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

Code testing in ABS?

- Very neglected but important!
- One paper [1] focusing on TDD with unit testing
- Unit testing not very suitable for ABS
- How deal with ABS stochastic nature?

Solution

Stochastic ABS + random property-based testing = ♡♡♡

[1] Collier, N., and Ozik, J. Test-driven agent-based simulation development. In 2013 Winter Simulations Conference (WSC) (Dec. 2013),pp. 1551 - 1559.

Property-Based Testing

- Express specifications directly in code
- *QuickCheck* library generates random test cases
- Developer can express expected coverage
- Integrate into discovery and hypotheses 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).
```

Property-Based Testing in ABS

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 simulation and model invariants
- Test agent specification

Property-Based Testing in ABS

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 simulation and model invariants
- **Test agent specification**

Test agent specification

Code / Implementation Testing

Follow formal model specification or informal description

- Express invariants of output given random inputs
- Probabilities of transitions and timeouts use `cover`
- Event-driven ABS: relate input events to output events
- Time-driven ABS: specify output stream

Test agent specification

Code / Implementation Testing

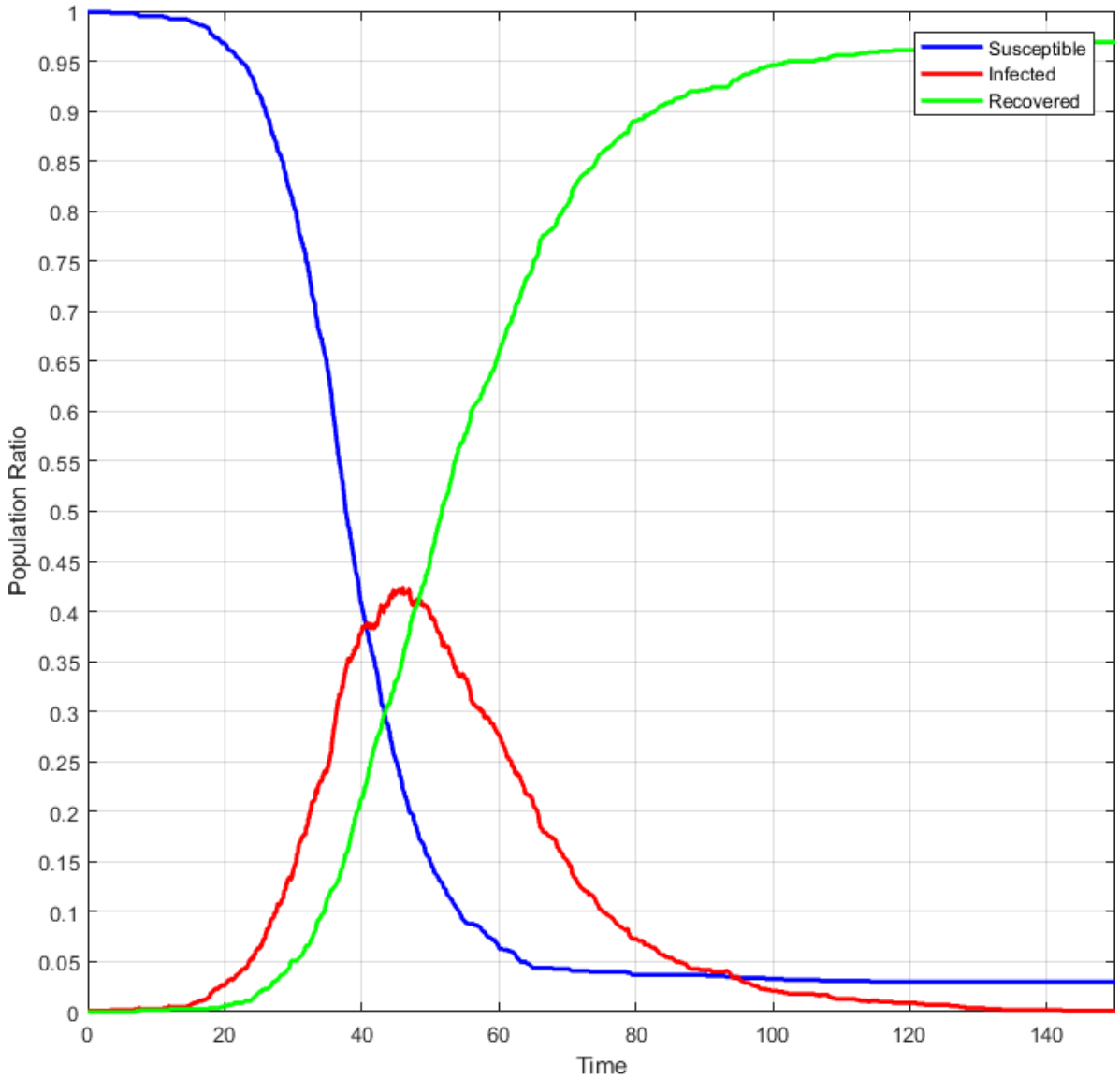
Follow formal model specification or informal description

- Express invariants of output given random inputs
- Probabilities of transitions and timeouts use `coverage`
- Event-driven ABS: relate input events to output events
- **Time-driven ABS: specify output stream**

Example: 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

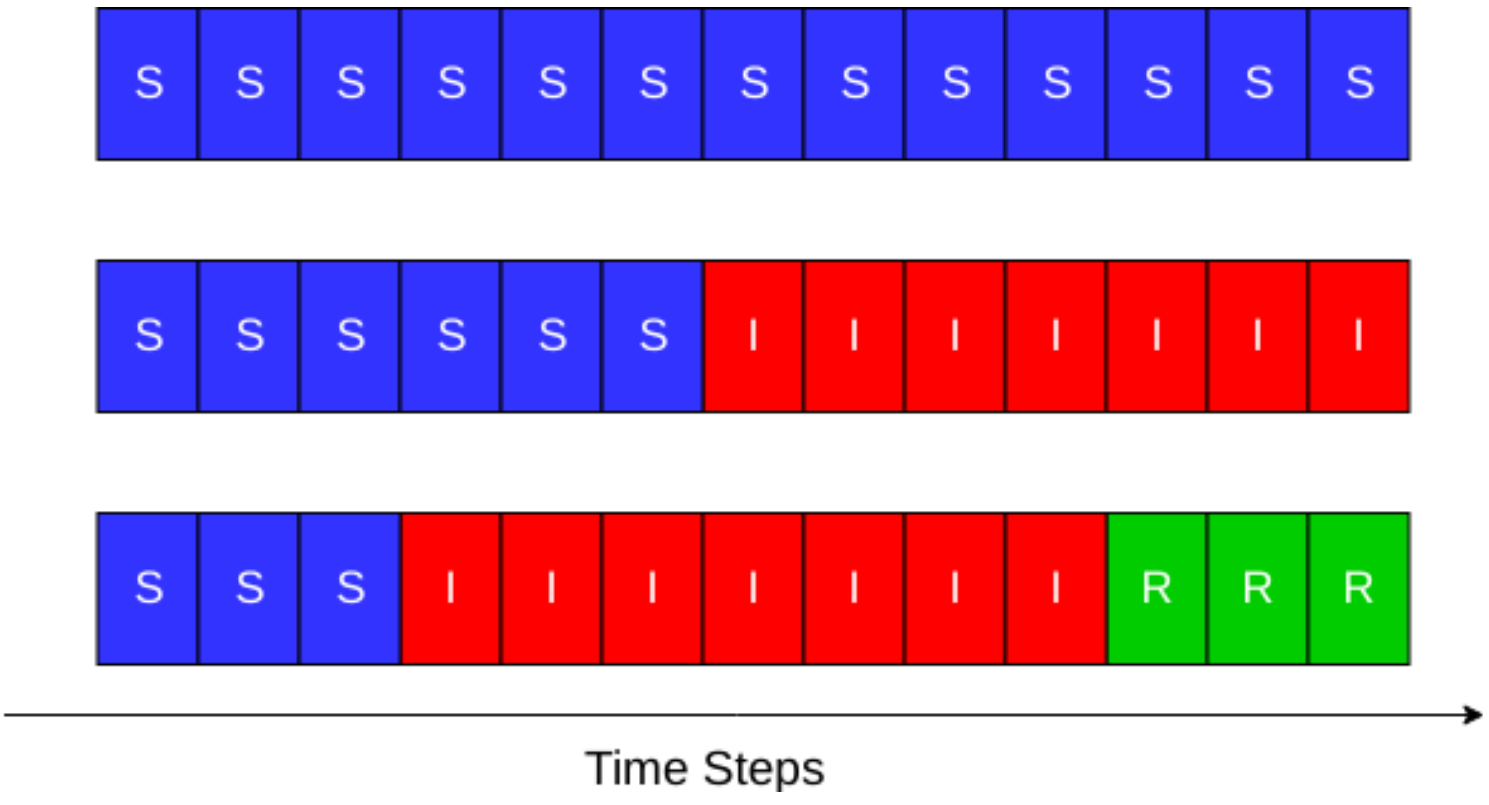
Dynamics of Agent-Based SIR Model



Susceptible Agent Specification

