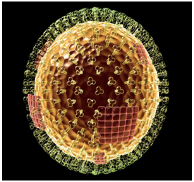
Algorithmic Verification of Population Protocols

Othon Michail
Joint work with:
Ioannis Chatzigiannakis
Paul Spirakis

Research Academic Computer Technology Institute (RACTI)

SSS 2010
September 2010

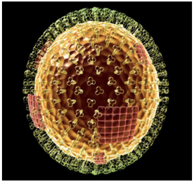
Monitoring Cows' Health



- Equip each cow in a heard with a sensor detecting influenza
- The sensor gives output 1

$$\begin{cases} 1, & \text{if the cow is infected} \\ 0, & \text{if it is not} \end{cases}$$

Monitoring Cows' Health



- Equip each cow in a heard with a sensor detecting influenza
- The sensor gives output 1

$$\begin{cases} \color{red}{1}, & \text{if the cow is infected} \\ \color{red}{0}, & \text{if it is not} \end{cases}$$

Monitoring Cows' Health

- Question: Are there at least 5 infected cows?
- A solution:
  - The base station informs all agents to sense their environment
  - When 2 cows come close to each other their agents interact
  - The initiator takes the sum of the values and the responder takes 0
  - If a sum reaches 5 the output value 1 is propagated, otherwise forever remains 0

Monitoring Cows' Health

- Question: Are there at least 5 infected cows?
- A solution:
  - The base station informs all agents to sense their environment
  - When 2 cows come close to each other their agents interact
  - The initiator takes the sum of the values and the responder takes 0
  - If a sum reaches 5 the output value 1 is propagated, otherwise forever remains 0

Monitoring Cows' Health

- Question: Are there at least 5 infected cows?
- A solution:
  - The base station informs all agents to sense their environment
  - When 2 cows come close to each other their agents interact
  - The initiator takes the sum of the values and the responder takes 0
  - If a sum reaches 5 the output value 1 is propagated, otherwise forever remains 0

Monitoring Cows' Health

- Question: Are there at least 5 infected cows?
- A solution:
  - The base station informs all agents to sense their environment
  - When 2 cows come close to each other their agents interact
  - The initiator takes the sum of the values and the responder takes 0
  - If a sum reaches 5 the output value 1 is propagated, otherwise forever remains 0

Monitoring Cows' Health

- Question: Are there at least 5 infected cows?
- A solution:
  - The base station informs all agents to sense their environment
  - When 2 cows come close to each other their agents interact
  - The initiator takes the sum of the values and the responder takes 0
  - If a sum reaches 5 the output value 1 is propagated, otherwise forever remains 0

Monitoring Cows' Health

- Question: Are there at least 5 infected cows?
- A solution:
  - The base station informs all agents to sense their environment
  - When 2 cows come close to each other their agents interact
  - The initiator takes the sum of the values and the responder takes 0
  - If a sum reaches 5 the output value 1 is propagated, otherwise forever remains 0

Outstanding Properties of PPs

[AADFP '04]

The agents

- have constant memory (uniformity)
- do not have uids (anonymity)
- are passively mobile

Outstanding Properties of PPs

[AADFP '04]

The agents

- have constant memory (uniformity)
- do not have uids (anonymity)
- are passively mobile

Outstanding Properties of PPs

[AADFP '04]

The agents

- have constant memory (uniformity)
- do not have uids (anonymity)
- are passively mobile

A Correct Population Protocol

- Input alphabet $X = \{0, 1\}$
- Output alphabet $Y = \{0, 1\}$
- Set of states $Q = \{q_0, q_1, \ldots, q_5\}$
- Input function $I : X \to Q$, defined as $I(\sigma) = q_\sigma$,
- Output function $O : Q \to Y$, defined as $O(q_5) = 1$ and $O(q) = 0$ for all $q \in Q - \{q_5\}$
- Transition function δ:

$$\begin{aligned}(q_i, q_j) &\to (q_{i+j}, q_0), \text{ if } i + j < 5 \\ &\to (q_5, q_5), \text{ otherwise}\end{aligned}$$

A Correct Population Protocol

- Input alphabet $X = \{0, 1\}$
- Output alphabet $Y = \{0, 1\}$
- Set of states $Q = \{q_0, q_1, \ldots, q_5\}$
- Input function $I : X \to Q$, defined as $I(\sigma) = q_\sigma$,
- Output function $O : Q \to Y$, defined as $O(q_5) = 1$ and $O(q) = 0$ for all $q \in Q - \{q_5\}$
- Transition function δ:

$$\begin{aligned}(q_i, q_j) &\to (q_{i+j}, q_0), \text{ if } i + j < 5\\ &\to (q_5, q_5), \text{ otherwise}\end{aligned}$$

A Correct Population Protocol

- Input alphabet $X = \{0, 1\}$
- Output alphabet $Y = \{0, 1\}$
- Set of states $Q = \{q_0, q_1, \ldots, q_5\}$
- Input function $I : X \to Q$, defined as $I(\sigma) = q_\sigma$,
- Output function $O : Q \to Y$, defined as $O(q_5) = 1$ and $O(q) = 0$ for all $q \in Q - \{q_5\}$
- Transition function δ:

$$\begin{aligned}(q_i, q_j) &\to (q_{i+j}, q_0),\ \text{if } i + j < 5 \\ &\to (q_5, q_5),\ \text{otherwise}\end{aligned}$$

A Correct Population Protocol

- Input alphabet $X = \{0, 1\}$
- Output alphabet $Y = \{0, 1\}$
- Set of states $Q = \{q_0, q_1, \ldots, q_5\}$
- Input function $I : X \to Q$, defined as $I(\sigma) = q_\sigma$,
- Output function $O : Q \to Y$, defined as $O(q_5) = 1$ and $O(q) = 0$ for all $q \in Q - \{q_5\}$
- Transition function δ:

$$\begin{aligned}(q_i, q_j) &\to (q_{i+j}, q_0), \text{ if } i + j < 5\\ &\to (q_5, q_5), \text{ otherwise}\end{aligned}$$

