Show me your properties! The potential of property-based testing in Agent-Based Simulation

Jonathan Thaler

University of Nottingham, Nottingham, United Kingdom

SummerSim'19, July 22-24, Berlin, Germany

Testing in ABS?

- not existent, 1 paper focusing on it completely, very neglected but important.
- unit testing not very suitable for ABS in general?
- lacking

Stochastic ABS + Randomised Property-Based Testing = ♡♡♡

- Express specifications directly in code.
- QuickCheck library generates random test cases.
- Developer can express expected coverage.
- Part of the discovery and hypothesis process.

QuickCheck

List Properties

```
-- the reverse of a reversed list is the original list
reverse_reverse xs = reverse (reverse xs) == xs

-- concatenation operator (++) is associative
append_associative xs ys zs
  = (xs ++ ys) ++ zs == xs ++ (ys ++ zs)

-- reverse is distributive over concatenation (++)
reverse_distributive xs ys
  = reverse (xs ++ ys) == reverse xs ++ reverse ys
```

QuickCheck cont'd

Running the tests...

```
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
*** Failed! Falsifiable (after 3 tests and 1 shrink):
[1]
[0]
```

QuickCheck cont'd

Labeling

```
reverse_reverse_label xs
  = label ("length of list is " ++ show (length xs))
          (reverse (reverse xs) == xs)
```

Running the tests...

```
+++ OK, passed 100 tests:
 5% length of list is 27
 5% length of list is 0
 4% length of list is 19
 ...
```

QuickCheck cont'd

Coverage

```
reverse_reverse_cover xs  = checkCoverage
  cover 15 (length xs >= 50) "length of list at least 50"
  (reverse (reverse xs) == xs)
```

Running the tests...

```
+++ OK, passed 12800 tests
    (15.445% length of list at least 50).
```

Randomised Property-Based Testing

Matches the constructive and exploratory nature of ABS.

- Exploratory models: hypothesis tests about dynamics.
- Explanatory models: validate against formal specification.
- Test agent specification.
- Test simulation invariants.

Randomised Property-Based Testing

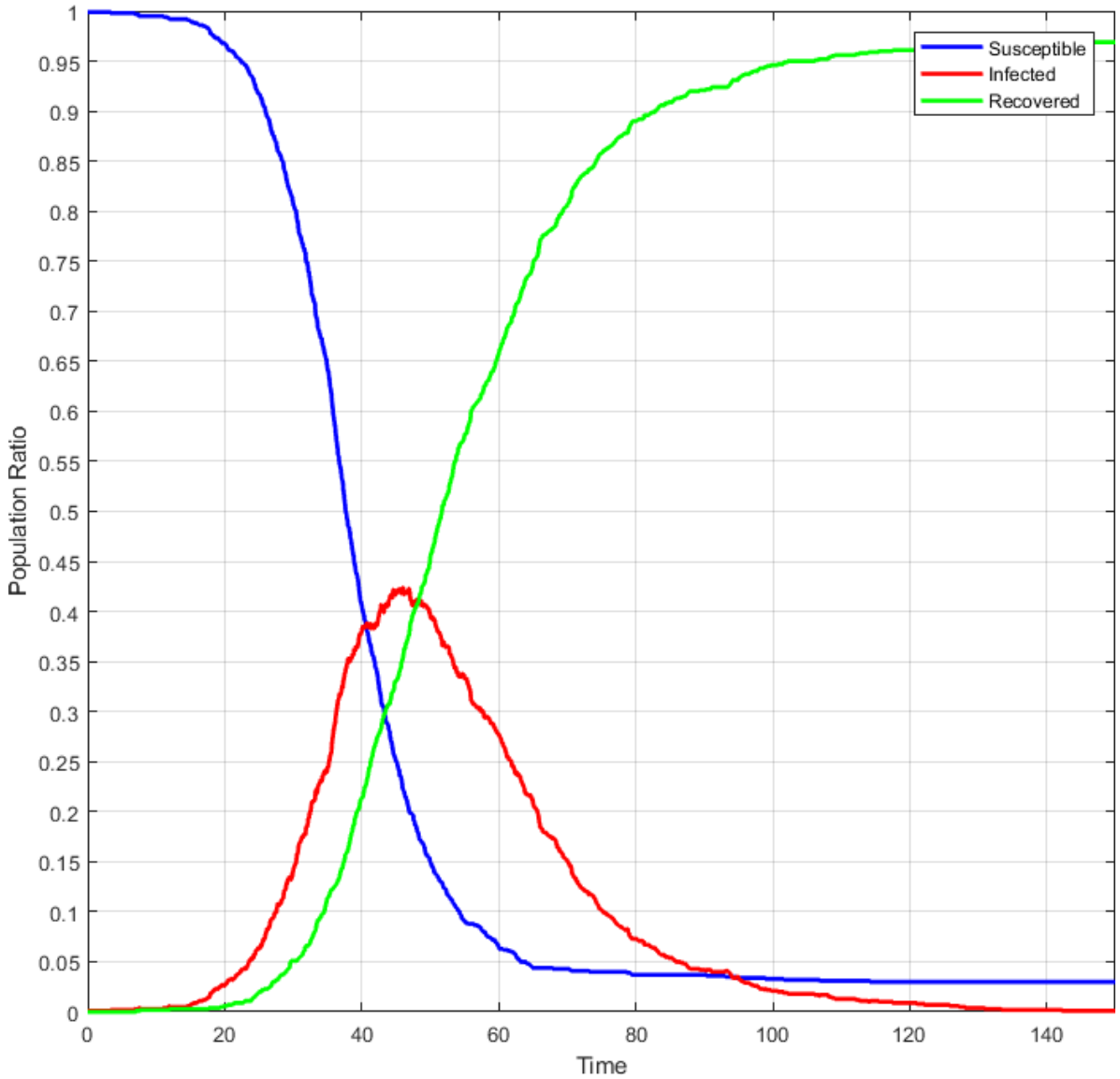
Matches the constructive and exploratory nature of ABS.

