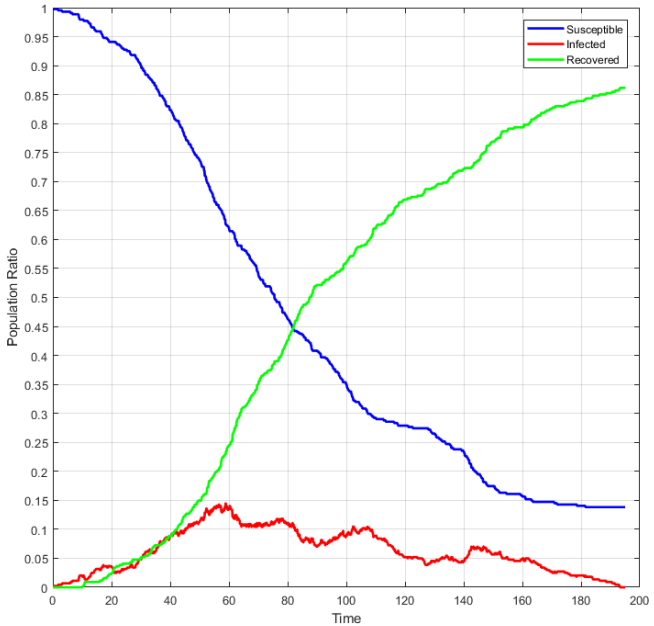


Defining Spatiality

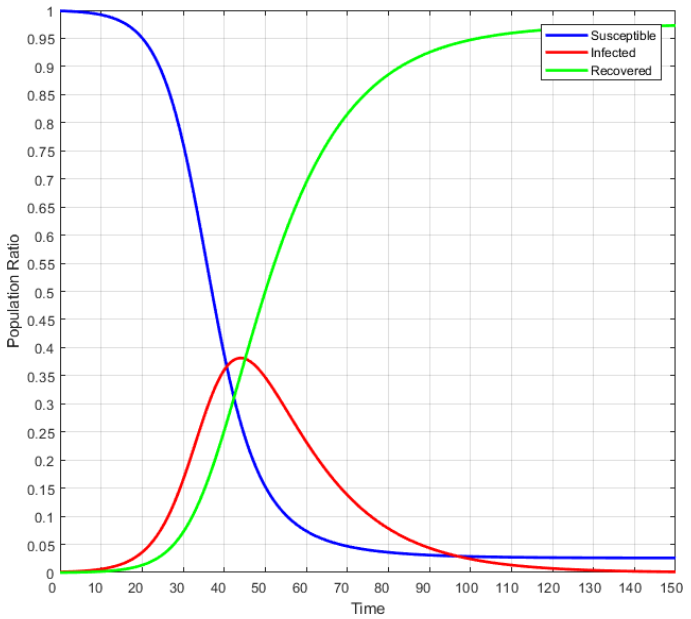


Moore Neighbourhood

Spatial Dynamics



Agent-Based



System Dynamics

Spatial Visualisation

What is pure functional programming?

Functions as first class citizens

Passed as arguments, returned as values and assigned to variables.

```
map :: (a -> b) -> [a] -> [b]

const :: a -> (b -> a)
const a = (\_ -> a)
```

What is pure functional programming cont'd?

Immutable data

Variables can not change, functions return new copy.
Data-Flow oriented programming.

```
let x   = [1..10]
    x'  = drop 5 x
    x'' = x' ++ [10..20]
```

What is pure functional programming cont'd?

Recursion

To iterate over and change data.

```
fact :: Int -> Int
fact 0 = 1
fact n = n * fact (n-1)
```

What is pure functional programming cont'd?

Declarative style

Describe *what* to compute instead of *how*.

```
mean :: [Double] -> Double
mean xs = sum xs / length xs
```

What is pure functional programming cont'd?

Explicit about Side-Effects

Distinguish between side-effects of a function *in its type*.

```
readFromFile :: String -> IO String
randomExponential :: Double -> Rand Double
statefulAlgorihm :: State Int (Maybe Double)
produceData :: Writer [Double] ()
```

How can we do ABS + FP?

How can we represent an Agent, its local state and its interface?

We don't have objects and mutable state...

How can we implement direct agent-to-agent interactions?

We don't have method calls and mutable state...

How can we implement an environment and agent-to-environment interactions?

We don't have method calls and mutable state...

Solution

Functional Reactive Programming with Monadic Stream Functions

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

Arrowized Functional Reactive Programming (AFRP)

```
type AgentId    = Int
data Message    = Tick Int | MatingRequest AgentGender ...
data AgentState = AgentState { agentAge :: Int, ... }
data Observable = Observable { agentAgeObs :: Int, ... }
data AgentOut   = AgentOut
  { kill       :: Bool
  , observable :: Observable
  , messages   :: [(AgentId, Message)] -- list of messages wit
  }
-- agent continuation has different types for input and output
newtype AgentCont inp out = AgentCont (inp -> (out, AgentCont
-- taking the initial AgentState as input and returns the cont
sugarscapeAgent :: AgentState -> AgentCont (AgentId, Message)
sugarscapeAgent asInit = AgentCont (\ (sender, msg) ->
  case msg of
    agentCont (sender, Tick t) = ... handle tick
    agentCont (sender, MatingRequest otherGender) = ... handle
```

ABS + FP = Type Saftey

Epstein et al (1996)

"... when the sequence of random numbers is specified ex ante the model is deterministic. Stated yet another way, model output is invariant from run to run when all aspects of the model are kept constant including the stream of random numbers."

- Purity guarantees reproducibility.
- Enforce and guarantee update semantics.

ABS + FP = Software Transactional Memory

- Concurrency using Software Transactional Memory (STM)
- Lock free!
- Tremendous performance improvement
- Substantially outperforms lock-based implementation
- STM semantics retain guarantees about non-determinism

ABS + FP = Software Transactional Memory cont'd

- With Haskell typesystem can be explicit in side-effects: STM only
- Guarantees that the non-determinism comes only from concurrency within STM and nothing else

ABS + FP = Property-Based Testing

- Express specifications directly in code and generate random test cases
- Stochastic nature of Property-Based TEsting and ABS should be perfect match

ABS + FP = Property-Based Testing cont'd

- Express specifications directly in code and generate random test cases
- Stochastic nature of Property-Based TEsting and ABS should be perfect match

Conclusion

- The direction is towards an simulation which is more likely to be correct with the ultimate goal being a correct-by-construction implementation (up to some specification).
- A correct-by-construction implementation does NOT relief us from actually running the simulation!
-

Thank You!