A Correct Population Protocol

- Input alphabet $X = \{0, 1\}$
- Output alphabet $Y = \{0, 1\}$
- Set of states $Q = \{q_0, q_1, \ldots, q_5\}$
- Input function $I : X \to Q$, defined as $I(\sigma) = q_\sigma$,
- Output function $O : Q \to Y$, defined as $O(q_5) = 1$ and $O(q) = 0$ for all $q \in Q - \{q_5\}$
- Transition function δ:

$$\begin{aligned}(q_i, q_j) &\to (q_{i+j}, q_0), \text{ if } i + j < 5\\ &\to (q_5, q_5), \text{ otherwise}\end{aligned}$$

A Correct Population Protocol

- Input alphabet $X = \{0, 1\}$
- Output alphabet $Y = \{0, 1\}$
- Set of states $Q = \{q_0, q_1, \ldots, q_5\}$
- Input function $I : X \to Q$, defined as $I(\sigma) = q_\sigma$,
- Output function $O : Q \to Y$, defined as $O(q_5) = 1$ and $O(q) = 0$ for all $q \in Q - \{q_5\}$
- Transition function δ:

$$\begin{aligned}(q_i, q_j) &\to (q_{i+j}, q_0),\ \text{if } i + j < 5 \\ &\to (q_5, q_5),\ \text{otherwise}\end{aligned}$$

The Code

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	(q_0, q_0)	(q_1, q_0)	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)
q_1	(q_1, q_0)	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)
q_2	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)
q_3	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)
q_4	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)
q_5	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)

A Typo Occured

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	(q_0, q_0)	(q_1, q_0)	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)
q_1	(q_1, q_0)	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)
q_2	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)
q_3	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)
q_4	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_4, q_0)	(q_5, q_5)
q_5	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)

A Typo Occured

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	(q_0, q_0)	(q_1, q_0)	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)
q_1	(q_1, q_0)	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)
q_2	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)
q_3	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)
q_4	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_4, q_0)	(q_5, q_5)
q_5	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)

A Typo Occured

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	(q_0, q_0)	(q_1, q_0)	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)
q_1	(q_1, q_0)	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)
q_2	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)
q_3	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)
q_4	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_4, q_0)	(q_5, q_5)
q_5	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)

- 400 cows are all sick
- It is possible that 100 agents go to q_4 and the rest to q_0
- But now the farmer will never be alarmed of the problem (alarm state q_5 never appears)
- Interestingly, the protocol also has an erroneous computation for only 8 cows

A Typo Occured

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	(q_0, q_0)	(q_1, q_0)	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)
q_1	(q_1, q_0)	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)
q_2	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)
q_3	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)
q_4	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_4, q_0)	(q_5, q_5)
q_5	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)

- 400 cows are all sick
- It is possible that 100 agents go to q_4 and the rest to q_0
- But now the farmer will never be alarmed of the problem (alarm state q_5 never appears)
- Interestingly, the protocol also has an erroneous computation for only 8 cows

A Typo Occured

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	(q_0, q_0)	(q_1, q_0)	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)
q_1	(q_1, q_0)	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)
q_2	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)
q_3	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)
q_4	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_4, q_0)	(q_5, q_5)
q_5	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)

- 400 cows are all sick
- It is possible that 100 agents go to q_4 and the rest to q_0
- But now the farmer will never be alarmed of the problem (alarm state q_5 never appears)
- Interestingly, the protocol also has an erroneous computation for only 8 cows

A Typo Occured

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	(q_0, q_0)	(q_1, q_0)	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)
q_1	(q_1, q_0)	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)
q_2	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)
q_3	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)
q_4	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_4, q_0)	(q_5, q_5)
q_5	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)

- 400 cows are all sick
- It is possible that 100 agents go to q_4 and the rest to q_0
- But now the farmer will never be alarmed of the problem (alarm state q_5 never appears)
- Interestingly, the protocol also has an erroneous computation for only 8 cows

Algorithmic Verification

- We want to algorithmically verify our protocols in order to avoid such total or partial failures, particularly in critical applications

Problem (*GBPVER*)

Given a population protocol $\mathcal{A}$ for the basic model for which $Y_{\mathcal{A}} = \{0, 1\}$ and a first-order logical formula ϕ in Presburger arithmetic representing the specifications of $\mathcal{A}$ determine whether $\mathcal{A}$ conforms to ϕ.

- Conforms : For any input assignment x, and no matter how the computation proceeds, an output-stable configuration is eventually reached under which all agents output $\phi(x)$

Algorithmic Verification

- We want to algorithmically verify our protocols in order to avoid such total or partial failures, particularly in critical applications

Problem (*GBPVER*)

Given a population protocol $\mathcal{A}$ for the basic model for which $Y_{\mathcal{A}} = \{0, 1\}$ and a first-order logical formula ϕ in Presburger arithmetic representing the specifications of $\mathcal{A}$ determine whether $\mathcal{A}$ conforms to ϕ.

- Conforms : For any input assignment x, and no matter how the computation proceeds, an output-stable configuration is eventually reached under which all agents output $\phi(x)$

Algorithmic Verification

- We want to algorithmically verify our protocols in order to avoid such total or partial failures, particularly in critical applications

Problem (*GBPVER*)

Given a population protocol $\mathcal{A}$ for the basic model for which $Y_{\mathcal{A}} = \{0, 1\}$ and a first-order logical formula ϕ in Presburger arithmetic representing the specifications of $\mathcal{A}$ determine whether $\mathcal{A}$ conforms to ϕ.

- Conforms : For any input assignment x, and no matter how the computation proceeds, an output-stable configuration is eventually reached under which all agents output $\phi(x)$

BPVER

- We mainly focus on the somewhat easier *BPVER* problem:
  - An integer $n \geq 2$ is also provided as part of the input
  - Here we want to determine whether $\mathcal{A}$ conforms to ϕ on K_n (complete communication digraph of n agents)
- Since a computation is infinite and there is a finite number of configurations, at least one configuration appears infinitely often
  - It is known [AADFP '06] that those configurations form a final strongly connected component of the transition graph $G(\mathcal{C}, E)$, where $\mathcal{C}$ is the set of all configurations and $(c, c') \in E$ iff $c \to c'$, e.g.

