A Tool for Verification and Simulation of Population Protocols

Philip Offtermatt

3.8.2017

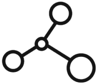
Overview

Population Protocols: models for distributed systems of mobile agents

Can solve many classical distributed tasks:

- Leader election
- Majority voting

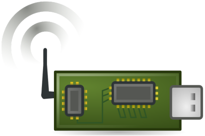
Creating new protocols is error-prone

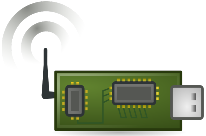


Contribution: A `Python` library for

- Specification
- Simulation
- Verification

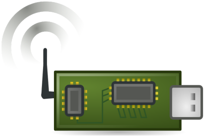
What are Population Protocols?



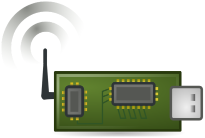


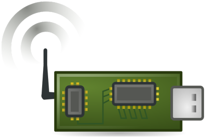
Population protocols:

- Models for distributed systems
- Agents have limited computational power
- Agents are passively mobile
- Agents are anonymous



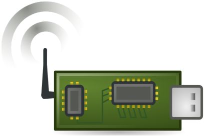
What are Population Protocols?





Population protocols:

- Models for distributed systems
- Agents have limited computational power
- Agents are passively mobile
- Agents are anonymous



What are Population Protocols?

Population protocols:

- Models for distributed systems
- Agents have limited computational power
- Agents are passively mobile
- Agents are anonymous

What are Population Protocols?

Population protocols:

- Models for distributed systems
- Agents have limited computational power
- Agents are passively mobile
- Agents are anonymous

What are Population Protocols?

- Agents are in one of finitely many states

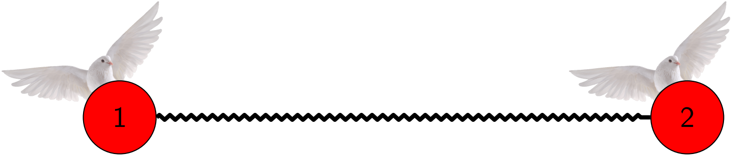
What are Population Protocols?

- Agents are in one of finitely many states
- Configuration: Multiset of states of agents

What are Population Protocols?

- Agents are in one of finitely many states
- Configuration: Multiset of states of agents
- When agents interact their states change

What are Population Protocols?



- Agents are in one of finitely many states
- Configuration: Multiset of states of agents
- When agents interact their states change

What are Population Protocols?

- Agents are in one of finitely many states
- Configuration: Multiset of states of agents
- When agents interact their states change

What are Population Protocols?

- Agents are in one of finitely many states
- Configuration: Multiset of states of agents
- When agents interact their states change
- Agents have output true/false

What are Population Protocols?

- Agents are in one of finitely many states
- Configuration: Multiset of states of agents
- When agents interact their states change
- Agents have output true/false

What are Population Protocols?

- Agents are in one of finitely many states
- Configuration: Multiset of states of agents
- When agents interact their states change
- Agents have output true/false
- Consensus: All agents have same output

Protocol $P = (Q, I, \delta, \omega)$

Q: a finite set of states

$I \subseteq Q$: the set of initial states

$\delta \subseteq Q^2 \times Q^2$: the transition relation

$\omega : Q \to \{0, 1\}$: the output function

Transition $t = a, b \to a', b'$ is enabled in C if $C = \{a, b, \dots\}$

$\Rightarrow$ Applying t to C results in $C' = \{a', b', \dots\}$

The Flock-Of-Birds Predicate

- Birds with normal or elevated temperature
- Given N, are there at least N birds with elevated temperature?
- Example for $N = 3$

The Flock-Of-Birds Predicate

- Birds with normal or elevated temperature
- Given N, are there at least N birds with elevated temperature?
- Example for $N = 3$

The Flock-Of-Birds Predicate

linear flock-of-birds protocol

N $= 3$

States: 0, 1, 2, 3

Transitions:

k , j $\rightarrow$ k+j , 0 if $k + j <$ N

k , j $\rightarrow$ N , N if $k + j \geq$ N

Initial states:

0, 1

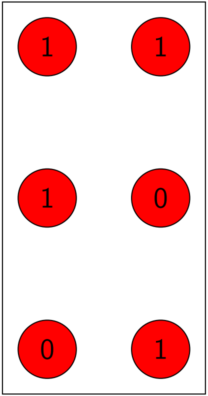
Output Function:

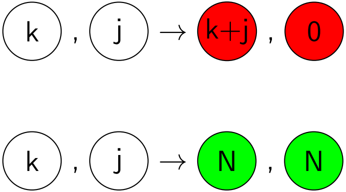
$\rightarrow 1$

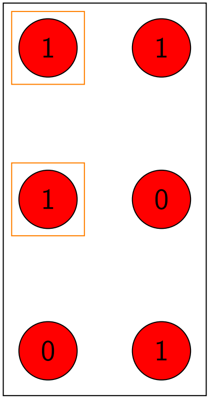
$\rightarrow 0$

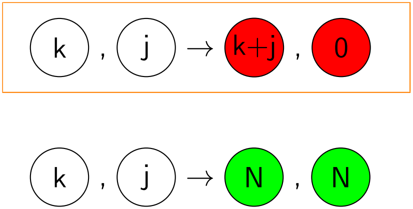
$$k \, , \, j \rightarrow k+j \, , \, 0$$

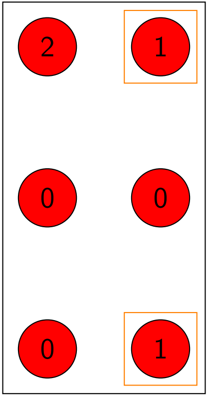
$$k \, , \, j \rightarrow N \, , \, N$$


1
1
1
0
0
1


k , j → k+j , 0
k , j → N , N

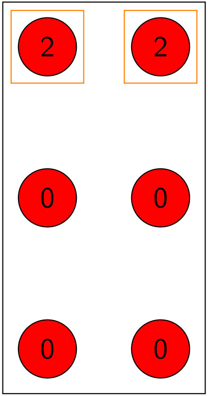

1 1
1 0
0 1


k , j → k+j , 0
k , j → N , N



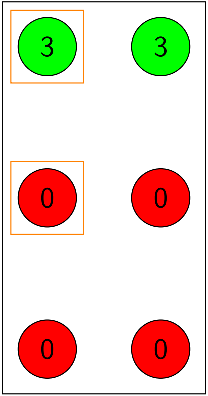
$$\boxed{k\ ,\ j \rightarrow k{+}j\ ,\ 0}$$

$$k\ ,\ j \rightarrow N\ ,\ N$$



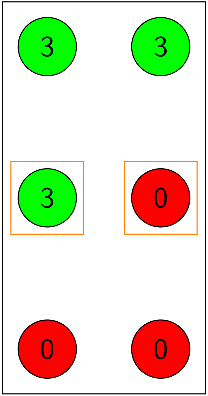
$$k \,,\, j \to k+j \,,\, 0$$

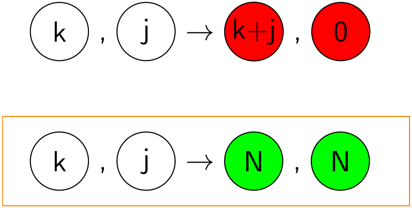
$$\boxed{k \,,\, j \to N \,,\, N}$$

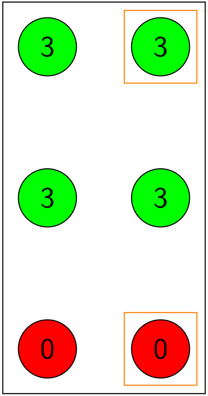


$$k \,,\, j \rightarrow k+j \,,\, 0$$

$$\boxed{k \,,\, j \rightarrow N \,,\, N}$$

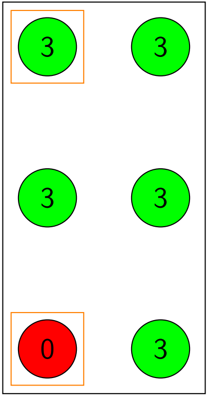

3
3
3
0
0
0


k , j → k+j , 0
k , j → N , N



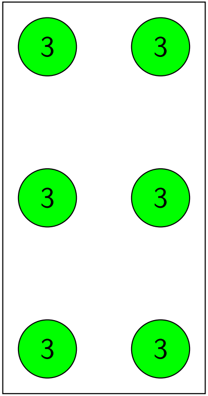
$$k \,,\, j \rightarrow k+j \,,\, 0$$

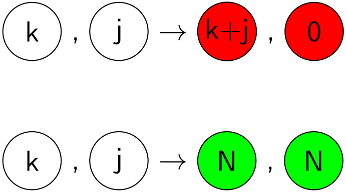
$$\boxed{k \,,\, j \rightarrow N \,,\, N}$$



