Passively Mobile Communicating Machines that Use Restricted Space

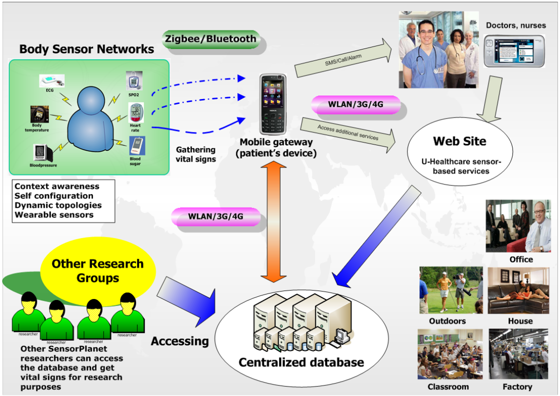
Andreas Pavlogiannis

Joint work with: I. Chatzigiannakis, O. Michail
S. Nikolaou, P. G. Spirakis

Research Academic Computer Technology Institute (RACTI)
Patras, Greece

Talk at FOMC 2011
June 2011

The Motivation



- Wireless Sensor Networks have received great attention recently due to their wide range of applications.

The Background Work

- Theoritical models for WSNs have become significantly important in order to understand their capabilities and limitations.
- Population Protocols [Angluin, Aspnes, Diamadi, Fischer, and Peralta, PODC '04] is a model for WSNs where:
  - Each node: limited computational device $\longrightarrow$ a *finite-state* machine $+$ *sensing* $+$ *communicating* device: **agent**.
  - **Passively mobile** agents: incapable to control or predict.
    - How: unstable environment, like water flow or wind, or the natural mobility of their carriers.
  - Significant properties:
    - **Uniformity:** Protocol descriptions are independent of the population size.
    - **Anonymity:** There is no room in the state of an agent to store a unique identifier.
- Why focus on such a minimalistic model?
  - Real case scenarios: severe restrictions on resources (power, etc).
  - Clearer understanding of the inherent properties and foundations.

The Population Protocols Model and Characteristics

- Agents interact in pairs according to a **communication graph** $G = (V, E)$ where:
  - V: A **population** of $|V| = n$ agents of constant memory (independent of n).
  - E: The permissible interactions between the agents.
- Interaction pattern: **adversary**
- Adversarial choices: **fairness condition**
- **fairness condition**: population partition (the adversary cannot avoid a possible step forever)

Computation

In every execution of a PP:

- Initially: Each agent senses its environment $\longrightarrow$ an input symbol from a finite **input alphabet** X.
  - **input assignment**: tuple specifying an input for each agent.
- the input symbol is mapped by the **input function** $I : X \to Q$ to a state from a finite set of **agent states** Q
  - **population configuration**(C): tuple specifying the state of each agent.
- each state is mapped by the **output function** $O : Q \to Y$ to an output symbol from a finite **output alphabet** Y (agent's output).
- Interaction: **transition function** $\delta : Q \times Q \to Q \times Q \Longrightarrow$ agents update their states according to δ.
  - population configuration(C) changes(C'): goes from C to C' in one step $(C \to C')$.

Stable Computation

- **Computation**: Infinite fair sequence $C_0, C_1, C_2, \ldots$, s.t. $C_i \rightarrow C_{i+1}$ for all i.
- **Population protocols do not halt. They stabilize.**
- **stability**: there is a point/configuration in the computation after which no agent can change its output.
- **stable computation**: regular computation + stabilization

Computational Power

- Due to the minimalistic nature of the model the class of computable predicates is fairly small.
- In [Angluin et al. 2004, 2006] it was proven that it is exactly the class of **semilinear predicates**.
- Formulas such as $N_a \geq 10$ or $N_a < N_b$ capturing scenarios such as the infection of a percentage of a fish population or fire detection by a majority of sensors scattered in a forest.
- This class does not include multiplication, exponentiation and other important operations on input variables.

Relaxing the PP constraints

- **Tiny** (constant) space $\rightarrow$ **Restricted** space
  - Allowing for logarithmic memory is reasonable.
  - 10^9 agents only need $\propto$ 30 bits!
- Preserve passive mobility - no control over the interactions.
  - But still, fair.
- **Passively Mobile Communicating Machines**
- Study space complexity of various problems.
  - Interest remains on problems that use *restricted space*.

Agent

- **Sensor**: Receive the input $x \in X$.
- **Working Tape**: Internal computation.
- **Output Tape**: Agent's output.
- **Outgoing Message Tape**: Send messages to other agents.
- **Incoming Message Tape**: Receive messages from other agents.
- **Working Flag**: When set, the agent is busy doing internal computation and *cannot interact*.

Agent

- **Sensor**: Receive the input $x \in X$.
- **Working Tape**: Internal computation.
- **Output Tape**: Agent's output.
- **Outgoing Message Tape**: Send messages to other agents.
- **Incoming Message Tape**: Receive messages from other agents.
- **Working Flag**: When set, the agent is busy doing internal computation and *cannot interact*.

