Verification of Population Protocols

Javier Esparza
Technical University of Munich

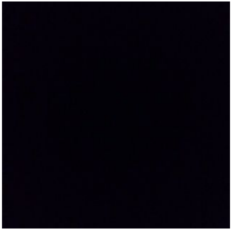
Joint work with Pierre Ganty, Jérôme Leroux,
and Rupak Majumdar

Deaf Black Ninjas in the Dark

- Deaf Black Ninjas meet at a Zen garden in the dark

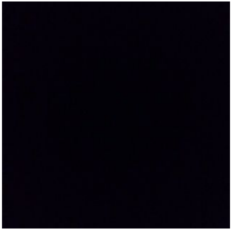
Deaf Black Ninjas in the Dark

- Deaf Black Ninjas meet at a Zen garden in the dark



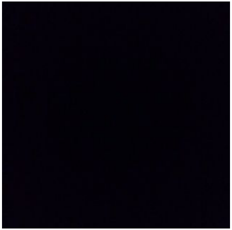
Deaf Black Ninjas in the Dark

- Deaf Black Ninjas meet at a Zen garden in the dark
- All Ninjas are indistinguishible, and don't know how many they are



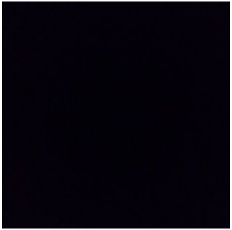
Deaf Black Ninjas in the Dark

- Deaf Black Ninjas meet at a Zen garden in the dark
- All Ninjas are indistinguishible, and don't know how many they are
- Ninjas must decide by majority to attack or not ("don't attack" if tie)



Deaf Black Ninjas in the Dark

- Deaf Black Ninjas meet at a Zen garden in the dark
- All Ninjas are indistinguishible, and don't know how many they are
- Ninjas must decide by majority to attack or not ("don't attack" if tie)
- How can they conduct the vote?



Deaf Black Ninjas in the Dark

Ninjas randomly wander around the garden, interacting when they bump into each other

Deaf Black Ninjas in the Dark

Ninjas randomly wander around the garden, interacting when they bump into each other

Each Ninja stores his current "guess" of the outcome (Yes / No). Additionally, it is either Active or Passive. (Four possible states)

Deaf Black Ninjas in the Dark

Ninjas randomly wander around the garden, interacting when they bump into each other

Each Ninja stores his current "guess" of the outcome (Yes / No). Additionally, it is either Active or Passive. (Four possible states)

Initially: all Ninjas Active, their guesses are their own votes

Deaf Black Ninjas in the Dark

Ninjas randomly wander around the garden, interacting when they bump into each other

Each Ninja stores his current "guess" of the outcome (Yes / No). Additionally, it is either Active or Passive. (Four possible states)

Initially: all Ninjas Active, their guesses are their own votes

Ninjas follow this protocol:

$$\begin{array}{rcl} (YA, NA) & \mapsto & (NP, NP) \\ (YA, NP) & \mapsto & (YA, YP) \\ (NA, YP) & \mapsto & (NA, NP) \\ (NP, YP) & \mapsto & (NP, NP) \end{array}$$

Deaf Black Ninjas in the Dark

Ninjas randomly wander around the garden, interacting when they bump into each other

Each Ninja stores his current "guess" of the outcome (Yes / No). Additionally, it is either Active or Passive. (Four possible states)

Initially: all Ninjas Active, their guesses are their own votes

Ninjas follow this protocol:

$$\begin{aligned}
(YA, NA) &\mapsto (NP, NP) \\
(YA, NP) &\mapsto (YA, YP) \\
(NA, YP) &\mapsto (NA, NP) \\
(NP, YP) &\mapsto (NP, NP)
\end{aligned}$$

Random bumps guarantee eventual consensus

Population protocols (PP)

Theoretical model for distributed computation

Proposed in 2004 by Angluin *et al.*

Population protocols (PP)

Theoretical model for distributed computation

Proposed in 2004 by Angluin *et al.*

Designed to model collections of

identical, finite-state, and mobile agents

like

Population protocols (PP)

Theoretical model for distributed computation

Proposed in 2004 by Angluin *et al.*

Designed to model collections of

identical, finite-state, and mobile agents

like

- ad-hoc networks of mobile sensors

Population protocols (PP)