BPVER

- We mainly focus on the somewhat easier *BPVER* problem:
  - An integer $n \geq 2$ is also provided as part of the input
  - Here we want to determine whether $\mathcal{A}$ conforms to ϕ on K_n (complete communication digraph of n agents)
- Since a computation is infinite and there is a finite number of configurations, at least one configuration appears infinitely often
  - It is known [AADFP '06] that those configurations form a final strongly connected component of the transition graph $G(\mathcal{C}, E)$, where $\mathcal{C}$ is the set of all configurations and $(c, c') \in E$ iff $c \rightarrow c'$, e.g.

BPVER

- We mainly focus on the somewhat easier *BPVER* problem:
  - An integer $n \geq 2$ is also provided as part of the input
  - Here we want to determine whether $\mathcal{A}$ conforms to ϕ on K_n (complete communication digraph of n agents)
- Since a computation is infinite and there is a finite number of configurations, at least one configuration appears infinitely often
  - It is known [AADFP '06] that those configurations form a final strongly connected component of the transition graph $G(\mathcal{C}, E)$, where $\mathcal{C}$ is the set of all configurations and $(c, c') \in E$ iff $c \to c'$, e.g.

BPVER

- We mainly focus on the somewhat easier *BPVER* problem:
  - An integer $n \geq 2$ is also provided as part of the input
  - Here we want to determine whether $\mathcal{A}$ conforms to ϕ on K_n (complete communication digraph of n agents)
- Since a computation is infinite and there is a finite number of configurations, at least one configuration appears infinitely often
  - It is known [AADFP '06] that those configurations form a final strongly connected component of the transition graph $G(\mathcal{C}, E)$, where $\mathcal{C}$ is the set of all configurations and $(c, c') \in E$ iff $c \rightarrow c'$, e.g.

BPVER

- We mainly focus on the somewhat easier *BPVER* problem:
  - An integer $n \geq 2$ is also provided as part of the input
  - Here we want to determine whether $\mathcal{A}$ conforms to ϕ on K_n (complete communication digraph of n agents)
- Since a computation is infinite and there is a finite number of configurations, at least one configuration appears infinitely often
  - It is known [AADFP '06] that those configurations form a final strongly connected component of the transition graph $G(\mathcal{C}, E)$, where $\mathcal{C}$ is the set of all configurations and $(c, c') \in E$ iff $c \rightarrow c'$, e.g.

BPVER

- We first focus on the somewhat easier *BPVER* problem ('B': Basic model, 'P': Predicate):
  - An integer $n \geq 2$ is also provided as part of the input.
  - Here we want to determine whether $\mathcal{A}$ conforms to ϕ on K_n (complete communication digraph of n agents).
- Since a computation is infinite and there is a finite number of configurations, at least one configuration appears infinitely often.
  - It is known that those configurations form a final strongly connected component of the transition graph $G(\mathcal{C}, E)$, where $\mathcal{C}$ is the set of all configurations and $(c, c') \in E$ iff $c \to c'$, e.g.

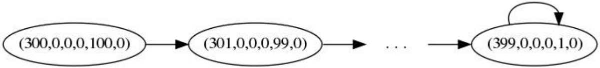


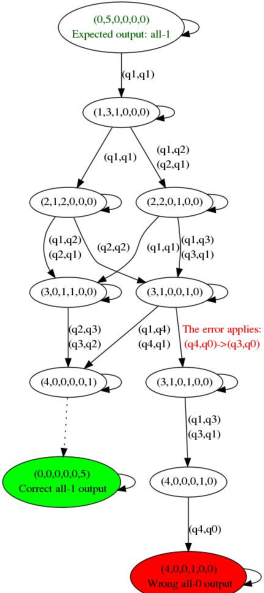
Figure: $(c_i)_{i=0,\ldots,5}$, c_i denotes the number of agents in state q_i

Another Typo

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	(q_0, q_0)	(q_1, q_0)	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)
q_1	(q_1, q_0)	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)
q_2	(q_2, q_0)	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)
q_3	(q_3, q_0)	(q_4, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)
q_4	(q_3, q_0)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)
q_5	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)	(q_5, q_5)

- It should be $(q_4, q_0) \to (q_4, q_0)$ instead, because now one counter is decreased without a reason

Another Typo: $(q_4, q_0) \rightarrow (q_3, q_0)$



Hardness of *BPVER*

Theorem

BPVER is coNP-hard*.*

- The reduction is from *HAMPATH* (directed)
- Given $\langle D, s, t \rangle$
- We construct a protocol $\mathcal{A}$ that does not conform to $(N_x < 0)$ on K_n iff D contains a directed hamiltonian path from s to t
- Also return $n = k - 1$, where $k = |V(D)|$
- The input alphabet X consists of the edges of D

Hardness of *BPVER*

Theorem

BPVER is coNP-hard.

- The reduction is from *HAMPATH* (directed)
- Given $\langle D, s, t\rangle$
- We construct a protocol $\mathcal{A}$ that does not conform to $(N_x < 0)$ on K_n iff D contains a directed hamiltonian path from s to t
- Also return $n = k - 1$, where $k = |V(D)|$
- The input alphabet X consists of the edges of D

Hardness of *BPVER*

Theorem

BPVER is coNP-hard.

- The reduction is from *HAMPATH* (directed)
- Given $\langle D, s, t \rangle$
- We construct a protocol $\mathcal{A}$ that does not conform to $(N_x < 0)$ on K_n iff D contains a directed hamiltonian path from s to t
- Also return $n = k - 1$, where $k = |V(D)|$
- The input alphabet X consists of the edges of D

Hardness of *BPVER*

Theorem

BPVER is coNP-hard.

- The reduction is from *HAMPATH* (directed)
- Given $\langle D, s, t\rangle$
- We construct a protocol $\mathcal{A}$ that does not conform to $(N_x < 0)$ on K_n iff D contains a directed hamiltonian path from s to t
- Also return $n = k - 1$, where $k = |V(D)|$
- The input alphabet X consists of the edges of D

Hardness of *BPVER*

Theorem

BPVER is coNP-hard*.*

- The reduction is from *HAMPATH* (directed)
- Given $\langle D, s, t \rangle$
- We construct a protocol $\mathcal{A}$ that does not conform to $(N_x < 0)$ on K_n iff D contains a directed hamiltonian path from s to t
- Also return $n = k - 1$, where $k = |V(D)|$
- The input alphabet X consists of the edges of D

Hardness of *BPVER*

Theorem

BPVER is coNP-hard.