Agent

- **Sensor**: Receive the input $x \in X$.
- **Working Tape**: Internal computation.
- **Output Tape**: Agent's output.
- **Outgoing Message Tape**: Send messages to other agents.
- **Incoming Message Tape**: Receive messages from other agents.
- **Working Flag**: When set, the agent is busy doing internal computation and *cannot interact*.

Agent

- **Sensor**: Receive the input $x \in X$.
- **Working Tape**: Internal computation.
- **Output Tape**: Agent's output.
- **Outgoing Message Tape**: Send messages to other agents.
- **Incoming Message Tape**: Receive messages from other agents.
- **Working Flag**: When set, the agent is busy doing internal computation and *cannot interact*.

Agent

- **Sensor**: Receive the input $x \in X$.
- **Working Tape**: Internal computation.
- **Output Tape**: Agent's output.
- **Outgoing Message Tape**: Send messages to other agents.
- **Incoming Message Tape**: Receive messages from other agents.
- **Working Flag**: When set, the agent is busy doing internal computation and *cannot interact*.

Agent

- **Sensor**: Receive the input $x \in X$.
- **Working Tape**: Internal computation.
- **Output Tape**: Agent's output.
- **Outgoing Message Tape**: Send messages to other agents.
- **Incoming Message Tape**: Receive messages from other agents.
- **Working Flag**: When set, the agent is busy doing internal computation and *cannot interact*.

Agent

- **Sensor**: Receive the input $x \in X$.
- **Working Tape**: Internal computation.
- **Output Tape**: Agent's output.
- **Outgoing Message Tape**: Send messages to other agents.
- **Incoming Message Tape**: Receive messages from other agents.
- **Working Flag**: When set, the agent is busy doing internal computation and *cannot interact*.

The Passively Mobile Machines Model (*PM*)

Definition

PM protocol: 6-tuple $(X, \Gamma, Q, \delta, \gamma, q_0)$

- $\mathbf{X}$: *input alphabet*, $\sqcup \notin X$,
- $\mathbf{\Gamma}$: *tape alphabet*, $\sqcup \in \Gamma$ and $X \subset \Gamma$,
- $\mathbf{Q}$: set of *states*,
- $\delta : \mathbf{Q} \times \mathbf{\Gamma^4} \rightarrow \mathbf{Q} \times \mathbf{\Gamma^4} \times \{\mathbf{L}, \mathbf{R}, \mathbf{S}\}^4 \times \{\mathbf{0}, \mathbf{1}\}$, the *internal transition function*,
  - Internal computation, Message processing...
- $\gamma : \mathbf{Q} \times \mathbf{Q} \rightarrow \mathbf{Q} \times \mathbf{Q}$, the *external transition function*,
  - Upon interaction, transition to a state that starts reading the incoming message.
- $\mathbf{q_0} \in \mathbf{Q}$, the *initial state*.

Computation in PM

- **Agent Configuration** $B \in \mathcal{B}$: A tuple specifying the agent "state". Configuration yieldability $C \to C'$: C' occurs from C in one step.
- **Population Configuration** $C \in \mathcal{C}$: A tuple capturing the population state.
- Initially, every agent is assigned an *input sybmol*.
- An **Input Function** $I : X \to \mathcal{B}$ specifies the initial configuration for each agent.
- The output of the agent is found in the **output message tape**.
- The adversary chooses:
    - An agent to execute on internal step (application of δ).
    - A pair of agents to interact (message exchange and application of γ)
        - initiator - responder distinction.
- But **fairly**.
    - If $C \to C'$ and C appears infinite times, C' also appears infinite times.

Computation in PM

- **Agent Configuration** $B \in \mathcal{B}$: A tuple specifying the agent "state". Configuration yieldability $C \to C'$: C' occurs from C in one step.
- **Population Configuration** $C \in \mathcal{C}$: A tuple capturing the population state.
- Initially, every agent is assigned an *input sybmol*.
- An **Input Function** $I : X \to \mathcal{B}$ specifies the initial configuration for each agent.
- The output of the agent is found in the **output message tape**.
- The adversary chooses:
  - An agent to execute on internal step (application of δ).
  - A pair of agents to interact (message exchange and application of γ)
    - initiator - responder distinction.
- But **fairly**.
  - If $C \to C'$ and C appears infinite times, C' also appears infinite times.

Computation in PM

- **Agent Configuration** $B \in \mathcal{B}$: A tuple specifying the agent "state". Configuration yieldability $C \to C'$: C' occurs from C in one step.
- **Population Configuration** $C \in \mathcal{C}$: A tuple capturing the population state.
- Initially, every agent is assigned an *input sybmol*.
- An **Input Function** $I : X \to \mathcal{B}$ specifies the initial configuration for each agent.
- The output of the agent is found in the **output message tape**.
- The adversary chooses:
  - An agent to execute on internal step (application of δ).
  - A pair of agents to interact (message exchange and application of γ)
    - initiator - responder distinction.
- But **fairly**.
  - If $C \to C'$ and C appears infinite times, C' also appears infinite times.