$$k \, , \, j \to k+j \, , \, 0$$

$$\boxed{k \, , \, j \to N \, , \, N}$$


3
3
3
3
3
3


k , j → k+j , 0
k , j → N , N

Well Specification and Correctness

Execution: infinite sequence of subsequent configurations from some initial configuration C

Convergence: outputs of agents stabilize to consensus ("lasting consensus")

Fairness: execution is fair if all configurations that are reachable infinitely often appear infinitely often

Well Specification: all fair executions starting at same initial configuration converge to the same value
Fixed-Size Well Specification: well specified up to given size

Computing a predicate $f : I^{\mathbb{N}} \to \{0,1\}$: all fair executions from initial configuration C converge to $f(C)$

Well Specification and Correctness

Execution: infinite sequence of subsequent configurations from some initial configuration C

Convergence: outputs of agents stabilize to consensus ("lasting consensus")

Fairness: execution is fair if all configurations that are reachable infinitely often appear infinitely often

Well Specification: all fair executions starting at same initial configuration converge to the same value

Fixed-Size Well Specification: well specified up to given size

Computing a predicate $f : I^{\mathbb{N}} \to \{0, 1\}$: all fair executions from initial configuration C converge to $f(C)$

$\Rightarrow$ **Automatically verifying a protocol?**

Existing Tools

- bp-ver: fixed-size verification through graph exploration

  Chatzigiannakis, Michail and Spirakis SSS'10
- Verification with PRISM/SPIN

  Clment, Delporte-Gallet, Fauconnier and Sighireanu ICDCS'2011
- PAT: Model checker with global fairness

  Sun, Liu, Dong and Chen TASE'2009
- peregrine: parametric verification for a subclass of protocols

  Blondin, Esparza, Jaax, Meyer PODC'2017

A New Library For Population Protocols

Available under

`gitlab.lrz.de/ga96jib/tool_for_population_protocols`

Features:

- Specifying protocols
- Specifying configurations
- Simulating protocols
- Verifying protocols
- Exporting protocols to PRISM/peregrine

A New Library For Population Protocols

```
output_function = lambda x: x == N
transitions = [(k, j, k + j, 0) if k + j < N else (k, j, N, N) for k in range(N + 1) for j in range(N + 1)]
initial_states = {0, 1}
flock = Protocol(transitions, initial_states, output_function)

C = Population([0, 1, 1, 1, 1])

flock.average_convergence_steps(initial_population = C, num_iterations = 50)

flock.well_specified(size = 10, expected_output = lambda x: x.amount(1) >= N]

export.export_to_prism(flock, initial_population = C)
```

A New Library For Population Protocols

Available under
`gitlab.lrz.de/ga96jib/tool_for_population_protocols`

Features:

- Specifying protocols
- Specifying configurations
- Simulating protocols
- Verifying protocols
- Exporting protocols to PRISM/peregrine

$\Rightarrow$ **Tested on existing protocols**

$\Rightarrow$ **Devised a new protocol as a case study**

A New Protocol For Flock-Of-Birds

Problem: existing protocols for flock-of-birds need an amount of states linear in N.

Goal: find a protocol that needs less states

A New Protocol For Flock-Of-Birds

Problem: existing protocols for flock-of-birds need an amount of states linear in N.