Theoretical model for distributed computation

Proposed in 2004 by Angluin *et al.*

Designed to model collections of

identical, finite-state, and mobile agents

like

- ad-hoc networks of mobile sensors
- "soups" of interacting molecules

Population protocols (PP)

Theoretical model for distributed computation

Proposed in 2004 by Angluin *et al.*

Designed to model collections of

identical, finite-state, and mobile agents

like

- ad-hoc networks of mobile sensors
- "soups" of interacting molecules
- people in social networks

Population protocols (PP)

Theoretical model for distributed computation

Proposed in 2004 by Angluin *et al.*

Designed to model collections of

identical, finite-state, and mobile agents

like

- ad-hoc networks of mobile sensors
- "soups" of interacting molecules
- people in social networks
- ... and ninjas

Syntax

A PP-scheme is a pair (Q, Δ), where

- Q is a finite set of states, and
- $\Delta \subseteq (Q \times Q) \times (Q \times Q)$ is a set of interactions.

Syntax

A PP-scheme is a pair (Q, Δ), where

- Q is a finite set of states, and
- $\Delta \subseteq (Q \times Q) \times (Q \times Q)$ is a set of interactions.

Intuition:

if	$(q_1, q_2) \mapsto (q_1', q_2') \in \Delta$ **and**
	two agents in states q_1 and q_2 "meet",
then	the agents can interact and
	change their states to q_1', q_2'.

Assumption: at least one interaction for each (q_1, q_2)

Semantics

Configuration: mapping $C\colon Q \to \mathbb{N}$, where $C(q)$ is the current number of agents in state q.

q_1	q_2	q_3	q_4
2	1	0	3

Semantics

Configuration: mapping $C\colon Q \to \mathbb{N}$, where $C(q)$ is the current number of agents in state q.

q_1	q_2	q_3	q_4
2	1	0	3

$$(q_1, q_2) \mapsto (q_3, q_4)$$

Semantics

Configuration: mapping $C \colon Q \to \mathbb{N}$, where $C(q)$ is the current number of agents in state q.

$$\begin{array}{cccccccccc} q_1 & q_2 & q_3 & q_4 & & q_1 & q_2 & q_3 & q_4 \\ 2 & 1 & 0 & 3 & \longrightarrow & 1 & 0 & 1 & 4 \end{array}$$

$$(q_1, q_2) \mapsto (q_3, q_4)$$

Semantics

Configuration: mapping $C\colon Q \to \mathbb{N}$, where $C(q)$ is the current number of agents in state q.

$$\begin{array}{cccc} q_1 & q_2 & q_3 & q_4 \\ \textcircled{2} & \textcircled{1} & \textcircled{0} & \textcircled{3} \end{array} \longrightarrow \begin{array}{cccc} q_1 & q_2 & q_3 & q_4 \\ \textcircled{1} & \textcircled{0} & \textcircled{1} & \textcircled{4} \end{array}$$

$$(q_1, q_2) \mapsto (q_3, q_4)$$

If several steps are possible, a scheduler chooses one

Semantics

Configuration: mapping $C\colon Q \to \mathbb{N}$, where $C(q)$ is the current number of agents in state q.

$$\begin{array}{cccccccccc} q_1 & q_2 & q_3 & q_4 & & q_1 & q_2 & q_3 & q_4 \\ 2 & 1 & 0 & 3 & \longrightarrow & 1 & 0 & 1 & 4 \end{array}$$

$$(q_1, q_2) \mapsto (q_3, q_4)$$

If several steps are possible, a scheduler chooses one

Execution: infinite sequence $C_0 \to C_1 \to C_2 \to \cdots$ of steps

Semantics

Configuration: mapping $C\colon Q \to \mathbb{N}$, where $C(q)$ is the current number of agents in state q.

$$\begin{array}{cccc} q_1 & q_2 & q_3 & q_4 \\ 2 & 1 & 0 & 3 \end{array} \longrightarrow \begin{array}{cccc} q_1 & q_2 & q_3 & q_4 \\ 1 & 0 & 1 & 4 \end{array}$$

$$(q_1, q_2) \mapsto (q_3, q_4)$$

If several steps are possible, a scheduler chooses one

Execution: infinite sequence $C_0 \to C_1 \to C_2 \to \cdots$ of steps

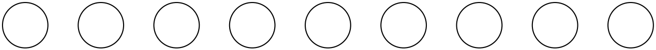
Fair Execution: **if** C appears infinitely often and $C \to C'$
then C' appears infinitely often.