- The reduction is from *HAMPATH* (directed)
- Given $\langle D, s, t \rangle$
- We construct a protocol $\mathcal{A}$ that does not conform to $(N_x < 0)$ on K_n iff D contains a directed hamiltonian path from s to t
- Also return $n = k - 1$, where $k = |V(D)|$
- The input alphabet X consists of the edges of D

Hardness of *BPVER*

- The protocol tries to verify whether its input assignment is a legal hamiltonian path
- If it encounters some violation, it rejects (otherwise, remains to an accepting output)
- Obviously
  - If $D \notin HAMPATH$ then no input assignment is a hamiltonian path and the protocol conforms to ϕ (always finds some violation and rejects)
  - If $D \in HAMPATH$ then the hamiltonian path is a possible input assignment and the resulting computation will be accepting

Remark: $\mathcal{A}$ is a PP because though its size depends on k, k is independent of the population size n

Hardness of *BPVER*

- The protocol tries to verify whether its input assignment is a legal hamiltonian path
- If it encounters some violation, it rejects (otherwise, remains to an accepting output)
- Obviously
  - If $D \notin HAMPATH$ then no input assignment is a hamiltonian path and the protocol conforms to ϕ (always finds some violation and rejects)
  - If $D \in HAMPATH$ then the hamiltonian path is a possible input assignment and the resulting computation will be accepting

Remark: $\mathcal{A}$ is a PP because though its size depends on k, k is independent of the population size n

Hardness of *BPVER*

- The protocol tries to verify whether its input assignment is a legal hamiltonian path
- If it encounters some violation, it rejects (otherwise, remains to an accepting output)
- Obviously
  - If $D \notin HAMPATH$ then no input assignment is a hamiltonian path and the protocol conforms to ϕ (always finds some violation and rejects)
  - If $D \in HAMPATH$ then the hamiltonian path is a possible input assignment and the resulting computation will be accepting

Remark: $\mathcal{A}$ is a PP because though its size depends on k, k is independent of the population size n

Hardness of *BPVER*

- The protocol tries to verify whether its input assignment is a legal hamiltonian path
- If it encounters some violation, it rejects (otherwise, remains to an accepting output)
- Obviously
  - If $D \notin HAMPATH$ then no input assignment is a hamiltonian path and the protocol conforms to ϕ (always finds some violation and rejects)
  - If $D \in HAMPATH$ then the hamiltonian path is a possible input assignment and the resulting computation will be accepting

Remark: $\mathcal{A}$ is a PP because though its size depends on k, k is independent of the population size n

Hardness of *BPVER*

- The protocol tries to verify whether its input assignment is a legal hamiltonian path
- If it encounters some violation, it rejects (otherwise, remains to an accepting output)
- Obviously
  - If $D \notin HAMPATH$ then no input assignment is a hamiltonian path and the protocol conforms to ϕ (always finds some violation and rejects)
  - If $D \in HAMPATH$ then the hamiltonian path is a possible input assignment and the resulting computation will be accepting

Remark: $\mathcal{A}$ is a PP because though its size depends on k, k is independent of the population size n

Hardness of *BPVER*

- The protocol tries to verify whether its input assignment is a legal hamiltonian path
- If it encounters some violation, it rejects (otherwise, remains to an accepting output)
- Obviously
  - If $D \notin HAMPATH$ then no input assignment is a hamiltonian path and the protocol conforms to ϕ (always finds some violation and rejects)
  - If $D \in HAMPATH$ then the hamiltonian path is a possible input assignment and the resulting computation will be accepting

Remark: $\mathcal{A}$ is a PP because though its size depends on k, k is independent of the population size n

Other Hardness Results

Theorem

GBPVER is coNP-hard.

- *BBPIVER* problem:
  - X is restricted to $\{0,1\}$
  - a specific input assignment is provided as part of the input
  - we ask whether the protocol is correct for that input assignment
- The bad news: even for this restricted version, hardness insists
- We can only hope for exponential-time algorithms that are efficient in practice
  - or for other interesting special cases that are efficiently solvable

Other Hardness Results

Theorem

GBPVER is coNP-hard.

- *BBPIVER* problem:
  - X is restricted to $\{0, 1\}$
  - a specific input assignment is provided as part of the input
  - we ask whether the protocol is correct for that input assignment
- The bad news: even for this restricted version, hardness insists
- We can only hope for exponential-time algorithms that are efficient in practice
  - or for other interesting special cases that are efficiently solvable

Other Hardness Results

Theorem

GBPVER is coNP-hard.

- *BBPIVER* problem:
  - X is restricted to $\{0, 1\}$
  - a specific input assignment is provided as part of the input
  - we ask whether the protocol is correct for that input assignment
- The bad news: even for this restricted version, hardness insists
- We can only hope for exponential-time algorithms that are efficient in practice
  - or for other interesting special cases that are efficiently solvable

Other Hardness Results

Theorem

GBPVER is coNP-hard.

- *BBPIVER* problem:
  - X is restricted to $\{0, 1\}$
  - a specific input assignment is provided as part of the input
  - we ask whether the protocol is correct for that input assignment
- The bad news: even for this restricted version, hardness insists
- We can only hope for exponential-time algorithms that are efficient in practice
  - or for other interesting special cases that are efficiently solvable

Other Hardness Results

Theorem

GBPVER is coNP-hard.

