Show me your properties!

Towards property-based testing in agent-based simulation

Jonathan Thaler

University of Nottingham, United Kingdom

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

Code testing in ABS?

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

A solution

Use of 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
 ...
```

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 testing

Follow formal model specification or informal description

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

Test Agent Specification

Code testing

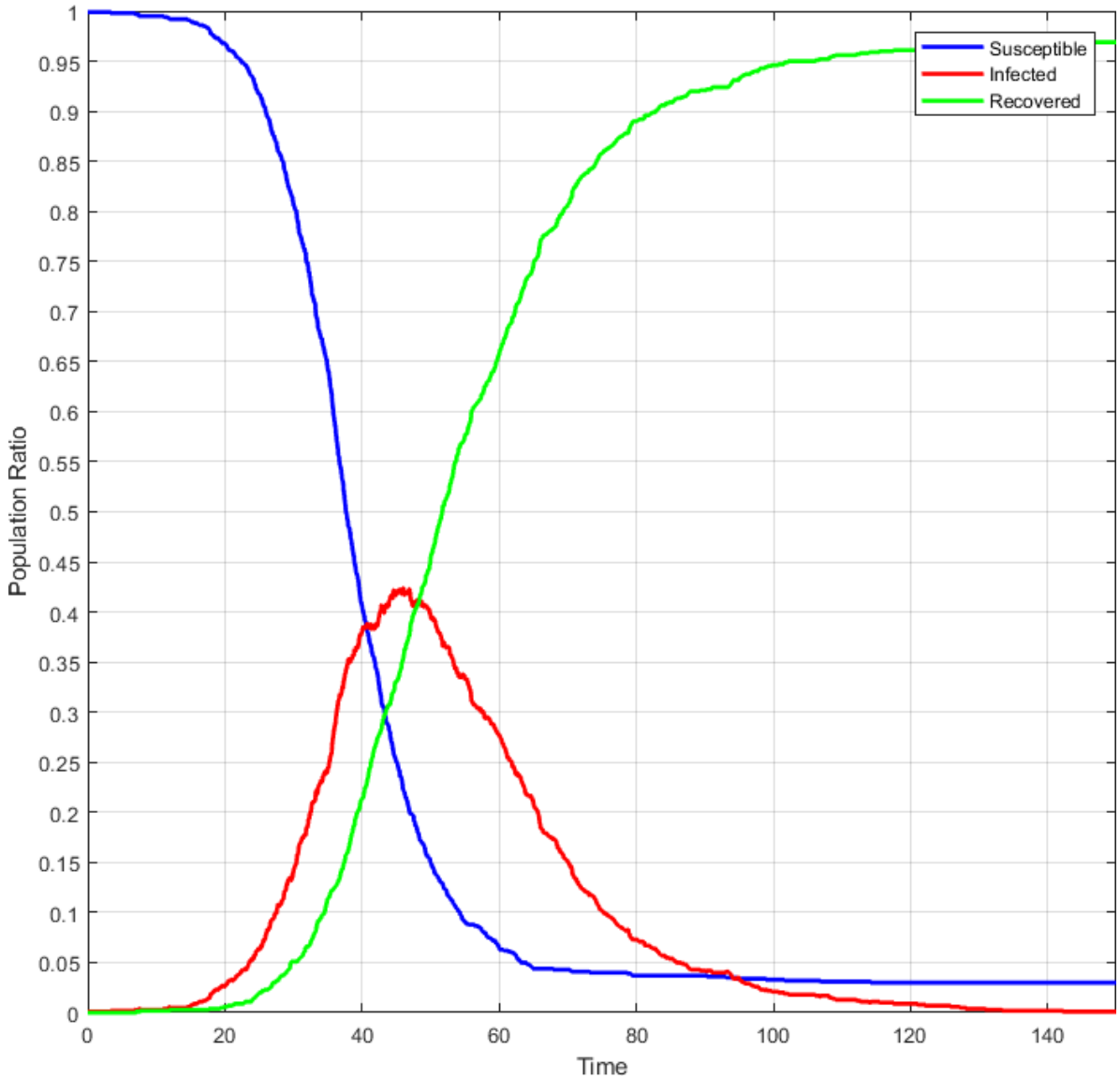
Follow formal model specification or informal description

- Express invariants of output given random inputs
- Probabilities of transitions and timeouts in *QuickCheck*
- 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

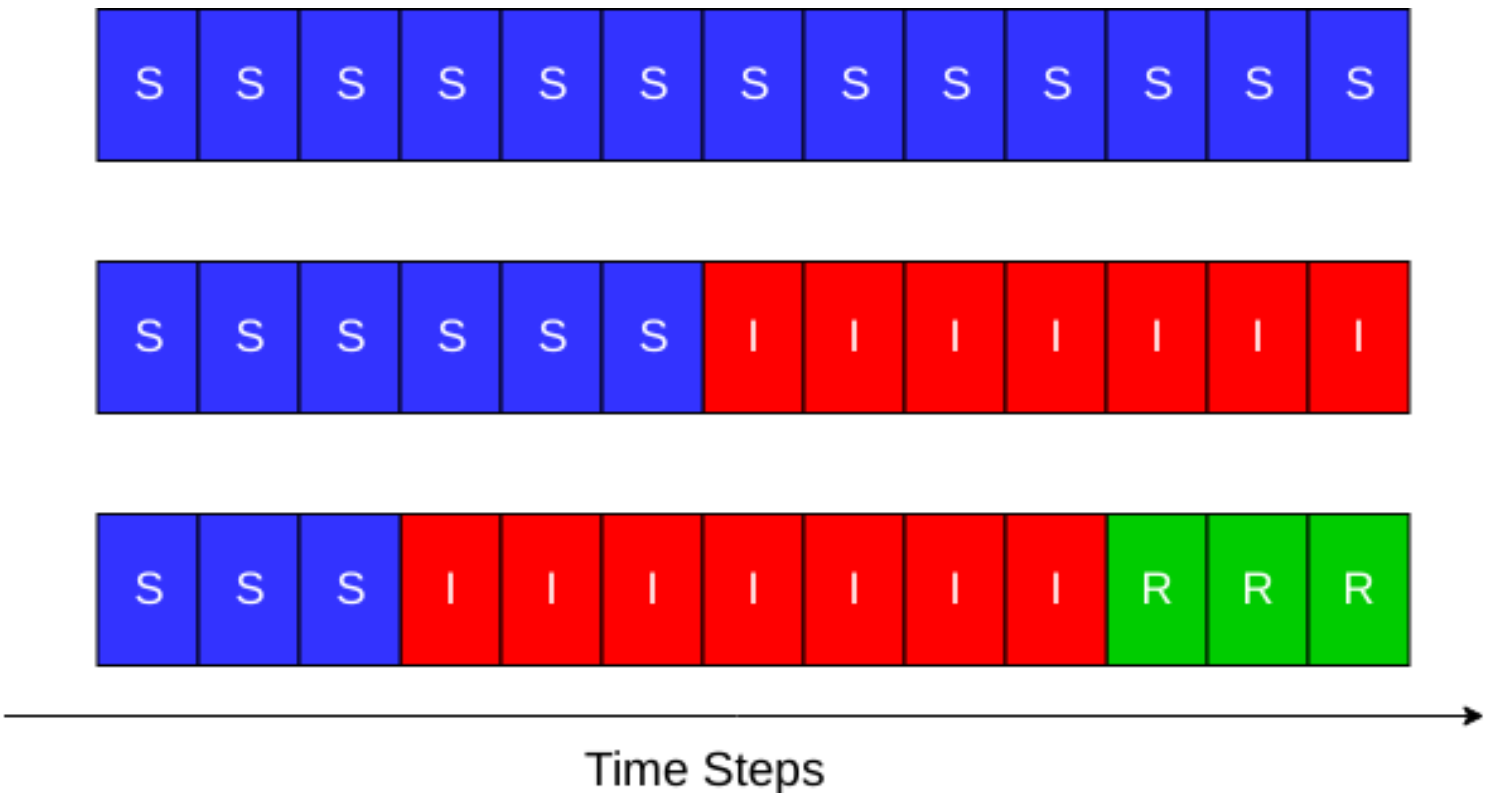
Dynamics of Agent-Based SIR Model



Susceptible Agent Specification

