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
Population Protocols
Definitions

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

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

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

$\Rightarrow$ Applying $t$ to $C$ results in $C' = \{a', b', \ldots\}$
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:
- \( \text{green} \rightarrow 1 \)
- \( \text{red} \rightarrow 0 \)
\( k \), \( j \) → \( k + j \), 0

\( k \), \( j \) → \( N \), \( N \)
[Diagram with red and green circles showing the following relations:]

- \( k, j \rightarrow k + j, 0 \) (red)
- \( k, j \rightarrow N, N \) (green)
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
(\text{k}, \text{j}) & \rightarrow (\text{k} + \text{j}, 0) \\
(\text{k}, \text{j}) & \rightarrow (\text{N}, \text{N})
\end{align*}
\]
<table>
<thead>
<tr>
<th>k</th>
<th>j</th>
<th>→</th>
<th>k+j</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>→</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
\[
(3, 3) \rightarrow (k+j, 0),
(3, 0) \rightarrow N, N
\]
\( k, j \rightarrow k+j, 0 \)

\( k, j \rightarrow N, N \)
$k, j \rightarrow k+j, 0$

$k, j \rightarrow N, N$
\[ (k, j) \rightarrow (k+j, 0) \]

\[ (k, j) \rightarrow (N, N) \]
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 $f : I^N \rightarrow \{0, 1\}$**: 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** $f : \mathbb{I}^\mathbb{N} \rightarrow \{0, 1\}$: 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
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, 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

⇒ Tested on existing protocols
⇒ 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

⇒ Prime-Flock-protocol
The Prime-Flock-Protocol

Example: \( N = 12 = \frac{2}{p_1} \cdot \frac{2}{p_2} \cdot \frac{3}{p_3} \)

Existing approach:
The Prime-Flock-Protocol

Example: \( N = 12 = 2^2 \cdot 2^1 \cdot 3^1 \)

Existing approach:
The Prime-Flock-Protocol

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

Existing approach:
The Prime-Flock-Protocol

Example: \( N = 12 = 2 \cdot 2 \cdot 3 \)

Existing approach:
The Prime-Flock-Protocol

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

Existing approach:
The Prime-Flock-Protocol

Example: \( N = 12 = 2 \cdot 2 \cdot 3 \)

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: \( N = 12 = \underbrace{2 \cdot 2 \cdot 3}_{p_1 \cdot p_2 \cdot p_3} \)

Existing approach:
The Prime-Flock-Protocol

Example: \( N = 12 = 2^{p_1} \cdot 2^{p_2} \cdot 3^{p_3} \)

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:** \( N = 12 = p_1 \cdot p_2 \cdot p_3 \)

Existing approach:

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Prime-Flock-Protocol

Example: \( N = 12 = 2 \cdot 2 \cdot 3 \)

Existing approach:
The Prime-Flock-Protocol

Example: \( N = 12 = 2^{p_1} \cdot 2^{p_2} \cdot 3^{p_3} \)

Existing approach:

Prime-Flock-protocol:

\[
\begin{array}{c}
\hline
& 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\
\hline
& 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\hline
\end{array}
\]

\( 1 = 1_1 \)
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:

\[ \begin{array}{c}
12 \\
11 \\
10 \\
9 \\
8 \\
7 \\
6 \\
5 \\
4 \\
3 \\
2 \\
1 \\
\end{array} \]

\[ \begin{array}{c}
2 \\
1 = 1_1 \\
\end{array} \]
The Prime-Flock-Protocol

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

Existing approach: 

<table>
<thead>
<tr>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
</table>

Prime-Flock-protocol:

\[
1 \cdot 2 = 1 \cdot p_1 = 1_2
\]
The Prime-Flock-Protocol

Example: \( N = 12 = 2 \cdot 2 \cdot 3 \)

Existing approach:

Prime-Flock-protocol:
The Prime-Flock-Protocol

Example: \( N = 12 = 2 \cdot 2 \cdot 3 \)

Existing approach:

Prime-Flock-protocol:

1. \( 1 \cdot 2 = 1 \cdot p_2 \cdot p_1 = 1_3 \)
The Prime-Flock-Protocol

Example: \( N = 12 = 2 \cdot 2 \cdot 3 \)

Existing approach:

Prime-Flock-protocol:

\[
\begin{array}{c}
12 \\
11 \\
10 \\
9 \\
8 \\
7 \\
6 \\
5 \\
4 \\
3 \\
2 \\
1
\end{array}
\]

\[
\begin{array}{c}
12 \\
11 \\
10 \\
9 \\
8 \\
7 \\
6 \\
5 \\
4 \\
3 \\
2 \\
1
\end{array}
\]

\[
2 \cdot 2 \cdot 2 = 2 \cdot p_2 \cdot p_1 = 2_3
\]

\[
1 \cdot 2 \cdot 2 = 1 \cdot p_2 \cdot p_1 = 1_3
\]
The Prime-Flock-Protocol

Example: \( N = 12 = 2 \cdot 2 \cdot 3 \)

Existing approach:

Prime-Flock-protocol:
The Prime-Flock-Protocol

Example: \( N = 12 = \frac{2}{p_1} \cdot \frac{2}{p_2} \cdot \frac{3}{p_3} \)

Existing approach:

Prime-Flock-protocol:

\[ 12 = 1 \cdot p_3 \cdot p_2 \cdot p_1 = N \]
The Prime-Flock-Protocol

\[ N = p_1 \cdot p_2 \cdots p_n \]