- *BBPIVER* problem:
  - X is restricted to $\{0, 1\}$
  - a specific input assignment is provided as part of the input
  - we ask whether the protocol is correct for that input assignment
- The bad news: even for this restricted version, hardness insists
- We can only hope for exponential-time algorithms that are efficient in practice
  - or for other interesting special cases that are efficiently solvable

Other Hardness Results

Theorem

GBPVER is coNP-hard*.*

- *BBPIVER* problem:
  - X is restricted to $\{0, 1\}$
  - a specific input assignment is provided as part of the input
  - we ask whether the protocol is correct for that input assignment
- The bad news: even for this restricted version, hardness insists
- We can only hope for exponential-time algorithms that are efficient in practice
  - or for other interesting special cases that are efficiently solvable

Other Hardness Results

Theorem

GBPVER is coNP-hard.

- *BBPIVER* problem:
  - X is restricted to $\{0, 1\}$
  - a specific input assignment is provided as part of the input
  - we ask whether the protocol is correct for that input assignment
- The bad news: even for this restricted version, hardness insists
- We can only hope for exponential-time algorithms that are efficient in practice
  - or for other interesting special cases that are efficiently solvable

Other Hardness Results

Theorem

GBPVER is coNP-hard.

- *BBPIVER* problem:
  - X is restricted to $\{0, 1\}$
  - a specific input assignment is provided as part of the input
  - we ask whether the protocol is correct for that input assignment
- The bad news: even for this restricted version, hardness insists
- We can only hope for exponential-time algorithms that are efficient in practice
  - or for other interesting special cases that are efficiently solvable

Protocol Correctness Criteria

1. $\phi(c) = -1$ for some $c \in C_I$
2. $\exists c, c' \in C_I$ such that $c \xrightarrow{*} c'$ and $\phi(c) \neq \phi(c')$
3. $\exists c \in C_I$ and $c' \in C_F$ such that $c \xrightarrow{*} c'$ and $O(c') = -1$
4. $\exists c \in C_I$ and $c' \in C_F$ such that $c \xrightarrow{*} c'$ and $\phi(c) \neq O(c')$
5. $\exists B' \in F_S$ such that $O(B') = -1$
6. $\exists B \in I_S$ and $B' \in F_S$ such that $B \xrightarrow{*} B'$ and $\phi(B) \neq O(B')$ (possibly $B = B'$)

Theorem (Complete Verifier)

Any algorithm checking criteria 1, 5, and 6 decides BPVER.

Protocol Correctness Criteria

1. $\phi(c) = -1$ for some $c \in C_I$
2. $\exists c, c' \in C_I$ such that $c \xrightarrow{*} c'$ and $\phi(c) \neq \phi(c')$
3. $\exists c \in C_I$ and $c' \in C_F$ such that $c \xrightarrow{*} c'$ and $O(c') = -1$
4. $\exists c \in C_I$ and $c' \in C_F$ such that $c \xrightarrow{*} c'$ and $\phi(c) \neq O(c')$
5. $\exists B' \in F_S$ such that $O(B') = -1$
6. $\exists B \in I_S$ and $B' \in F_S$ such that $B \xrightarrow{*} B'$ and $\phi(B) \neq O(B')$ (possibly $B = B'$)

Theorem (Complete Verifier)

Any algorithm checking criteria 1, 5, and 6 decides BPVER.

Protocol Correctness Criteria

1. $\phi(c) = -1$ for some $c \in C_I$
2. $\exists c, c' \in C_I$ such that $c \xrightarrow{*} c'$ and $\phi(c) \neq \phi(c')$
3. $\exists c \in C_I$ and $c' \in C_F$ such that $c \xrightarrow{*} c'$ and $O(c') = -1$
4. $\exists c \in C_I$ and $c' \in C_F$ such that $c \xrightarrow{*} c'$ and $\phi(c) \neq O(c')$
5. $\exists B' \in F_S$ such that $O(B') = -1$
6. $\exists B \in I_S$ and $B' \in F_S$ such that $B \xrightarrow{*} B'$ and $\phi(B) \neq O(B')$ (possibly $B = B'$)

Theorem (Complete Verifier)

Any algorithm checking criteria 1, 5, and 6 decides BPVER.

Protocol Correctness Criteria

1. $\phi(c) = -1$ for some $c \in C_I$
2. $\exists c, c' \in C_I$ such that $c \xrightarrow{*} c'$ and $\phi(c) \neq \phi(c')$
3. $\exists c \in C_I$ and $c' \in C_F$ such that $c \xrightarrow{*} c'$ and $O(c') = -1$
4. $\exists c \in C_I$ and $c' \in C_F$ such that $c \xrightarrow{*} c'$ and $\phi(c) \neq O(c')$
5. $\exists B' \in F_S$ such that $O(B') = -1$
6. $\exists B \in I_S$ and $B' \in F_S$ such that $B \xrightarrow{*} B'$ and $\phi(B) \neq O(B')$ (possibly $B = B'$)

Theorem (Complete Verifier)

Any algorithm checking criteria 1, 5, and 6 decides BPVER.

Protocol Correctness Criteria

1. $\phi(c) = -1$ for some $c \in C_I$
2. $\exists c, c' \in C_I$ such that $c \xrightarrow{*} c'$ and $\phi(c) \neq \phi(c')$
3. $\exists c \in C_I$ and $c' \in C_F$ such that $c \xrightarrow{*} c'$ and $O(c') = -1$
4. $\exists c \in C_I$ and $c' \in C_F$ such that $c \xrightarrow{*} c'$ and $\phi(c) \neq O(c')$
5. $\exists B' \in F_S$ such that $O(B') = -1$
6. $\exists B \in I_S$ and $B' \in F_S$ such that $B \xrightarrow{*} B'$ and $\phi(B) \neq O(B')$ (possibly $B = B'$)

Theorem (Complete Verifier)

Any algorithm checking criteria 1, 5, and 6 decides BPVER.

Protocol Correctness Criteria

1. $\phi(c) = -1$ for some $c \in C_I$
2. $\exists c, c' \in C_I$ such that $c \xrightarrow{*} c'$ and $\phi(c) \neq \phi(c')$
3. $\exists c \in C_I$ and $c' \in C_F$ such that $c \xrightarrow{*} c'$ and $O(c') = -1$
4. $\exists c \in C_I$ and $c' \in C_F$ such that $c \xrightarrow{*} c'$ and $\phi(c) \neq O(c')$
5. $\exists B' \in F_S$ such that $O(B') = -1$
6. $\exists B \in I_S$ and $B' \in F_S$ such that $B \xrightarrow{*} B'$ and $\phi(B) \neq O(B')$ (possibly $B = B'$)

Theorem (Complete Verifier)

Any algorithm checking criteria 1, 5, and 6 decides BPVER.