Computation in PM

- **Agent Configuration** $B \in \mathcal{B}$: A tuple specifying the agent "state". Configuration yieldability $C \to C'$: C' occurs from C in one step.
- **Population Configuration** $C \in \mathcal{C}$: A tuple capturing the population state.
- Initially, every agent is assigned an *input sybmol*.
- An **Input Function** $I : X \to \mathcal{B}$ specifies the initial configuration for each agent.
- The output of the agent is found in the **output message tape**.
- The adversary chooses:
  - An agent to execute on internal step (application of δ).
  - A pair of agents to interact (message exchange and application of γ)
    - initiator - responder distinction.
- But **fairly**.
  - If $C \to C'$ and C appears infinite times, C' also appears infinite times.

Computation in PM (Continued)

- **Execution**: a sequence of population configurations $(C_1, C_2, \dots)$ such that $C_i \to C_{i+1}$.
- **Computation**: an infinite fair execution.
- PM protocols **stabilize**: $\exists i : \forall v \in V,\ \forall j \geq i$, agent v does not change his output tape in C_j.
- Stable computation of predicates $p : X^{|V|} \to \{0, 1\}$.
  - Symmetric predicates: $p(a) = 1 \iff p(\tilde{a}) = 1$, $\tilde{a}$: permutation of a.
- **Space Complexity Classes**:
  - **PMSPACE(f(n))**: Predicates computable by a PM protocol using $O(f(n))$ space.
  - **SSPACE(f(n))**, **SNSPACE(f(n))**: Symmetric subsets of predicates in $SPACE(f(n))$, $NSPACE(f(n))$.
  - **SEM**: Class of Semilinear predicates.

Computation in PM (Continued)

- **Execution**: a sequence of population configurations $(C_1, C_2, \dots)$ such that $C_i \rightarrow C_{i+1}$.
- **Computation**: an infinite fair execution.
- PM protocols **stabilize**: $\exists i : \forall v \in V,\ \forall j \geq i$, agent v does not change his output tape in C_j.
- Stable computation of predicates $p : X^{|V|} \rightarrow \{0, 1\}$.
    - Symmetric predicates: $p(a) = 1 \iff p(\tilde{a}) = 1$, $\tilde{a}$: permutation of a.
- **Space Complexity Classes**:
    - **PMSPACE(f(n))**: Predicates computable by a PM protocol using $O(f(n))$ space.
    - **SSPACE(f(n))**, **SNSPACE(f(n))**: Symmetric subsets of predicates in $SPACE(f(n))$, $NSPACE(f(n))$.
    - **SEM**: Class of Semilinear predicates.

Computation in PM (Continued)

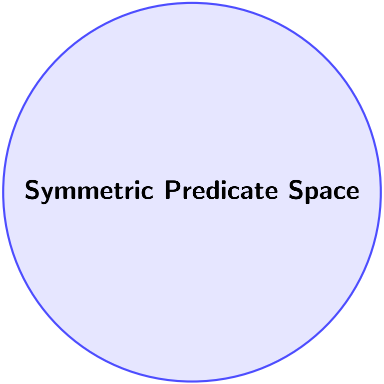
- **Execution**: a sequence of population configurations $(C_1, C_2, \dots)$ such that $C_i \to C_{i+1}$.
- **Computation**: an infinite fair execution.
- PM protocols **stabilize**: $\exists i : \forall v \in V,\ \forall j \geq i$, agent v does not change his output tape in C_j.
- Stable computation of predicates $p : X^{|V|} \to \{0, 1\}$.
  - Symmetric predicates: $p(a) = 1 \iff p(\tilde{a}) = 1$, $\tilde{a}$: permutation of a.
- **Space Complexity Classes**:
  - **PMSPACE(f(n))**: Predicates computable by a PM protocol using $O(f(n))$ space.
  - **SSPACE(f(n))**, **SNSPACE(f(n))**: Symmetric subsets of predicates in $SPACE(f(n))$, $NSPACE(f(n))$.
  - **SEM**: Class of Semilinear predicates.

Computation in PM (Continued)

- **Execution**: a sequence of population configurations $(C_1, C_2, \dots)$ such that $C_i \rightarrow C_{i+1}$.
- **Computation**: an infinite fair execution.
- PM protocols **stabilize**: $\exists i : \forall v \in V,\ \forall j \geq i$, agent v does not change his output tape in C_j.
- Stable computation of predicates $p : X^{|V|} \rightarrow \{0, 1\}$.
  - Symmetric predicates: $p(a) = 1 \iff p(\tilde{a}) = 1$, $\tilde{a}$: permutation of a.
- **Space Complexity Classes**:
  - **PMSPACE(f(n))**: Predicates computable by a PM protocol using $O(f(n))$ space.
  - **SSPACE(f(n))**, **SNSPACE(f(n))**: Symmetric subsets of predicates in $SPACE(f(n))$, $NSPACE(f(n))$.
  - **SEM**: Class of Semilinear predicates.

