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 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 indistinguishable, 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 indistinguishable, 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?
Ninjas randomly wander around the garden, interacting when they bump into each other
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)
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 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:

\[
\begin{align*}
(YA, NA) &\rightarrow (NP, NP) \\
(YA, NP) &\rightarrow (YA, YP) \\
(NA, YP) &\rightarrow (NA, NP) \\
(NP, YP) &\rightarrow (NP, NP)
\end{align*}
\]

Random bumps guarantee eventual consensus.
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:

\[(YA, NA) \mapsto (NP, NP)\]
\[(YA, NP) \mapsto (YA, YP)\]
\[(NA, YP) \mapsto (NA, NP)\]
\[(NP, YP) \mapsto (NP, NP)\]

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

... and ninjas
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

\[ \text{identical, finite-state, and mobile agents} \]

like

- ad-hoc networks of mobile sensors
- “soups” of interacting molecules
- people in social networks
- ... and ninjas
A PP-scheme is a pair $(Q, \Delta)$, 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 $q'_1$ and $q'_2$. 

Assumption: at least one interaction for each $(q_1, q_2)$.
A **PP-scheme** is a pair $(Q, \Delta)$, 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 $C : Q \rightarrow \mathbb{N}$, where $C(q)$ is the current number of agents in state $q$.

$$
\begin{array}{c}
\circ \quad \circ \quad \circ \quad \circ \\
2 \quad 1 \quad 0 \quad 3
\end{array}
$$
Semantics

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

\[
\begin{array}{cccc}
q_1 & q_2 & q_3 & q_4 \\
\circ2 & \circ1 & \circ0 & \circ3 \\
\end{array}
\]

\[(q_1, q_2) \mapsto (q_3, q_4)\]
**Semantics**

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

\[
\begin{align*}
q_1 & \quad q_2 & \quad q_3 & \quad q_4 \\
2 & \quad 1 & \quad 0 & \quad 3 \\
\end{align*}
\overset{\rightarrow}{\longrightarrow}
\begin{align*}
q_1 & \quad q_2 & \quad q_3 & \quad q_4 \\
1 & \quad 0 & \quad 1 & \quad 4 \\
\end{align*}
\]

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

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

If several steps are possible, a scheduler chooses one
**Semantics**

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

\[
\begin{array}{cccc}
q_1 & q_2 & q_3 & q_4 \\
\circ2 & \circ1 & \circ0 & \circ3 \\
\end{array} \quad \rightarrow \quad \begin{array}{cccc}
q_1 & q_2 & q_3 & q_4 \\
\circ1 & \circ0 & \circ1 & \circ4 \\
\end{array}
\]

\[(q_1, q_2) \mapsto (q_3, q_4)\]

If several steps are possible, a scheduler chooses one.

**Execution:** infinite sequence $C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow \cdots$ of steps
Semantics

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

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

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

If several steps are possible, a scheduler chooses one.

**Execution:** infinite sequence $C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow \cdots$ of steps.

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

(Fairness constraint approximating random scheduler)
A population protocol (PP) consists of:

- A PP-scheme \((Q, \Delta)\)

\[ Q: \quad \bigcirc \quad \bigcirc \quad \bigcirc \quad \bigcirc \quad \bigcirc \quad \bigcirc \quad \bigcirc \quad \bigcirc \quad \bigcirc \quad \bigcirc \]
A population protocol (PP) consists of

- A PP-scheme \((Q, \Delta)\)
- A tuple \((in_1, \ldots, in_k)\) of input states

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

- A PP-scheme \((Q, \Delta)\)
- 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 \(b \in \{\text{true}, \text{false}\}\) if from some point on every agent stays within the \(b\)-states. ("All agents agree on \(b\).")
A population protocol (PP) consists of

- A PP-scheme \((Q, \Delta)\)
- 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 \(b \in \{\text{true, false}\}\) if from some point on every agent stays within the \(b\)-states. ("All agents agree on \(b\)").
A PP computes the value $b$ for input $(n_1, \ldots, n_k)$ if every fair execution starting at the configuration $\text{in}_1$-$\text{in}_2$ stabilizes to $b$. Intuitively: all agents agree on $b$ whatever the (random) scheduler.
A PP computes the value $b$ for input $(n_1, \ldots, n_k)$ if every fair execution starting at the configuration

$$n_1 \cdot \text{in}_1$$

Intuitively: all agents agree on $b$ whatever the (random) scheduler $A$ PP computes $P$:

$$n \rightarrow \{\text{true}, \text{false}\}$$

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

$$n_1 \cdot \text{in}_1 + \cdots + n_k \cdot \text{in}_k$$
A PP computes the value $b$ for input $(n_1, \ldots, n_k)$ if every fair execution starting at the configuration

$$n_1 \cdot \text{in}_1 + \cdots + n_k \cdot \text{in}_k$$

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

$$n_1 \cdot \text{in}_1 + \cdots + n_k \cdot \text{in}_k$$

stabilizes to $b$.