Protocol Correctness Criteria

1. $\phi(c) = -1$ for some $c \in C_I$
2. $\exists c, c' \in C_I$ such that $c \xrightarrow{*} c'$ and $\phi(c) \neq \phi(c')$
3. $\exists c \in C_I$ and $c' \in C_F$ such that $c \xrightarrow{*} c'$ and $O(c') = -1$
4. $\exists c \in C_I$ and $c' \in C_F$ such that $c \xrightarrow{*} c'$ and $\phi(c) \neq O(c')$
5. $\exists B' \in F_S$ such that $O(B') = -1$
6. $\exists B \in I_S$ and $B' \in F_S$ such that $B \xrightarrow{*} B'$ and $\phi(B) \neq O(B')$ (possibly $B = B'$)

Theorem (Complete Verifier)

Any algorithm checking criteria 1, 5, and 6 decides BPVER.

An Exponential Algorithm for *BPVER*

- Find the reachable portion of the transition graph
  - To find the initial configurations we can use Fenichel's algorithm [1968] for finding the distributions of indistinguishable objects (agents) into distinguisable slots (initial states)
- Partition the induced graph into its strongly connected components by e.g. Tarjan's algorithm [1972]
- Replace each component with a node to obtain a dag
- Initial strongly connected component: contains at least one initial configuration
  - 0-initial: all its initial configurations expect the all-0 output
  - 1-initial: all its initial configurations expect the all-1 output
  - mixed: reject

An Exponential Algorithm for *BPVER*

- Find the reachable portion of the transition graph
  - To find the initial configurations we can use Fenichel's algorithm [1968] for finding the distributions of indistinguishable objects (agents) into distinguisable slots (initial states)
- Partition the induced graph into its strongly connected components by e.g. Tarjan's algorithm [1972]
- Replace each component with a node to obtain a dag
- Initial strongly connected component: contains at least one initial configuration
  - 0-initial: all its initial configurations expect the all-0 output
  - 1-initial: all its initial configurations expect the all-1 output
  - mixed: reject

An Exponential Algorithm for *BPVER*

- Find the reachable portion of the transition graph
  - To find the initial configurations we can use Fenichel's algorithm [1968] for finding the distributions of indistinguishable objects (agents) into distinguisable slots (initial states)
- Partition the induced graph into its strongly connected components by e.g. Tarjan's algorithm [1972]
- Replace each component with a node to obtain a dag
- Initial strongly connected component: contains at least one initial configuration
  - 0-initial: all its initial configurations expect the all-0 output
  - 1-initial: all its initial configurations expect the all-1 output
  - mixed: reject

An Exponential Algorithm for *BPVER*

- Find the reachable portion of the transition graph
  - To find the initial configurations we can use Fenichel's algorithm [1968] for finding the distributions of indistinguishable objects (agents) into distinguisable slots (initial states)
- Partition the induced graph into its strongly connected components by e.g. Tarjan's algorithm [1972]
- Replace each component with a node to obtain a dag
- Initial strongly connected component: contains at least one initial configuration
  - 0-initial: all its initial configurations expect the all-0 output
  - 1-initial: all its initial configurations expect the all-1 output
  - mixed: reject

An Exponential Algorithm for *BPVER*

- Find the reachable portion of the transition graph
  - To find the initial configurations we can use Fenichel's algorithm [1968] for finding the distributions of indistinguishable objects (agents) into distinguisable slots (initial states)
- Partition the induced graph into its strongly connected components by e.g. Tarjan's algorithm [1972]
- Replace each component with a node to obtain a dag
- Initial strongly connected component: contains at least one initial configuration
  - 0-initial: all its initial configurations expect the all-0 output
  - 1-initial: all its initial configurations expect the all-1 output
  - mixed: reject

An Exponential Algorithm for *BPVER*

- Find the reachable portion of the transition graph
  - To find the initial configurations we can use Fenichel's algorithm [1968] for finding the distributions of indistinguishable objects (agents) into distinguisable slots (initial states)
- Partition the induced graph into its strongly connected components by e.g. Tarjan's algorithm [1972]
- Replace each component with a node to obtain a dag
- Initial strongly connected component: contains at least one initial configuration
  - 0-initial: all its initial configurations expect the all-0 output
  - 1-initial: all its initial configurations expect the all-1 output
  - mixed: reject

An Exponential Algorithm for *BPVER*

- Find the reachable portion of the transition graph
  - To find the initial configurations we can use Fenichel's algorithm [1968] for finding the distributions of indistinguishable objects (agents) into distinguisable slots (initial states)
- Partition the induced graph into its strongly connected components by e.g. Tarjan's algorithm [1972]
- Replace each component with a node to obtain a dag
- Initial strongly connected component: contains at least one initial configuration
  - 0-initial: all its initial configurations expect the all-0 output
  - 1-initial: all its initial configurations expect the all-1 output
  - mixed: reject

An Exponential Algorithm for *BPVER*

- Find the reachable portion of the transition graph
  - To find the initial configurations we can use Fenichel's algorithm [1968] for finding the distributions of indistinguishable objects (agents) into distinguisable slots (initial states)
- Partition the induced graph into its strongly connected components by e.g. Tarjan's algorithm [1972]
- Replace each component with a node to obtain a dag
- Initial strongly connected component: contains at least one initial configuration
  - 0-initial: all its initial configurations expect the all-0 output
  - 1-initial: all its initial configurations expect the all-1 output
  - mixed: reject

An Exponential Algorithm for *BPVER*

- Final strongly connected component: has no outgoing edges
  - 0-final: all its configurations give the all-0 output
  - 1-final: all its configurations give the all-1 output
  - mixed: reject
- If there is a directed path from a x-initial component to a $|x-1|$-final component reject, otherwise accept
- An erroneous computation in the original graph begins from an initial configuration c expecting all-x output w.r.t. ϕ and leads to a final strongly connected component which contains a configuration that does not give the all-x output