Dividing the predicate space

- Study of the impact of passive mobility in computational capabilities of distributed systems.



Goal: Divide predicate space according to predicate space complexity.

Assigning Unique Ids

Theorem

Any PM protocol $\mathcal{A}$ can assume the existance of unique ids and knowledge of the population size, at the cost of $O(\log n)$ space.

Proof: A protocol $\mathcal{I}$ for UID assignment.

- All agents start with $uid = 0$.
- **Effective** interactions only between agents with the same *uid*.
  - Initiator increments *uid*.
- **$\mathcal{I}$ does not terminate. Every time a *uid* is incremented, the agent broadcasts a message for $\mathcal{A}$ to reinitiate computation.**
- Agents ignore such messages with *uid* smaller than the last one (ignore *late* messages).
  - After $uid = n - 1$, reinitiations stop, and $\mathcal{A}$ finally is executed correctly.

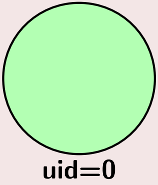
□

Assigning Unique Ids (Continued)

Theorem

Any PM protocol $\mathcal{A}$ can assume the existance of unique ids and knowledge of the population size, at the cost of $O(\log n)$ space.

Proof.

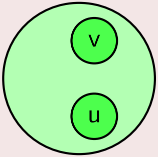


Assigning Unique Ids (Continued)

Theorem

Any PM protocol $\mathcal{A}$ can assume the existance of unique ids and knowledge of the population size, at the cost of $O(\log n)$ space.

Proof.



uid=0

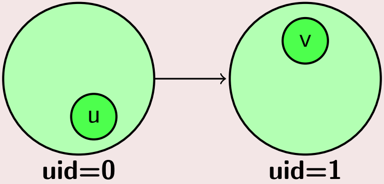
$\square$

Assigning Unique Ids (Continued)

Theorem

Any PM protocol $\mathcal{A}$ can assume the existance of unique ids and knowledge of the population size, at the cost of $O(\log n)$ space.

Proof.



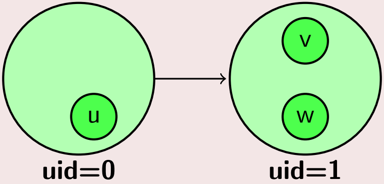
$\square$

Assigning Unique Ids (Continued)

Theorem

Any PM protocol $\mathcal{A}$ can assume the existance of unique ids and knowledge of the population size, at the cost of $O(\log n)$ space.

Proof.



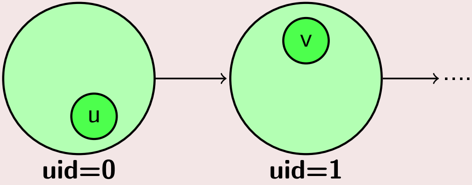
□

Assigning Unique Ids (Continued)

Theorem

Any PM protocol $\mathcal{A}$ can assume the existance of unique ids and knowledge of the population size, at the cost of $O(\log n)$ space.

Proof.



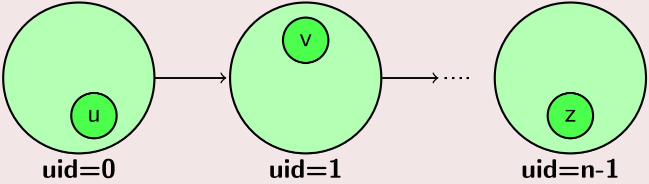
□

Assigning Unique Ids (Continued)

Theorem

Any PM protocol $\mathcal{A}$ can assume the existance of unique ids and knowledge of the population size, at the cost of $O(\log n)$ space.

Proof.

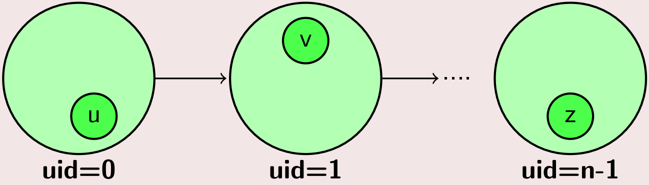


Assigning Unique Ids (Continued)

Theorem

Any PM protocol $\mathcal{A}$ can assume the existance of unique ids and knowledge of the population size, at the cost of $O(\log n)$ space.

Proof.



Still anonymous!

□

Simulating a Deterministic Turing Machine

Theorem

$\mathbf{SSPACE(\Omega(n\log n)) \subseteq PMSPACE(\Omega(\log n))}$

Proof.

Input string $w \in SSPACE(\Omega(\log n))$ decided by a TM D, $|w| = n$.