Goal: find a protocol that needs less states
$\Rightarrow$ Prime-Flock-protocol

The Prime-Flock-Protocol

Example: $\mathsf{N} = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

Existing approach:

The Prime-Flock-Protocol

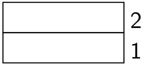
Example: $N = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

Existing approach:

The Prime-Flock-Protocol

Example: $N = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

Existing approach:



The Prime-Flock-Protocol

Example: $N = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

Existing approach:

The Prime-Flock-Protocol

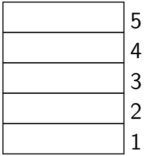
Example: $N = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

Existing approach:

The Prime-Flock-Protocol

Example: $\mathrm{N} = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

Existing approach:



The Prime-Flock-Protocol

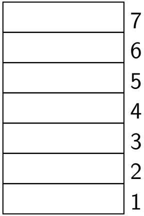
Example: $N = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

Existing approach:

The Prime-Flock-Protocol

Example: $N = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

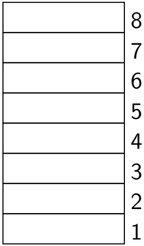
Existing approach:



The Prime-Flock-Protocol

Example: $\mathrm{N} = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

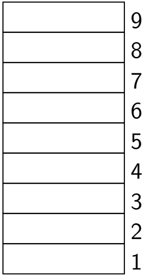
Existing approach:



The Prime-Flock-Protocol

Example: $N = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

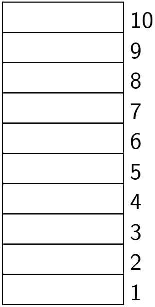
Existing approach:



The Prime-Flock-Protocol

Example: $N = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

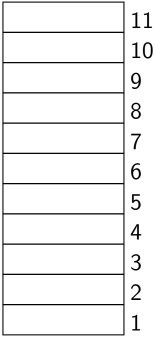
Existing approach:



The Prime-Flock-Protocol

Example: $N = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

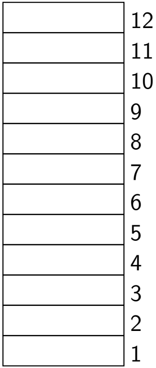
Existing approach:



The Prime-Flock-Protocol

Example: $N = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

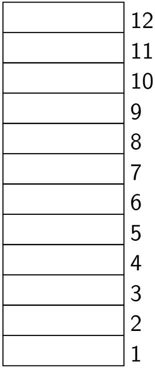
Existing approach:



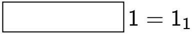
The Prime-Flock-Protocol

Example: $N = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

Existing approach:



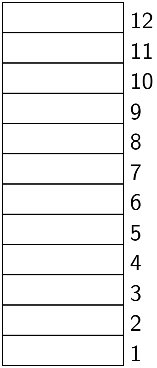
Prime-Flock-protocol:



The Prime-Flock-Protocol

Example: $N = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

Existing approach:

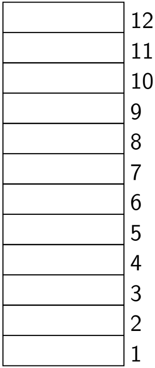


Prime-Flock-protocol:

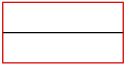
The Prime-Flock-Protocol

Example: $N = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

Existing approach:



Prime-Flock-protocol:

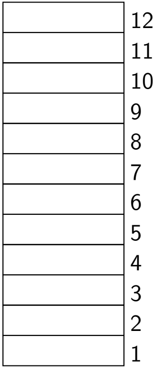


$1 \cdot 2 = 1 \cdot p_1 = 1_2$

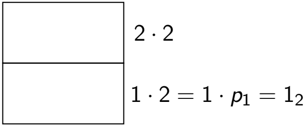
The Prime-Flock-Protocol

Example: $N = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

Existing approach:



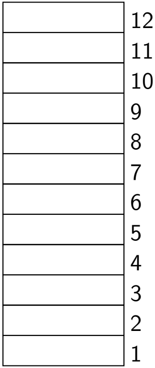
Prime-Flock-protocol:



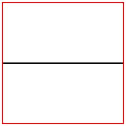
The Prime-Flock-Protocol

Example: $N = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

Existing approach:



Prime-Flock-protocol:

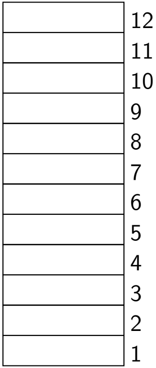


$1 \cdot 2 \cdot 2 = 1 \cdot p_2 \cdot p_1 = 1_3$

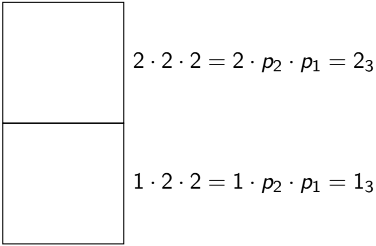
The Prime-Flock-Protocol

Example: $N = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

Existing approach:



Prime-Flock-protocol:



The Prime-Flock-Protocol

Example: $N = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

Existing approach:

12
11
10
9
8
7
6
5
4
3
2
1

Prime-Flock-protocol:

$3 \cdot 2 \cdot 2$

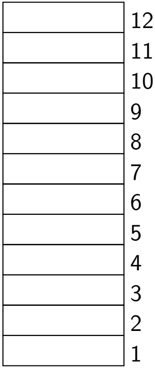
$2 \cdot 2 \cdot 2 = 2 \cdot p_2 \cdot p_1 = 2_3$

$1 \cdot 2 \cdot 2 = 1 \cdot p_2 \cdot p_1 = 1_3$

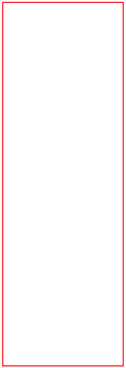
The Prime-Flock-Protocol

Example: $N = 12 = \underbrace{2}_{p_1} \cdot \underbrace{2}_{p_2} \cdot \underbrace{3}_{p_3}$

Existing approach:



Prime-Flock-protocol:



$12 = 1 \cdot p_3 \cdot p_2 \cdot p_1 = N$

The Prime-Flock-Protocol

$$\mathsf{N} = p_1 \cdot p_2 \cdots p_n$$

States:

$1_1, 2_1, \ldots, (p_1 - 1)_1$
$1_2, 2_2, \ldots, (p_2 - 1)_2$
$\vdots$
$1_n, 2_n, \ldots, (p_n - 1)_n$
N

Transitions:

$$\begin{aligned}
i_k, j_k &\rightarrow (i+j)_k, 0 && i+j < p_k \\
i_k, j_k &\rightarrow 1_{k+1}, ((i+j) - p_k)_k && i+j \geq p_k \\
\mathsf{N}, x &\rightarrow \mathsf{N}, \mathsf{N} && \forall x \in Q
\end{aligned}$$

The Prime-Flock-Protocol

$$N = p_1 \cdot p_2 \cdots p_n$$

States:

$1_1, 2_1, \ldots, (p_1 - 1)_1$
$1_2, 2_2, \ldots, (p_2 - 1)_2$
$\vdots$
$1_n, 2_n, \ldots, (p_n - 1)_n$
N

Transitions:

$$\begin{aligned}
i_k, j_k &\to (i+j)_k, 0 && i + j < p_k \\
i_k, j_k &\to 1_{k+1}, ((i+j) - p_k)_k && i + j \geq p_k \\
\mathrm{N}, x &\to \mathrm{N}, \mathrm{N} && \forall x \in Q
\end{aligned}$$

$|Q|$ in $O(\sum_{i=1}^{n} p_i)$

The Prime-Flock-Protocol

$\mathrm{N} = p_1 \cdot p_2 \cdots p_n$

States:

$1_1, 2_1, \ldots, (p_1 - 1)_1$
$1_2, 2_2, \ldots, (p_2 - 1)_2$
$\vdots$
$1_n, 2_n, \ldots, (p_n - 1)_n$
N

Transitions:

$$
\begin{aligned}
i_k, j_k &\to (i+j)_k, 0 && i + j < p_k \\
i_k, j_k &\to 1_{k+1}, ((i+j) - p_k)_k && i + j \geq p_k \\
\mathrm{N}, x &\to \mathrm{N}, \mathrm{N} && \forall x \in Q
\end{aligned}
$$

$|Q|$ in $O(\sum_{i=1}^{n} p_i)$

Proof $\Rightarrow$ Thesis

Probability Distributions

How do we choose the next transition in each step?