An Exponential Algorithm for *BPVER*

- Final strongly connected component: has no outgoing edges
  - 0-final: all its configurations give the all-0 output
  - 1-final: all its configurations give the all-1 output
  - mixed: reject
- If there is a directed path from a x-initial component to a $|x-1|$-final component reject, otherwise accept
- An erroneous computation in the original graph begins from an initial configuration c expecting all-x output w.r.t. ϕ and leads to a final strongly connected component which contains a configuration that does not give the all-x output

An Exponential Algorithm for *BPVER*

- Final strongly connected component: has no outgoing edges
  - 0-final: all its configurations give the all-0 output
  - 1-final: all its configurations give the all-1 output
  - mixed: reject
- If there is a directed path from a x-initial component to a $|x-1|$-final component reject, otherwise accept
- An erroneous computation in the original graph begins from an initial configuration c expecting all-x output w.r.t. ϕ and leads to a final strongly connected component which contains a configuration that does not give the all-x output

An Exponential Algorithm for *BPVER*

- Final strongly connected component: has no outgoing edges
  - 0-final: all its configurations give the all-0 output
  - 1-final: all its configurations give the all-1 output
  - mixed: reject
- If there is a directed path from a x-initial component to a $|x-1|$-final component reject, otherwise accept
- An erroneous computation in the original graph begins from an initial configuration c expecting all-x output w.r.t. ϕ and leads to a final strongly connected component which contains a configuration that does not give the all-x output

An Exponential Algorithm for *BPVER*

- Final strongly connected component: has no outgoing edges
  - 0-final: all its configurations give the all-0 output
  - 1-final: all its configurations give the all-1 output
  - mixed: reject
- If there is a directed path from a x-initial component to a $|x-1|$-final component reject, otherwise accept
- An erroneous computation in the original graph begins from an initial configuration c expecting all-x output w.r.t. ϕ and leads to a final strongly connected component which contains a configuration that does not give the all-x output

An Exponential Algorithm for *BPVER*

- Final strongly connected component: has no outgoing edges
  - 0-final: all its configurations give the all-0 output
  - 1-final: all its configurations give the all-1 output
  - mixed: reject
- If there is a directed path from a x-initial component to a $|x-1|$-final component reject, otherwise accept
- An erroneous computation in the original graph begins from an initial configuration c expecting all-x output w.r.t. ϕ and leads to a final strongly connected component which contains a configuration that does not give the all-x output

Conclusions

- Most natural verification problems concerning population protocols are coNP-hard
- There are easily checkable criteria for determining correctness
- Checking any of these defines a possibly non-complete verifier
- Checking 3 of these defines a complete verifier
- All of them are exponential since they are based on searching the transition graph
- We have implemented the first verification tool for population protocols
  - C++
  - Some non-complete and 1 complete verifier
- Experiments show that the running time can be greatly improved in the reject case if the transition graph is constructed on the fly

Conclusions

- Most natural verification problems concerning population protocols are coNP-hard
- There are easily checkable criteria for determining correctness
- Checking any of these defines a possibly non-complete verifier
- Checking 3 of these defines a complete verifier
- All of them are exponential since they are based on searching the transition graph
- We have implemented the first verification tool for population protocols
  - C++
  - Some non-complete and 1 complete verifier
- Experiments show that the running time can be greatly improved in the reject case if the transition graph is constructed on the fly

Conclusions

- Most natural verification problems concerning population protocols are coNP-hard
- There are easily checkable criteria for determining correctness
- Checking any of these defines a possibly non-complete verifier
- Checking 3 of these defines a complete verifier
- All of them are exponential since they are based on searching the transition graph
- We have implemented the first verification tool for population protocols
  - C++
  - Some non-complete and 1 complete verifier
- Experiments show that the running time can be greatly improved in the reject case if the transition graph is constructed on the fly

Conclusions

- Most natural verification problems concerning population protocols are coNP-hard
- There are easily checkable criteria for determining correctness
- Checking any of these defines a possibly non-complete verifier
- Checking 3 of these defines a complete verifier
- All of them are exponential since they are based on searching the transition graph
- We have implemented the first verification tool for population protocols
  - C++
  - Some non-complete and 1 complete verifier
- Experiments show that the running time can be greatly improved in the reject case if the transition graph is constructed on the fly

Conclusions

- Most natural verification problems concerning population protocols are coNP-hard
- There are easily checkable criteria for determining correctness
- Checking any of these defines a possibly non-complete verifier
- Checking 3 of these defines a complete verifier
- All of them are exponential since they are based on searching the transition graph
- We have implemented the first verification tool for population protocols
  - C++
  - Some non-complete and 1 complete verifier
- Experiments show that the running time can be greatly improved in the reject case if the transition graph is constructed on the fly

Conclusions

- Most natural verification problems concerning population protocols are coNP-hard
- There are easily checkable criteria for determining correctness
- Checking any of these defines a possibly non-complete verifier
- Checking 3 of these defines a complete verifier
- All of them are exponential since they are based on searching the transition graph
- We have implemented the first verification tool for population protocols
  - C++
  - Some non-complete and 1 complete verifier
- Experiments show that the running time can be greatly improved in the reject case if the transition graph is constructed on the fly

Conclusions

- Most natural verification problems concerning population protocols are coNP-hard
- There are easily checkable criteria for determining correctness
- Checking any of these defines a possibly non-complete verifier
- Checking 3 of these defines a complete verifier
- All of them are exponential since they are based on searching the transition graph
- We have implemented the first verification tool for population protocols
  - C++
  - Some non-complete and 1 complete verifier
- Experiments show that the running time can be greatly improved in the reject case if the transition graph is constructed on the fly

Conclusions

- Most natural verification problems concerning population protocols are coNP-hard
- There are easily checkable criteria for determining correctness
- Checking any of these defines a possibly non-complete verifier
- Checking 3 of these defines a complete verifier
- All of them are exponential since they are based on searching the transition graph
- We have implemented the first verification tool for population protocols
  - C++
  - Some non-complete and 1 complete verifier
- Experiments show that the running time can be greatly improved in the reject case if the transition graph is constructed on the fly