```
susceptibleAgent :: SF [SIRState] SIRState

data SIRState = Susceptible | Infected | Recovered
```



Susceptible invariants

```
susceptibleInv :: [SIRState] -- ^ output stream of the susceptible agent
               -> Bool       -- ^ population contains an infected agent
               -> Bool       -- ^ True in case the invariant holds
susceptibleInv aos infInPop
    -- Susceptible became Infected and then Recovered
    | isJust recIdxMay
      = infIdx < recIdx &&  -- agent has to become infected before recovering
        all (==Susceptible) (take infIdx aos) &&
        all (==Infected)    (take (recIdx - infIdx) (drop infIdx aos)) &&
        all (==Recovered)   (drop recIdx aos) &&
        infInPop  -- can only happen if there are infected in the population

    -- Susceptible became Infected
    | isJust infIdxMay
      = all (==Susceptible) (take infIdx aos) &&
        all (==Infected)    (drop infIdx aos) &&
        infInPop -- can only happen if there are infected in the population

    -- Susceptible stayed Susceptible
    | otherwise = all (==Susceptible) aos
  where
    infIdxMay = elemIndex Infected aos
    recIdxMay = elemIndex Recovered aos
    infIdx    = fromJust infIdxMay
    recIdx    = fromJust recIdxMay
```

Susceptible property test

```
prop_susceptible :: Positive Double -- ^ contact rate
                 -> Probability     -- ^ infectivity within (0,1)
                 -> Positive Double -- ^ illness duration
                 -> TimeRange       -- ^ simulation duration
                 -> [SIRState]      -- ^ population
                 -> Property
prop_susceptible
      (Positive beta) (P gamma) (Positive delta) (T t) as = property (do
    -- check if population contains an infected agent
    let infInPop = Infected `elem` as
    aos <- genSusceptible beta gamma delta as t
    return
        -- label all test cases
        label (labelTestCase aos)
        -- check invariants on output stream
        (property (susceptibleInv aos infInPop))
  where
    labelTestCase :: [SIRState] -> String
    labelTestCase aos
      | Recovered `elem` aos  = "Susceptible -> Infected -> Recovered"
      | Infected  `elem` aos  = "Susceptible -> Infected"
      | otherwise             = "Susceptible"
```

Checking the property

Running 10,000 test cases

```
> let args = stdArgs { maxSuccess = 10000 }
> quickCheckWith args prop_susceptible

> +++ OK, passed 10000 tests:
   55.78% Susceptible -> Infected -> Recovered
   37.19% Susceptible -> Infected
    7.03% Susceptible
```

Conclusion

- Property-Based Testing + ABS match naturally.
- Sufficient coverage is main difficulty.
- *SmallCheck* enumerates test cases deterministically.

Hopefully code testing will become more common in ABS.

Thank You!