(Fairness constraint approximating random scheduler)

Population protocols (PPs)

A population protocol (PP) consists of

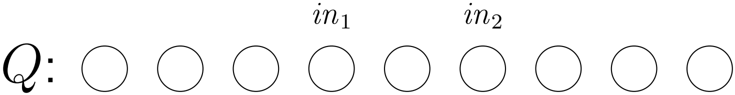
- A PP-scheme (Q, Δ)

Q: 

Population protocols (PPs)

A population protocol (PP) consists of

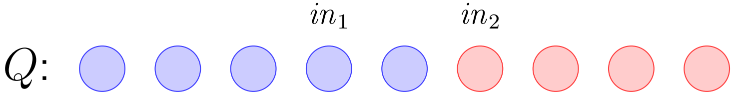
- A PP-scheme (Q, Δ)
- A tuple $(in_1, \dots, in_k)$ of input states



Population protocols (PPs)

A population protocol (PP) consists of

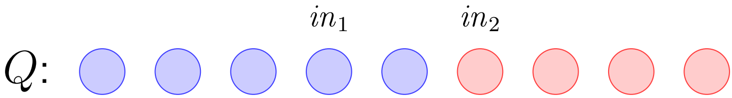
- A PP-scheme (Q, Δ)
- A tuple $(in_1, \ldots, in_k)$ of input states
- A partition of Q into true-states and false-states



Population protocols (PPs)

A population protocol (PP) consists of

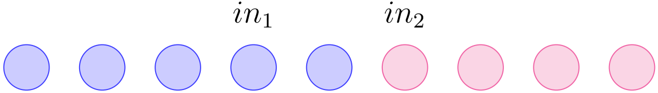
- A PP-scheme (Q, Δ)
- A tuple $(in_1, \ldots, in_k)$ of input states
- A partition of Q into true-states and false-states



A fair execution stabilizes to $b \in \{\mathrm{true}, \mathrm{false}\}$ if from some point on every agent stays within the b-states. ("All agents agree on b").

Computing with PPs

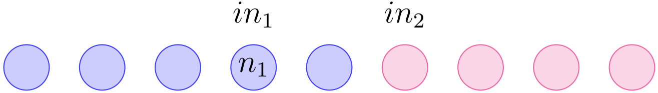
A PP computes the value b for input $(n_1, \ldots, n_k)$ if every fair execution starting at the configuration



Computing with PPs

A PP computes the value b for input $(n_1, \ldots, n_k)$ if every fair execution starting at the configuration

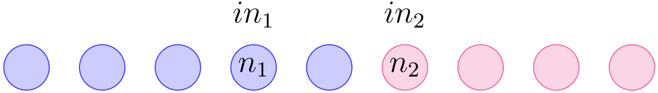
$$n_1 \cdot \mathbf{in_1}$$



Computing with PPs

A PP computes the value b for input $(n_1, \ldots, n_k)$ if every fair execution starting at the configuration

$$n_1 \cdot \mathbf{in_1} + \cdots + n_k \cdot \mathbf{in_k}$$

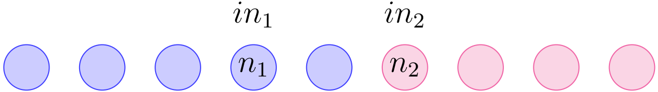


Computing with PPs

A PP computes the value b for input $(n_1, \ldots, n_k)$ if every fair execution starting at the configuration

$$n_1 \cdot \mathbf{in_1} + \cdots + n_k \cdot \mathbf{in_k}$$

stabilizes to b.

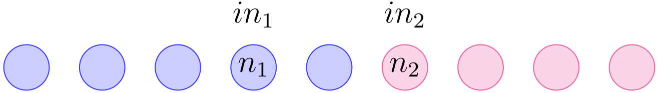


Computing with PPs

A PP computes the value b for input $(n_1, \ldots, n_k)$ if every fair execution starting at the configuration

$$n_1 \cdot \mathbf{in_1} + \cdots + n_k \cdot \mathbf{in_k}$$

stabilizes to b.