Conclusions

- Most natural verification problems concerning population protocols are coNP-hard
- There are easily checkable criteria for determining correctness
- Checking any of these defines a possibly non-complete verifier
- Checking 3 of these defines a complete verifier
- All of them are exponential since they are based on searching the transition graph
- We have implemented the first verification tool for population protocols
  - C++
  - Some non-complete and 1 complete verifier
- Experiments show that the running time can be greatly improved in the reject case if the transition graph is constructed on the fly

Some Thoughts

- Can we avoid searching the huge transition graph by somehow looking inside the protocol?
- There is a constructive proof [AADFP '06] that a semilinear predicate is stably computable by the basic model
  - Unfortunately, there exists an unbounded number of correct protocols for the same predicate (we can simply add an unbounded number of dummy states and transitions)
  - e.g. replace $(q_0, q_1) \rightarrow (q_1, q_0)$ with

$$(q_0, q_1) \rightarrow (q_1, q_{d_1})$$
$$(q_{d_1}, q) \rightarrow (q_{d_2}, q)$$
$$\vdots$$
$$(q_{d_t}, q) \rightarrow (q_0, q)$$

Some Thoughts

- Can we avoid searching the huge transition graph by somehow looking inside the protocol?
- There is a constructive proof [AADFP '06] that a semilinear predicate is stably computable by the basic model
  - Unfortunately, there exists an unbounded number of correct protocols for the same predicate (we can simply add an unbounded number of dummy states and transitions)
  - e.g. replace $(q_0, q_1) \rightarrow (q_1, q_0)$ with

$$\begin{gathered}(q_0, q_1) \rightarrow (q_1, q_{d_1}) \\ (q_{d_1}, q) \rightarrow (q_{d_2}, q) \\ \vdots \\ (q_{d_t}, q) \rightarrow (q_0, q)\end{gathered}$$

Some Thoughts

- Can we avoid searching the huge transition graph by somehow looking inside the protocol?
- There is a constructive proof [AADFP '06] that a semilinear predicate is stably computable by the basic model
  - Unfortunately, there exists an unbounded number of correct protocols for the same predicate (we can simply add an unbounded number of dummy states and transitions)
  - e.g. replace $(q_0, q_1) \to (q_1, q_0)$ with

$$
\begin{aligned}
(q_0, q_1) &\to (q_1, q_{d_1}) \\
(q_{d_1}, q) &\to (q_{d_2}, q) \\
&\vdots \\
(q_{d_t}, q) &\to (q_0, q)
\end{aligned}
$$

Some Thoughts

- Can we avoid searching the huge transition graph by somehow looking inside the protocol?
- There is a constructive proof [AADFP '06] that a semilinear predicate is stably computable by the basic model
  - Unfortunately, there exists an unbounded number of correct protocols for the same predicate (we can simply add an unbounded number of dummy states and transitions)
  - e.g. replace $(q_0, q_1) \rightarrow (q_1, q_0)$ with

$$
\begin{aligned}
(q_0, q_1) &\rightarrow (q_1, q_{d_1}) \\
(q_{d_1}, q) &\rightarrow (q_{d_2}, q) \\
&\vdots \\
(q_{d_t}, q) &\rightarrow (q_0, q)
\end{aligned}
$$

Some Thoughts

- Can we somehow compare the default protocol (given by the constructive proof) with the provided one?
  - Possibly by truncating the unnecessary states
- Design verifiers that are not based on transition graph searching
- For noncomplete communication graphs, protocols with stabilizing inputs [AACFJP '05], MPPs [CMS '09] and protocols using non-constant space (PM protocols) [CMNPS '10] we do not have such constructive proofs
- Study verification of protocols in these models
- Model checking

Some Thoughts

- Can we somehow compare the default protocol (given by the constructive proof) with the provided one?
  - Possibly by truncating the unnecessary states
- Design verifiers that are not based on transition graph searching
- For noncomplete communication graphs, protocols with stabilizing inputs [AACFJP '05], MPPs [CMS '09] and protocols using non-constant space (PM protocols) [CMNPS '10] we do not have such constructive proofs
- Study verification of protocols in these models
- Model checking

Some Thoughts

- Can we somehow compare the default protocol (given by the constructive proof) with the provided one?
  - Possibly by truncating the unnecessary states
- Design verifiers that are not based on transition graph searching
- For noncomplete communication graphs, protocols with stabilizing inputs [AACFJP '05], MPPs [CMS '09] and protocols using non-constant space (PM protocols) [CMNPS '10] we do not have such constructive proofs
- Study verification of protocols in these models
- Model checking

Some Thoughts

- Can we somehow compare the default protocol (given by the constructive proof) with the provided one?
  - Possibly by truncating the unnecessary states
- Design verifiers that are not based on transition graph searching
- For noncomplete communication graphs, protocols with stabilizing inputs [AACFJP '05], MPPs [CMS '09] and protocols using non-constant space (PM protocols) [CMNPS '10] we do not have such constructive proofs
- Study verification of protocols in these models
- Model checking

Some Thoughts

- Can we somehow compare the default protocol (given by the constructive proof) with the provided one?
  - Possibly by truncating the unnecessary states
- Design verifiers that are not based on transition graph searching
- For noncomplete communication graphs, protocols with stabilizing inputs [AACFJP '05], MPPs [CMS '09] and protocols using non-constant space (PM protocols) [CMNPS '10] we do not have such constructive proofs
- Study verification of protocols in these models
- Model checking

Some Thoughts

- Can we somehow compare the default protocol (given by the constructive proof) with the provided one?
  - Possibly by truncating the unnecessary states
- Design verifiers that are not based on transition graph searching
- For noncomplete communication graphs, protocols with stabilizing inputs [AACFJP '05], MPPs [CMS '09] and protocols using non-constant space (PM protocols) [CMNPS '10] we do not have such constructive proofs
- Study verification of protocols in these models
- Model checking

FRONTS & VITRO

This work has been partially supported by the ICT Programmes of the European Union under contracts number ICT-2008-215270 (FRONTS) and ICT-2010-257245 (VITRO).

Thank You!