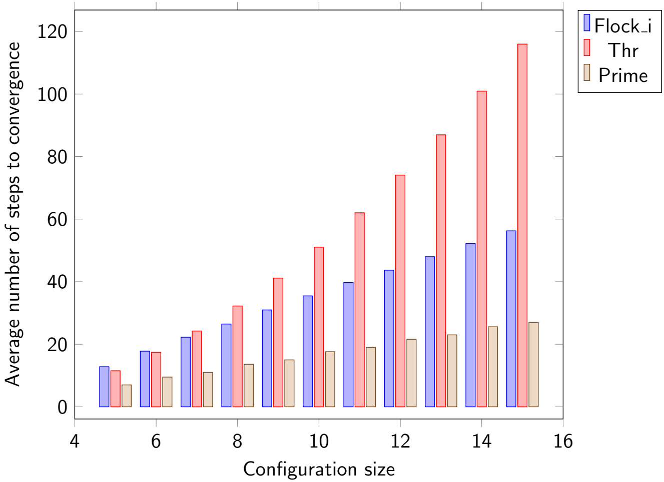
Uniform rules scheduling: Choose uniformly at random among enabled transitions

Uniform pairs scheduling: Choose two agents uniformly at random from configuration, then choose uniformly at random among transitions for the states of these agents

$\Rightarrow$ Compare convergence behaviour of protocols for both

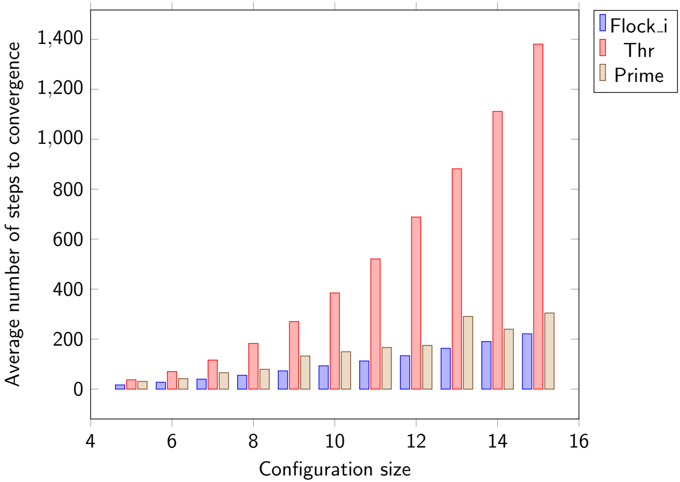
Comparing Protocols For Flock-Of-Birds

Uniform rules scheduling



Comparing Protocols For Flock-Of-Birds

Uniform pairs scheduling



Summary

- New `Python` library for specifying, simulating and verifying population protocols
- New protocol computing the flock-of-birds predicate that uses less states than existing protocols

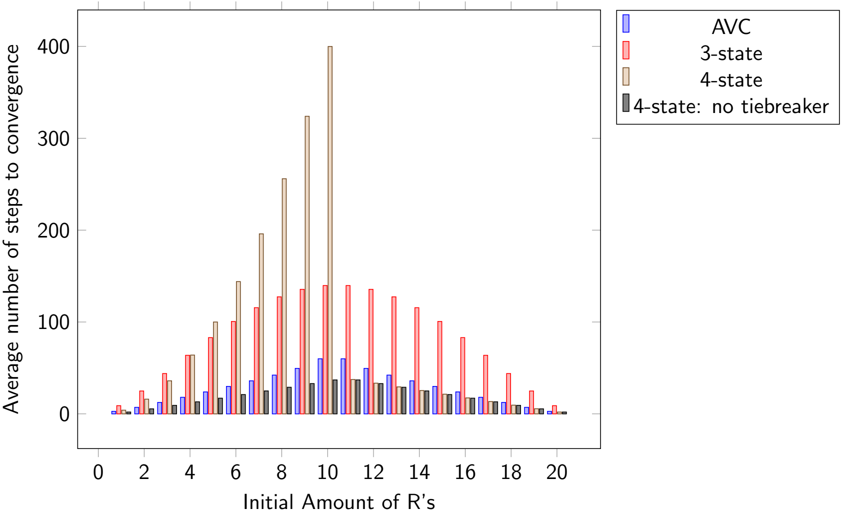
Outlook - future work

- multiple transitions in one step
- export to more model checkers
- optimizing verification
- flock-of-birds: lower bound for states?

Thank you!