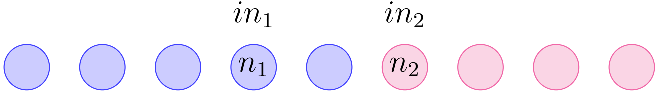
Intuitively: all agents agree on b whatever the (random) scheduler

Computing with PPs

A PP computes the value b for input $(n_1, \ldots, n_k)$ if every fair execution starting at the configuration

$$n_1 \cdot \mathbf{in_1} + \cdots + n_k \cdot \mathbf{in_k}$$

stabilizes to b.



Intuitively: all agents agree on b whatever the (random) scheduler

A PP computes $P \colon \mathbb{N}^n \to \{\mathrm{true}, \mathrm{false}\}$ if it computes $P(n_1, \ldots, n_k)$ for every input $(n_1, \ldots, n_k)$

Previous work

Expressive power thoroughly studied:

- PPs compute exactly the Presburger predicates
  (Angluin *et al.* 2007)

Previous work

Expressive power thoroughly studied:

- PPs compute exactly the Presburger predicates (Angluin *et al.* 2007)
- Probabilistic PPs (Angluin *et al.* 2004-2006, Chatzigiannakis and Spirakis, 2008)
- Fault-tolerant PPs (Delporte-Gallet *et al.* 2006)
- Private computation in PPs (Delporte-Gallet *et al.* 2007)
- PPs with identifiers (Guerraoui *et al.* 2007)
- PPs with a leader (Angluin *et al.* 2008)
- Mediated PPs (Michail *et al.*, 2011)
- Trustful PPs (Bournez *et al.*, 2013)

Well-specified protocols

Q: And if some fair execution does not stabilize ?

Well-specified protocols

Q: And if some fair execution does not stabilize ?

A: *Then your protocol is not well specified. Repair it!*

Well-specified protocols

Q: And if some fair execution does not stabilize ?

A: *Then your protocol is not well specified. Repair it!*

Q: And if two fair executions for the same input stabilize to different values?

Well-specified protocols

Q: And if some fair execution does not stabilize ?

A: *Then your protocol is not well specified. Repair it!*

Q: And if two fair executions for the same input stabilize to different values?

A: *Then your protocol is not well specified. Repair it!*

Well-specified protocols

Q: And if some fair execution does not stabilize ?

A: *Then your protocol is not well specified. Repair it!*

Q: And if two fair executions for the same input stabilize to different values?

A: *Then your protocol is not well specified. Repair it!*

Q: And how do I know if my protocol is well specified?

Well-specified protocols

Q: And if some fair execution does not stabilize ?

A: *Then your protocol is not well specified. Repair it!*

Q: And if two fair executions for the same input stabilize to different values?

A: *Then your protocol is not well specified. Repair it!*

Q: And how do I know if my protocol is well specified?

A: *That's your problem . . .*

Well-specified protocols

Q: And if some fair execution does not stabilize ?

A: *Then your protocol is not well specified. Repair it!*

Q: And if two fair executions for the same input stabilize to different values?

A: *Then your protocol is not well specified. Repair it!*

Q: And how do I know if my protocol is well specified?

A: *That's your problem . . .*

Well-specification problem: Given a protocol, decide if it is well-specified.

Correctness problem: Given a protocol and a Presburger predicate, decide if the protocol is well-specified and computes the predicate.

Verifying population protocols: Previous work

- Use model-checkers (SPIN, PRISM , . . .) to verify correctness for some inputs
  Pang *et al.*, 2008; Sun *et al.*, 2009; Clément *et al.*, 2011

Verifying population protocols: Previous work

- Use model-checkers (SPIN, PRISM , . . .) to verify correctness for some inputs
  Pang *et al.*, 2008; Sun *et al.*, 2009; Clément *et al.*, 2011
- Use dedicated programs to check sufficient conditions for well-specification
  Chatzigiannakis *et al.*, 2010

Verifying population protocols: Previous work

- Use model-checkers (SPIN, PRISM , . . .) to verify correctness for some inputs
  Pang *et al.*, 2008; Sun *et al.*, 2009; Clément *et al.*, 2011
- Use dedicated programs to check sufficient conditions for well-specification
  Chatzigiannakis *et al.*, 2010
- Use interactive theorem provers (Coq)
  Deng *et al.*, 2009 and 2011

Verifying population protocols: Previous work

- Use model-checkers (SPIN, PRISM , . . .) to verify correctness for some inputs
  Pang *et al.*, 2008; Sun *et al.*, 2009; Clément *et al.*, 2011
- Use dedicated programs to check sufficient conditions for well-specification
  Chatzigiannakis *et al.*, 2010
- Use interactive theorem provers (Coq)
  Deng *et al.*, 2009 and 2011

Not complete or not automatic.