- Exploratory models: hypothesis tests about dynamics.
- Explanatory models: validate against formal specification.
- **Test agent specification.**
- **Test simulation invariants.**

Agent-Based SIR Model

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

Agent-Based SIR Dynamics



Test agent specification

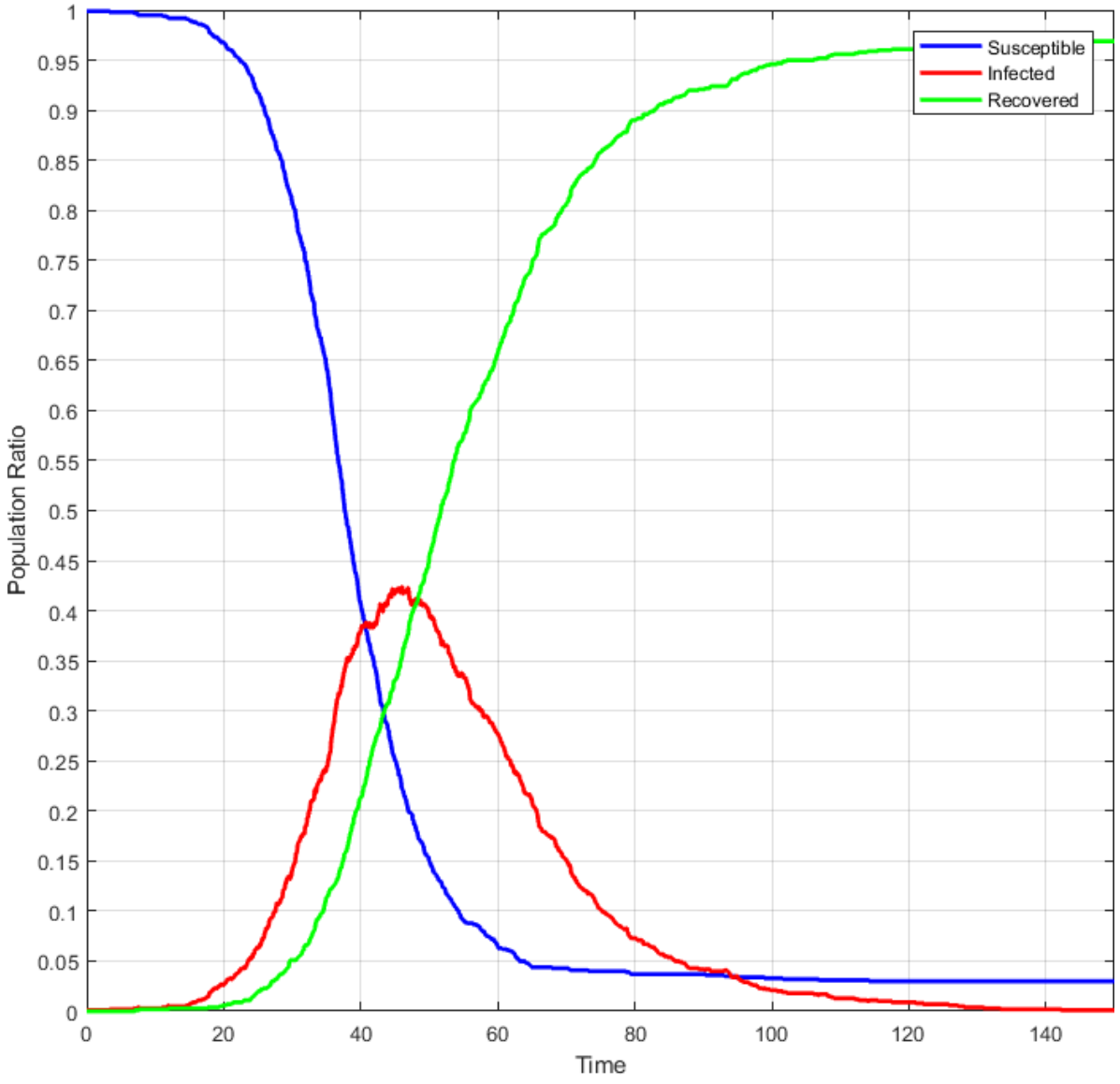
Code / Implementation Testing

Follow (formal) model specification / description.

Examples

- event-based: relate input to output events and perform random sampling
- time-based: encode invariants of output stream given random inputs
- use coverage to encode probabilities of e.g. transitions, timeouts,...

Agent-Based SIR Dynamics



Test simulation invariants

Any Models

Invariants have been established, fix them as test, used for regression testing.

Example: agent-based SIR

- Time is monotonic decreasing
- Number of susceptible agents *S* is monotonic decreasing.
- Number of recovered agents *R* is monotonic increasing.
- Number of agents *N* stays constant: N = S + I + R.
- Number of infected agents follows: I = N - (S + R).

Property has to hold under random model parameters for all test cases, therefore no statistical testing.

Example: SIR invariants

```
prop_sir_invariants :: Positive Int     -- ^ contact rate
                    -> Probability      -- ^ infectivity (0,1)
                    -> Positive Double  -- ^ illness duration
                    -> TimeRange        -- ^ duration
                    -> [SIRState]       -- ^ population
                    -> Property