```
susceptibleAgent :: SF [SIRState] SIRState

data SIRState = Susceptible | Infected | Recovered
```



Susceptible Invariants

```
1   susceptibleInv :: [SIRState] -> Bool -> Bool
2   susceptibleInv aos infInPop
3       -- Susceptible -> Infected -> Recovered
4       | isJust recIdxMay
5         = infIdx < recIdx &&
6           all (==Susceptible) (take infIdx aos) &&
7           all (==Infected)    (take (recIdx - infIdx) (drop infIdx aos)) &&
8           all (==Recovered)   (drop recIdx aos) &&
9           infInPop
10
11      -- Susceptible -> Infected
12      | isJust infIdxMay
13        = all (==Susceptible) (take infIdx aos) &&
14          all (==Infected)    (drop infIdx aos) &&
15          infInPop
16
17      -- Susceptible
18      | otherwise = all (==Susceptible) aos
19    where
20      infIdxMay = elemIndex Infected aos
21      recIdxMay = elemIndex Recovered aos
22      infIdx    = fromJust infIdxMay
23      recIdx    = fromJust recIdxMay
```

Susceptible Property Test

```
1   prop_susceptible :: Positive Double -- ^ contact rate
2                    -> Probability     -- ^ infectivity within (0,1)
3                    -> Positive Double -- ^ illness duration
4                    -> TimeRange       -- ^ simulation duration
5                    -> [SIRState]      -- ^ population
6                    -> Property
7   prop_susceptible (Positive beta) (P gamma) (Positive delta) (T t) as = property (do
8       let infInPop = Infected `elem` as
9       aos <- genSusceptible beta gamma delta as t
10      return
11          label (labelTestCase aos)
12          (property (susceptibleInv aos infInPop))
13    where
14      labelTestCase :: [SIRState] -> String
15      labelTestCase aos
16        | Recovered `elem` aos  = "Susceptible -> Infected -> Recovered"
17        | Infected  `elem` aos  = "Susceptible -> Infected"
18        | otherwise             = "Susceptible"
```

Checking the Property

Running 10,000 test cases

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

> +++ OK, passed 10000 tests (12.86s):
   55.78% Susceptible -> Infected -> Recovered
   37.19% Susceptible -> Infected
    7.03% Susceptible
```

Conclusion

- Property-Based Testing + ABS match naturally
- Drawback: sufficient coverage
- Solution: *SmallCheck* enumerates test cases deterministically

Hopefully code testing will become more common in ABS

Thank You!