Main result

Are the well-specification and correctness problems decidable?

Main result

Are the well-specification and correctness problems decidable?

Open for about 10 years.

Main result

Are the well-specification and correctness problems decidable?

Open for about 10 years.

Theorem: The well-specification and correctness problems can be reduced to the reachability problem for Petri nets, and are thus decidable.

Main result

Are the well-specification and correctness problems decidable?

Open for about 10 years.

Theorem: The well-specification and correctness problems can be reduced to the reachability problem for Petri nets, and are thus decidable.

Theorem: The reachability problem for Petri nets can be reduced to the well-specification and correctness problems for PPs with leader.

From PPs to Petri nets

Population protocols	Petri nets
State	Place

From PPs to Petri nets

Population protocols	Petri nets
State	Place
Interaction $(q_1, q_2) \mapsto (q'_1, q'_2)$	Transition with input places q_1, q_2 output places q'_1, q'_2

From PPs to Petri nets

Population protocols	Petri nets
State	Place
Interaction $(q_1, q_2) \mapsto (q_1', q_2')$	Transition with input places q_1, q_2 output places q_1', q_2'
PP-scheme	Net without marking

From PPs to Petri nets

Population protocols	Petri nets
State	Place
Interaction $(q_1, q_2) \mapsto (q_1', q_2')$	Transition with input places q_1, q_2 output places q_1', q_2'
PP-scheme	Net without marking
Configuration	Marking

From PPs to Petri nets

Population protocols	Petri nets
State	Place
Interaction $(q_1, q_2) \mapsto (q_1', q_2')$	Transition with input places q_1, q_2 output places q_1', q_2'
PP-scheme	Net without marking
Configuration	Marking
Configuration graph	Reachability graph

From PPs to Petri nets

Population protocols	Petri nets
State	Place
Interaction $(q_1, q_2) \mapsto (q_1', q_2')$	Transition with input places q_1, q_2 output places q_1', q_2'
PP-scheme	Net without marking
Configuration	Marking
Configuration graph	Reachability graph
PP	Net $+$ infinite family of initial markings

Well-specification is decidable

Fact: Every fair execution of a PP gets eventually trapped in a bottom SCC of its configuration graph, and visits all its states infinitely often.

Well-specification is decidable

Fact: Every fair execution of a PP gets eventually trapped in a bottom SCC of its configuration graph, and visits all its states infinitely often.

Fact: A PP is not well-specified iff there is an initial configuration C and

- a bottom configuration C' reachable from C with agents in both true and false states; or

Well-specification is decidable

Fact: Every fair execution of a PP gets eventually trapped in a bottom SCC of its configuration graph, and visits all its states infinitely often.

Fact: A PP is not well-specified iff there is an initial configuration C and

- a bottom configuration C' reachable from C with agents in both true and false states; or
- two bottom configurations C_1 and C_2, one "true" and one "false", both reachable from C.

Well-specification is decidable

Fact: Every fair execution of a PP gets eventually trapped in a bottom SCC of its configuration graph, and visits all its states infinitely often.

Fact: A PP is not well-specified iff there is an initial configuration C and

- a bottom configuration C' reachable from C with agents in both true and false states; or
- two bottom configurations C_1 and C_2, one "true" and one "false", both reachable from C.

Key Theorem: The set of bottom configurations of a PP is effectively Presburger.