**States:**
1. 1, 2, \ldots, (p_1 - 1)
2. 1, 2, \ldots, (p_2 - 1)
3. \vdots
4. 1, 2, \ldots, (p_n - 1)

**Transitions:**
\[ i_k, j_k \rightarrow (i + j)_k, 0 \quad i + j < p_k \]
\[ i_k, j_k \rightarrow 1_{k+1}, ((i + j) - p_k)_k \quad i + j \geq p_k \]
\[ N, x \rightarrow N, N \quad \forall x \in Q \]
The Prime-Flock-Protocol

\[ N = p_1 \cdot p_2 \cdots p_n \]

States:
1, 2, \ldots, (p_1 - 1)
1_2, 2_2, \ldots, (p_2 - 1)
\vdots
1_n, 2_n, \ldots, (p_n - 1)
N

Transitions:
\[ i_k, j_k \rightarrow (i + j)_k, 0 \quad i + j < p_k \]
\[ i_k, j_k \rightarrow 1_{k+1}, ((i + j) - p_k)_k \quad i + j \geq p_k \]
N, x \rightarrow N, N
\[ \forall x \in Q \]

|Q|\ in \ \mathcal{O}(\sum_{i=1}^{n} p_i)
The Prime-Flock-Protocol

\[ N = p_1 \cdot p_2 \cdots p_n \]

**States:**

1. \(1_1, 2_1, \ldots, (p_1 - 1)_1\)
2. \(1_2, 2_2, \ldots, (p_2 - 1)_2\)
3. \(
   \vdots
\)
4. \(1_n, 2_n, \ldots, (p_n - 1)_n\)

**Transitions:**

\[ i_k, j_k \rightarrow (i + j)_k, 0 \quad \text{if} \quad i + j < p_k \]
\[ i_k, j_k \rightarrow 1_{k+1}, ((i + j) - p_k)_k \quad \text{if} \quad i + j \geq p_k \]
\[ N, x \rightarrow N, N \quad \forall x \in Q \]

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

**Proof \Rightarrow Thesis**
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

⇒ Compare convergence behaviour of protocols for both
Comparing Protocols For Flock-Of-Birds

Uniform rules scheduling

![Graph showing the average number of steps to convergence for different configuration sizes. The x-axis represents configuration size, ranging from 4 to 16. The y-axis represents the average number of steps to convergence, ranging from 0 to 120. The graph compares three protocols: Flock_i, Thr, and Prime. Each protocol is represented by a different color: blue for Flock_i, red for Thr, and brown for Prime. The graph shows a generally increasing trend in the average number of steps to convergence as the configuration size increases.]
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

![Graph showing the average number of steps to convergence for different protocols.

- **AVC**
- **3-state**
- **4-state**
- **4-state: no tiebreaker**

Initial Amount of R's vs. Average number of steps to convergence.
Comparing Protocols For The Majority Predicate

Uniform pairs scheduling

![Graph showing the average number of steps to convergence for different protocols. The x-axis represents the initial amount of R's, ranging from 0 to 20. The y-axis represents the average number of steps to convergence, ranging from $10^0$ to $10^{18}$. The graph includes bars for AVC, 3-state, 4-state, and 4-state tiebreaker protocols.]
Exporting Protocols to Prism

\[
\begin{align*}
R & : [0..4] \text{ init } 2; \\
B & : [0..4] \text{ init } 2; \\
"0" & : [0..4] \text{ init } 0; \\
"1" & : [0..4] \text{ init } 0;
\end{align*}
\]
Exporting Protocols to Prism
Arbitrary Probability Distributions

\[
\begin{align*}
[] & R = 2 & B = 2 & "0" = 0 & "1" = 0 \rightarrow \\
& 1.0: (R' = R-1) & (B' = B-1) & ("0"' = "0" + 1) & ("1"' = "1" + 1); \\
[] & R = 1 & B = 1 & "0" = 1 & "1" = 1 \rightarrow \\
& 0.25: (R' = R-1) & (B' = B-1) & ("0"' = "0" + 1) & ("1"' = "1" + 1) + \\
& 0.5: ("1"' = "1" - 1) & ("0"' = "0" + 1) + \\
& 0.25: ("0"' = "0" - 1) & ("1"' = "1" + 1) \\
[] & R = 0 & B = 0 & "0" = 2 & "1" = 2 \rightarrow \\
& 1.0: ("0"' = "0" + 1) & ("1"' = "1" - 1);
\end{align*}
\]
Exporting Protocols to Prism

Nondeterministic Encoding for Uniform Distributions

\[
\begin{align*}
[\] & R \geq 1 & B \geq 1 & "0" \leq 3 & "1" \leq 3 \implies \\
& \quad (R' = R - 1) & (B' = B - 1) & ("0"' = "0" + 1) & ("1"' = "1" + 1); \\
[\] & R \geq 1 & "1" \geq 1 & "0" \leq 3 \implies \\
& \quad ("1"' = "1" - 1) & ("0"' = "0" + 1); \\
[\] & B \geq 1 & "0" \geq 1 & "1" \leq 3 \implies \\
& \quad ("0"' = "0" - 1) & ("1"' = "1" + 1); \\
[\] & "0" \geq 1 & "1" \geq 1 & "0" \leq 3 \implies \\
& \quad ("0"' = "0" + 1) & ("1"' = "1" - 1);
\end{align*}
\]
Deterministic Encoding for Uniform Distributions

\[
e(t_x) = 1 \text{ if } t_x \text{ is enabled, else } 0
\]

\[
\rho = \frac{(i + 1)}{T}, \text{ where } T = \sum_x e(t_x)
\]