- Each agent receives a symbol of w.
- Use $\mathcal{I}$ to align all agents.
- Use this alignment as a tape in a **modular** fashion.
  - The **local tape** of each agent provides $O(\log n)$ cells.
- Each time, one **active** agent carries the simulation.
- State transition rules of D embedded in the PM protocol.
- Head move $\rightarrow$ pass **control** + **current state** to neighbor.
- **Simulation accepts a permutation of** w. ✓

Simulating a Deterministic Turing Machine

Theorem

$\mathbf{SSPACE}(\boldsymbol{\Omega}(\mathbf{n}\log\mathbf{n})) \subseteq \mathbf{PMSPACE}(\boldsymbol{\Omega}(\log\mathbf{n}))$

Proof.

Input string $w \in SSPACE(\Omega(\log n))$ decided by a TM D, $|w| = n$.

- Each agent receives a symbol of w.
- Use $\mathcal{I}$ to align all agents.
- Use this alignment as a tape in a **modular** fashion.
  - The **local tape** of each agent provides $O(\log n)$ cells.
- Each time, one **active** agent carries the simulation.
- State transition rules of D embedded in the PM protocol.
- Head move $\rightarrow$ pass **control** + **current state** to neighbor.
- **Simulation accepts a permutation of** w. ✓

□

Simulating a Deterministic Turing Machine

Theorem

$\mathbf{SSPACE}(\boldsymbol{\Omega}(\mathbf{n}\log\mathbf{n})) \subseteq \mathbf{PMSPACE}(\boldsymbol{\Omega}(\log\mathbf{n}))$

Proof.

Input string $w \in SSPACE(\Omega(\log n))$ decided by a TM D, $|w| = n$.

- Each agent receives a symbol of w.
- Use $\mathcal{I}$ to align all agents.
- Use this alignment as a tape in a **modular** fashion.
  - The **local tape** of each agent provides $O(\log n)$ cells.
- Each time, one **active** agent carries the simulation.
- State transition rules of D embedded in the PM protocol.
- Head move $\rightarrow$ pass **control** + **current state** to neighbor.
- **Simulation accepts a permutation of** w. ✓

Allowing for non-determinism

Theorem

$\mathbf{SNSPACE}(\mathbf{\Omega}(\mathbf{n}\log\mathbf{n})) \subseteq \mathbf{PMSPACE}(\mathbf{\Omega}(\log\mathbf{n}))$

Proof.

Input string $w \in SNSPACE(\Omega(\log n))$ decided by a NTM N, $|w| = n$.

- Initial configuration C: all agents set output to **reject**.
- Use simulation of D.
- **Non deterministic** choice out of k possible.
- **Exploit fairness of the adversary!**
  - Pause simulation and wait for interaction.
  - Pick choice based on *uid* of the other participant.
- Simulating branch N **rejects:** Reset population to C.
- N **accepts:** A *good* simulating branch starting from C exists.
  - Simulation keeps reinitiating to C, until that branch is followed.

Allowing for non-determinism

Theorem

$\mathbf{SNSPACE(\Omega(n \log n)) \subseteq PMSPACE(\Omega(\log n))}$

Proof.

Input string $w \in SNSPACE(\Omega(\log n))$ decided by a NTM N, $|w| = n$.

- Initial configuration C: all agents set output to **reject**.
- Use simulation of D.
- **Non deterministic** choice out of k possible.
- **Exploit faireness of the adversary!**
  - Pause simulation and wait for interaction.
  - Pick choice based on *uid* of the other participant.
- Simulating branch N **rejects:** Reset population to C.
- N **accepts:** A *good* simulating branch starting from C exists.
  - Simulation keeps reinitiating to C, until that branch is followed.

Allowing for non-determinism

Theorem

$\mathbf{SNSPACE}(\boldsymbol{\Omega}(\mathbf{n}\log\mathbf{n})) \subseteq \mathbf{PMSPACE}(\boldsymbol{\Omega}(\log\mathbf{n}))$

Proof.

Input string $w \in SNSPACE(\Omega(\log n))$ decided by a NTM N, $|w| = n$.

- Initial configuration C: all agents set output to **reject**.
- Use simulation of D.
- **Non deterministic** choice out of k possible.
- **Exploit faireness of the adversary!**
  - Pause simulation and wait for interaction.
  - Pick choice based on *uid* of the other participant.
- Simulating branch N **rejects:** Reset population to C.
- N **accepts:** A *good* simulating branch starting from C exists.
  - Simulation keeps reinitiating to C, until that branch is followed.

Allowing for non-determinism

Theorem

$\mathbf{SNSPACE(\Omega(n \log n)) \subseteq PMSPACE(\Omega(\log n))}$

Proof.

Input string $w \in SNSPACE(\Omega(\log n))$ decided by a NTM N, $|w| = n$.

- Initial configuration C: all agents set output to **reject**.
- Use simulation of D.
- **Non deterministic** choice out of k possible.
- **Exploit faireness of the adversary!**
  - Pause simulation and wait for interaction.
  - Pick choice based on *uid* of the other participant.
- Simulating branch N **rejects:** Reset population to C.
- N **accepts:** A *good* simulating branch starting from C exists.
  - Simulation keeps reinitiating to C, until that branch is followed.