Intuitively: all agents agree on $b$ whatever the (random) scheduler.
A PP computes the value $b$ for input $(n_1, \ldots, n_k)$ if every fair execution starting at the configuration

$$n_1 \cdot \text{in}_1 + \cdots + n_k \cdot \text{in}_k$$

stabilizes to $b$.

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

A PP computes $P: \mathbb{N}^n \rightarrow \{\text{true, false}\}$ 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)
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.
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.
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?
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 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?
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?

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.
Main result

Are the well-specification and correctness problems decidable?

Open for about 10 years.
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

<table>
<thead>
<tr>
<th>Population protocols</th>
<th>Petri nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>Place</td>
</tr>
</tbody>
</table>

Interaction: 

\[ (q_1, q_2) \mapsto (q'_1, q'_2) \]

Input places: \( q_1, q_2 \)

Output places: \( q'_1, q'_2 \)
### From PPs to Petri nets

<table>
<thead>
<tr>
<th>Population protocols</th>
<th>Petri nets</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>State</strong></td>
<td><strong>Place</strong></td>
</tr>
<tr>
<td>Interaction</td>
<td>Transition with</td>
</tr>
<tr>
<td>((q_1, q_2) \mapsto (q'_1, q'_2))</td>
<td>input places (q_1, q_2) output places (q'_1, q'_2)</td>
</tr>
<tr>
<td>Population protocols</td>
<td>Petri nets</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------</td>
</tr>
<tr>
<td><strong>State</strong></td>
<td><strong>Place</strong></td>
</tr>
<tr>
<td>Interaction</td>
<td>Transition with</td>
</tr>
<tr>
<td>((q_1, q_2) \mapsto (q'_1, q'_2))</td>
<td>input places (q_1, q_2)</td>
</tr>
<tr>
<td>PP-scheme</td>
<td>output places (q'_1, q'_2)</td>
</tr>
<tr>
<td></td>
<td>Net without marking</td>
</tr>
</tbody>
</table>
## From PPs to Petri nets

<table>
<thead>
<tr>
<th>Population protocols</th>
<th>Petri nets</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>State</strong></td>
<td>Place</td>
</tr>
<tr>
<td><strong>Interaction</strong></td>
<td>Transition with</td>
</tr>
<tr>
<td>$(q_1, q_2) \mapsto (q_1', q_2')$</td>
<td>input places $q_1, q_2$</td>
</tr>
<tr>
<td>PP-scheme</td>
<td>Net without marking</td>
</tr>
<tr>
<td>Configuration</td>
<td>Marking</td>
</tr>
</tbody>
</table>
## From PPs to Petri nets

<table>
<thead>
<tr>
<th>Population protocols</th>
<th>Petri nets</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>State</strong></td>
<td><strong>Place</strong></td>
</tr>
<tr>
<td>Interaction</td>
<td>Transition with</td>
</tr>
<tr>
<td>$(q_1, q_2) \mapsto (q_1', q_2')$</td>
<td>input places $q_1, q_2$</td>
</tr>
<tr>
<td><strong>PP-scheme</strong></td>
<td><strong>Net without marking</strong></td>
</tr>
<tr>
<td><strong>Configuration</strong></td>
<td><strong>Marking</strong></td>
</tr>
<tr>
<td><strong>Configuration graph</strong></td>
<td><strong>Reachability graph</strong></td>
</tr>
<tr>
<td>Population protocols</td>
<td>Petri nets</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
</tr>
<tr>
<td><strong>State</strong></td>
<td>Place</td>
</tr>
<tr>
<td>Interaction</td>
<td>Transition with</td>
</tr>
<tr>
<td>((q_1, q_2) \mapsto (q'_1, q'_2))</td>
<td>input places (q_1, q_2)</td>
</tr>
<tr>
<td>PP-scheme</td>
<td>Net without marking</td>
</tr>
<tr>
<td>Configuration</td>
<td>Marking</td>
</tr>
<tr>
<td>Configuration graph</td>
<td>Reachability graph</td>
</tr>
<tr>
<td>PP</td>
<td>Net + infinite family of initial markings</td>
</tr>
</tbody>
</table>
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: 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
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 $C$. 
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

Decision procedure:

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