Well-specification is decidable

Given a PP, let

- $\mathcal{N}$: Petri net for the PP
- $\mathcal{I}$: markings corresponding to initial configurations
- $\mathcal{B}$: markings corresponding to bottom configurations

Well-specification is decidable

Given a PP, let

- $\mathcal{N}$: Petri net for the PP
- $\mathcal{I}$: markings corresponding to initial configurations
- $\mathcal{B}$: markings corresponding to bottom configurations

Decision procedure:

Well-specification is decidable

Given a PP, let

- $\mathcal{N}$: Petri net for the PP
- $\mathcal{I}$: markings corresponding to initial configurations
- $\mathcal{B}$: markings corresponding to bottom configurations

Decision procedure:

- Partition $\mathcal{B}$ into $\mathcal{B}_{\text{true}}$, $\mathcal{B}_{\text{false}}$, $\mathcal{B}_{\text{neither}}$

Well-specification is decidable

Given a PP, let

- $\mathcal{N}$: Petri net for the PP
- $\mathcal{I}$: markings corresponding to initial configurations
- $\mathcal{B}$: markings corresponding to bottom configurations

Decision procedure:

- Partition $\mathcal{B}$ into $\mathcal{B}_{\text{true}}$, $\mathcal{B}_{\text{false}}$, $\mathcal{B}_{\text{neither}}$
- Check if $\mathcal{B}_{\text{neither}}$ is reachable from $\mathcal{I}$ (using reachability in Petri nets)

Well-specification is decidable

Given a PP, let

- $\mathcal{N}$: Petri net for the PP
- $\mathcal{I}$: markings corresponding to initial configurations
- $\mathcal{B}$: markings corresponding to bottom configurations

Decision procedure:

- Partition $\mathcal{B}$ into $\mathcal{B}_{\text{true}}$, $\mathcal{B}_{\text{false}}$, $\mathcal{B}_{\text{neither}}$
- Check if $\mathcal{B}_{\text{neither}}$ is reachable from $\mathcal{I}$ (using reachability in Petri nets)
- Construct the net $\mathcal{N} \parallel \mathcal{N}$ (two copies of $\mathcal{N}$ side by side).

Well-specification is decidable

Given a PP, let

- $\mathcal{N}$: Petri net for the PP
- $\mathcal{I}$: markings corresponding to initial configurations
- $\mathcal{B}$: markings corresponding to bottom configurations

Decision procedure:

- Partition $\mathcal{B}$ into $\mathcal{B}_{\text{true}}$, $\mathcal{B}_{\text{false}}$, $\mathcal{B}_{\text{neither}}$
- Check if $\mathcal{B}_{\text{neither}}$ is reachable from $\mathcal{I}$
  (using reachability in Petri nets)
- Construct the net $\mathcal{N} \parallel \mathcal{N}$ (two copies of $\mathcal{N}$ side by side).
- Construct the set $\mathcal{I}_2 = \{(M, M) \mid M \in \mathcal{I}\}$.

Well-specification is decidable

Given a PP, let

- $\mathcal{N}$: Petri net for the PP
- $\mathcal{I}$: markings corresponding to initial configurations
- $\mathcal{B}$: markings corresponding to bottom configurations

Decision procedure:

- Partition $\mathcal{B}$ into $\mathcal{B}_{\mathrm{true}}$, $\mathcal{B}_{\mathrm{false}}$, $\mathcal{B}_{\mathrm{neither}}$
- Check if $\mathcal{B}_{\mathrm{neither}}$ is reachable from $\mathcal{I}$
  (using reachability in Petri nets)
- Construct the net $\mathcal{N} \parallel \mathcal{N}$ (two copies of $\mathcal{N}$ side by side).
- Construct the set $\mathcal{I}_2 = \{(M, M) \mid M \in \mathcal{I}\}$.
- Check if $\mathcal{B}_{\mathrm{true}} \times \mathcal{B}_{\mathrm{false}}$ is reachable from $\mathcal{I}_2$
  (using reachability in Petri nets)

And to conclude ...

- Decidability of correctness: similar argument

And to conclude ...

- Decidability of correctness: similar argument
- Reduction from the single-zero-place reachability problem to well-specification of PPs with leader problem

And to conclude ...

- Decidability of correctness: similar argument
- Reduction from the single-zero-place reachability problem to well-specification of PPs with leader problem
- Open problems: complexity of the promise correctness problem, complexity for PPs without leader.

And to conclude ...

- Decidability of correctness: similar argument
- Reduction from the single-zero-place reachability problem to well-specification of PPs with leader problem
- Open problems: complexity of the promise correctness problem, complexity for PPs without leader.

Thank You