A Space Hierarchy

Theorem

For $h(n) \in \Omega(\log n)$ and recursive $l(n)$, separated by a nondeterministically fully space constructible function $g(n)$, with $h(n) \in \Omega(g(n))$ but $l(n) \notin \Omega(g(n))$, $\exists$ language in **PMSPACE**$(h(n))$ − **PMSPACE**$(l(n))$.

Proof.

- A unary seperation language has been shown to exist for **NSPACE**.
  - *V. Geffert. Space hierarchy theorem revised.*
- Unary languages are symmetric: **NSPACE** = **SNSPACE**.
- But when $h(n) \in \Omega(\log n) \rightarrow SNSPACE(h(n)) = PMSPACE(h(n))$.

□

A Computational Threshold

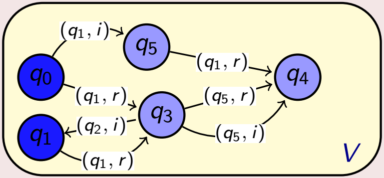
Theorem

Threshold. $\mathbf{PMSPACE}(\mathbf{o}(\log\log \mathbf{n})) = \mathbf{SEM}$.

Proof Idea

Agent Configuration Graph: Describes the effects of interactions of protocol A, but ignores the *deterministic* internal computation.

- Fixed for specific A, V.



A Computational Threshold

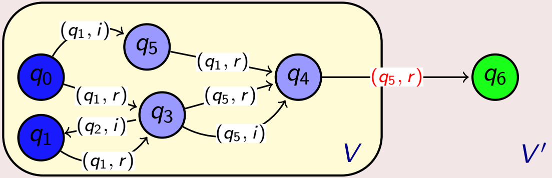
Theorem

Threshold. $\mathbf{PMSPACE}(\mathbf{o}(\log\log\mathbf{n})) = \mathbf{SEM}$.

Proof Idea

Agent Configuration Graph: Describes the effects of interactions of protocol A, but ignores the *deterministic* internal computation.

- Fixed for specific A, V.
- Moving to V', $|V'| > |V|$ adds new configurations k.
  - Accessible through interacting configurations (a, b) existing in V.
  - Since k does not exist in V, a and b cannot exist concurently in V.



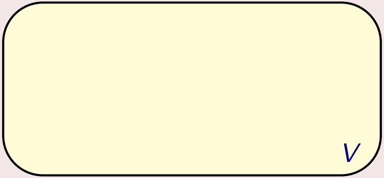
A Computational Threshold (Continued)

Theorem

Threshold. $\mathbf{PMSPACE}(\mathbf{o}(\log\log\mathbf{n})) = \mathbf{SEM}$.

Proof Idea

- **Important Lemma:** When $f(n) = o(\log\log n)$, $\exists V$ such that any configuration can occur in a subpopulation of size $\frac{|V|}{2}$.



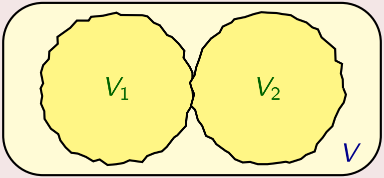
A Computational Threshold (Continued)

Theorem

Threshold. $\mathbf{PMSPACE}(\mathbf{o}(\log\log \mathbf{n})) = \mathbf{SEM}$.

Proof Idea

- **Important Lemma:** When $f(n) = o(\log\log n)$, $\exists V$ such that any configuration can occur in a subpopulation of size $\frac{|V|}{2}$.
- Partition V in V_1, V_2.



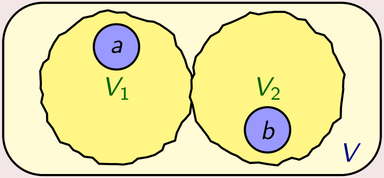
A Computational Threshold (Continued)

Theorem

Threshold. $\mathbf{PMSPACE(o(\log\log n)) = SEM}$.

Proof Idea

- **Important Lemma:** When $f(n) = o(\log\log n)$, $\exists V$ such that any configuration can occur in a subpopulation of size $\frac{|V|}{2}$.
- Partition V in V_1, V_2.
- V_1 creates a, V_2 creates b.



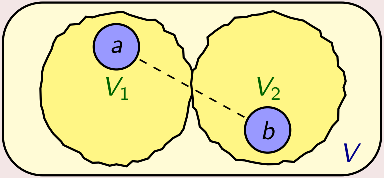
A Computational Threshold (Continued)

Theorem

Threshold. $\mathbf{PMSPACE}(\mathbf{o}(\log\log \mathbf{n})) = \mathbf{SEM}$.

Proof Idea

- **Important Lemma:** When $f(n) = o(\log\log n)$, $\exists V$ such that any configuration can occur in a subpopulation of size $\frac{|V|}{2}$.
- Partition V in V_1, V_2.
- V_1 creates a, V_2 creates b.
- Interaction creates $k \rightarrow k$ not new in V'!