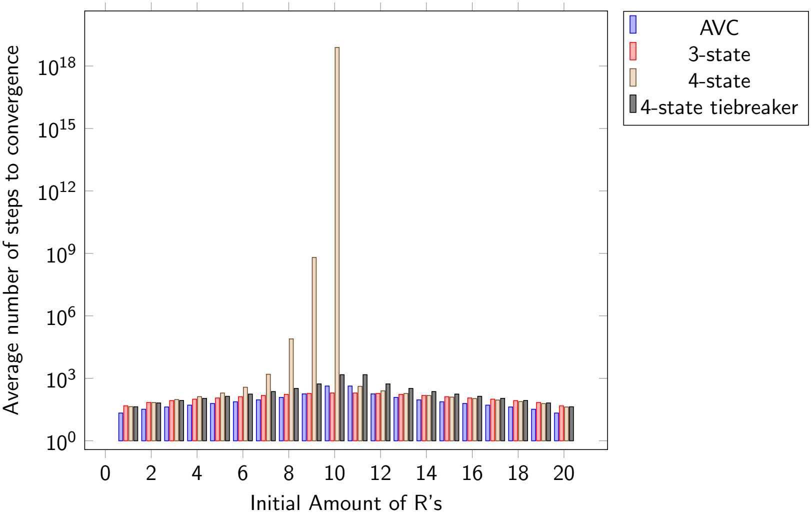
Comparing Protocols For The Majority Predicate

Uniform rules scheduling



Comparing Protocols For The Majority Predicate

Uniform pairs scheduling



Exporting Protocols to Prism

```
R : [0..4] init 2;
B : [0..4] init 2;
"0" : [0..4] init 0;
"1" : [0..4] init 0;
```

Exporting Protocols to Prism

Arbitrary Probability Distributions

```
[] R = 2 & B = 2 & "0" = 0 & "1" = 0 ->
1.0: (R'=R-1) & (B'=B-1) & ("0"'="0"+1) & ("1"'="1"+1);
[] R = 1 & B = 1 & "0" = 1 & "1" = 1 ->
0.25: (R'=R-1) & (B'=B-1) & ("0"'="0"+1) & ("1"'="1"+1) +
0.5: ("1"'="1"-1) & ("0"'="0"+1) +
0.25: ("0"'="0"-1) & ("1"'="1"+1)
[] R = 0 & B = 0 & "0" = 2 & "1" = 2 ->
1.0: ("0"'="0"+1) & ("1"'="1"-1);
```

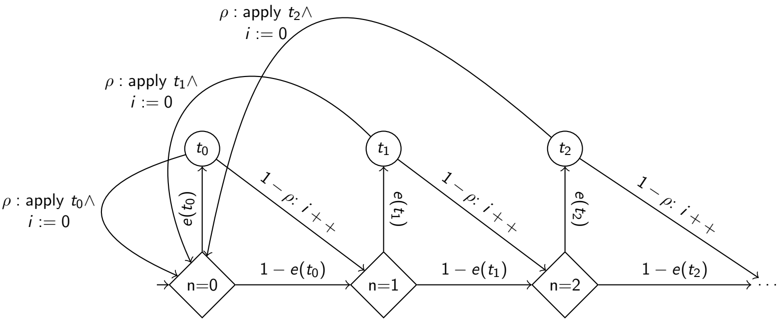
Exporting Protocols to Prism

Nondeterministic Endocing for Uniform Distributions

```
[] R >= 1 & B >= 1 & "0" <= 3 & "1" <= 3 ->
    (R'=R-1) & (B'=B-1) & ("0"'="0"+1) & ("1"'="1"+1);
[] R >= 1 & "1" >= 1 & "0" <= 3 ->
    ("1"'="1"-1) & ("0"'="0"+1);
[] B >= 1 & "0" >= 1 & "1" <= 3 ->
    ("0"'="0"-1) & ("1"'="1"+1);
[] "0" >= 1 & "1" >= 1 & "0" <= 3 ->
    ("0"'="0"+1) & ("1"'="1"-1);
```

Exporting Protocols to Prism

Deterministic Endocing for Uniform Distributions



$e(t_x) = 1$ if t_x is enabled, else 0

$\rho = (i+1)/T$, where $T = \sum_x e(t_x)$