prop_sir_invariants
    (Positive beta) (P gamma) (Positive delta) (T t) as
  = property (do
    -- total agent count
    let n = length as
    -- run the SIR simulation with a new RNG
    ret <- genSimulationSIR as beta gamma delta t
    -- check invariants and return result
    return (sirInvariants n ret)
```

Example: SIR invariants

```
sirInvariants :: Int                        -- ^ N total number of agents
              -> [(Time,(Int,Int,Int))] -- ^ output each step: (Time, (S,I,R))
              -> Bool
sirInvariants n aos = timeInc && aConst && susDec && recInc && infInv
  where
    (ts, sirs)  = unzip aos
    (ss, _, rs) = unzip3 sirs

    -- 1. time is monotonic increasing
    timeInc = allPairs (<=) ts
    -- 2. number of agents N stays constant in each step
    aConst = all agentCountInv sirs
    -- 3. number of susceptible S is monotonic decreasing
    susDec = allPairs (>=) ss
    -- 4. number of recovered R is monotonic increasing
    recInc = allPairs (<=) rs
    -- 5. number of infected I = N - (S + R)
    infInv = all infectedInv sirs

    agentCountInv :: (Int,Int,Int) -> Bool
    agentCountInv (s,i,r) = s + i + r == n

    infectedInv :: (Int,Int,Int) -> Bool
    infectedInv (s,i,r) = i == n - (s + r)

    allPairs :: (Ord a, Num a) => (a -> a -> Bool) -> [a] -> Bool
    allPairs f xs = all (uncurry f) (pairs xs)

    pairs :: [a] -> [(a,a)]
    pairs xs = zip xs (tail xs)
```

Conclusion

- drawback: time consuming, sufficient coverage?
- deterministic: SmallCheck, enumerate test cases up to some given depth

Thank You!