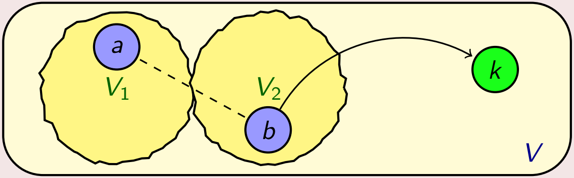
A Computational Threshold (Continued)

Theorem

Threshold. $\mathbf{PMSPACE}(\mathbf{o}(\log\log\mathbf{n})) = \mathbf{SEM}$.

Proof Idea

- **Important Lemma:** When $f(n) = o(\log\log n)$, $\exists V$ such that any configuration can occur in a subpopulation of size $\frac{|V|}{2}$.
- Partition V in V_1, V_2.
- V_1 creates a, V_2 creates b.
- Interaction creates $k \rightarrow k$ not new in V'!



No new states in V'!.

Power of 2 predicate

Theorem

Predicate $p: \ \log N_a = t$, *for some* t *is in* $PMSPACE(\log \log n)$.

Proof.

A PM protocol A computing p in $O(\log \log n)$ space.

- Agent v that received an a sets $x_v = 1$, otherwise $x_v = 0$.
- Agents u and v interact only if $x_u = x_v \neq 0$.
  - $x_u = x_u + 1$, $x_v = 0$.
- In parallel, a PP B checks whether $\exists\ u, v : x_u, x_v \geq 1$.
  - If so, set output to 0, otherwise 1.
- B runs on **stabilizing inputs**.
  - *D. Angluin, J. Aspnes, and D. Eisenstat. Stably computable predicates are semilinear.*

Power of 2 predicate

Theorem

Predicate p : $\log N_a = t$*, for some t is in PMSPACE*$(\log \log n)$*.*

Proof.

A PM protocol A computing p in $O(\log \log n)$ space.

- Agent v that received an a sets $x_v = 1$, otherwise $x_v = 0$.
- Agents u and v interact only if $x_u = x_v \neq 0$.
  - $x_u = x_u + 1$, $x_v = 0$.
- In parallel, a PP B checks whether $\exists\ u, v : x_u, x_v \geq 1$.
  - If so, set output to 0, otherwise 1.
- B runs on **stabilizing inputs**.
  - *D. Angluin, J. Aspnes, and D. Eisenstat. Stably computable predicates are semilinear.*

Power of 2 predicate

Theorem

Predicate p : $\log N_a = t$, for some t is in PMSPACE$(\log \log n)$.

Proof.

A PM protocol A computing p in $O(\log \log n)$ space.

- Agent v that received an a sets $x_v = 1$, otherwise $x_v = 0$.
- Agents u and v interact only if $x_u = x_v \neq 0$.
  - $x_u = x_u + 1$, $x_v = 0$.
- In parallel, a PP B checks whether $\exists\ u, v : x_u, x_v \geq 1$.
  - If so, set output to 0, otherwise 1.
- B runs on **stabilizing inputs**.
  - *D. Angluin, J. Aspnes, and D. Eisenstat. Stably computable predicates are semilinear.*

Power of 2 predicate

Theorem

Predicate p : $\log N_a = t$, for some t is in PMSPACE$(\log\log n)$.

Proof.

A PM protocol A computing p in $O(\log\log n)$ space.

- Agent v that received an a sets $x_v = 1$, otherwise $x_v = 0$.
- Agents u and v interact only if $x_u = x_v \neq 0$.
  - $x_u = x_u + 1$, $x_v = 0$.
- In parallel, a PP B checks whether $\exists\ u, v : x_u, x_v \geq 1$.
  - If so, set output to 0, otherwise 1.
- B runs on **stabilizing inputs**.
  - *D. Angluin, J. Aspnes, and D. Eisenstat. Stably computable predicates are semilinear.*

□

Power of 2 predicate

Theorem

Predicate $p: \log N_a = t$, for some t is in $PMSPACE(\log\log n)$.

Proof.

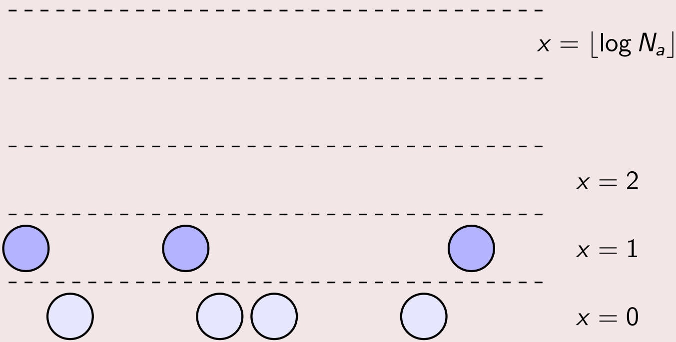
A PM protocol A computing p in $O(\log\log n)$ space.

- Agent v that received an a sets $x_v = 1$, otherwise $x_v = 0$.
- Agents u and v interact only if $x_u = x_v \neq 0$.
  - $x_u = x_u + 1$, $x_v = 0$.
- In parallel, a PP B checks whether $\exists\ u, v : x_u, x_v \geq 1$.
  - If so, set output to 0, otherwise 1.
- B runs on **stabilizing inputs**. ✓
  - *D. Angluin, J. Aspnes, and D. Eisenstat. Stably computable predicates are semilinear.*

□

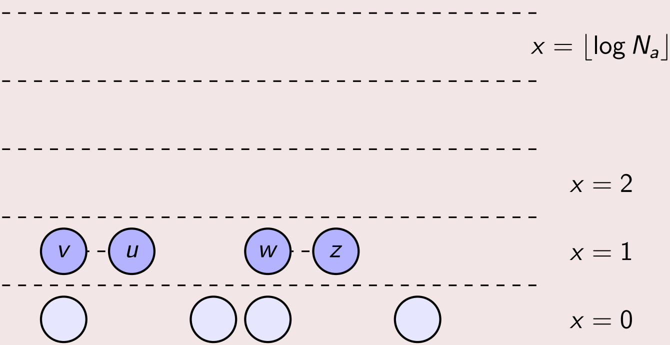
Proof.

- Whenever $x_v = x_v + 1$ for some v, there are at least 2^{x_v+1} a's in the population.
- $x_v \neq 0$ **for only one** $v \iff 2^{x_v+1}$.
- $\mathbf{Max}(x_v) = \log N_a \leq \log n \implies O(\log\log n)$ **space**.



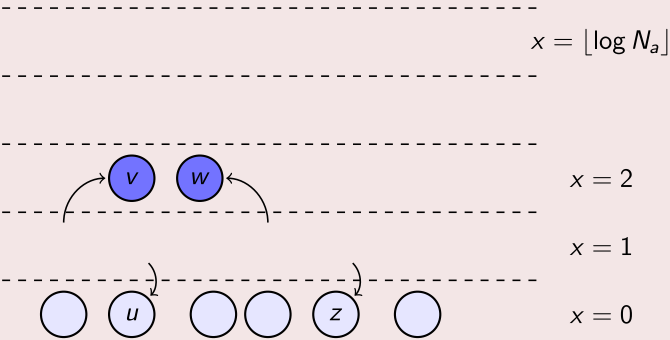
Proof.

- Whenever $x_v = x_v + 1$ for some v, there are at least 2^{x_v+1} a's in the population.
- $x_v \neq 0$ **for only one** $v \iff 2^{x_v+1}$.
- **Max**$(x_v) = \log N_a \leq \log n \implies O(\log\log n)$ **space**.



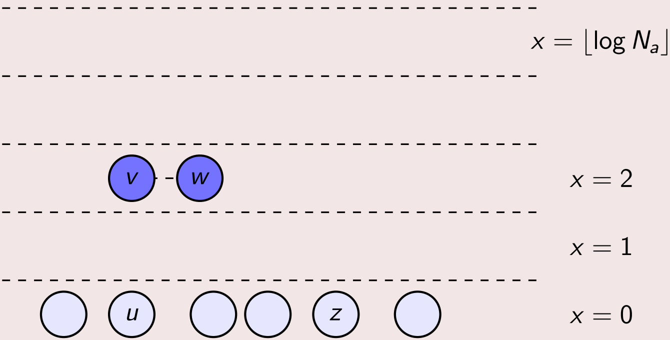
Proof.

- Whenever $x_v = x_v + 1$ for some v, there are at least 2^{x_v+1} a's in the population.
- $x_v \neq 0$ **for only one** $v \iff 2^{x_v+1}$.
- $\mathbf{Max}(x_v) = \log N_a \leq \log n \implies O(\log\log n)$ **space**.



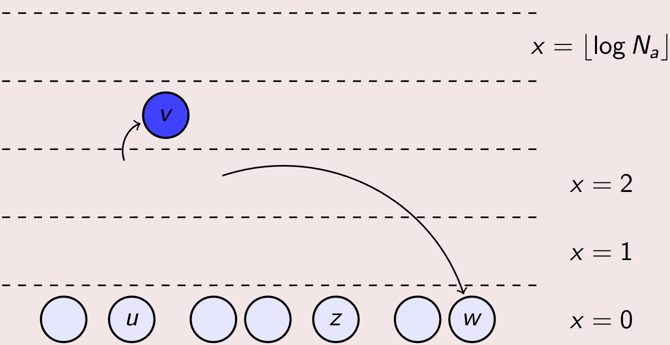
Proof.

- Whenever $x_v = x_v + 1$ for some v, there are at least 2^{x_v+1} a's in the population.
- $x_v \neq 0$ **for only one** $v \iff 2^{x_v+1}$.
- **Max**$(x_v) = \log N_a \leq \log n \implies O(\log \log n)$ **space**.



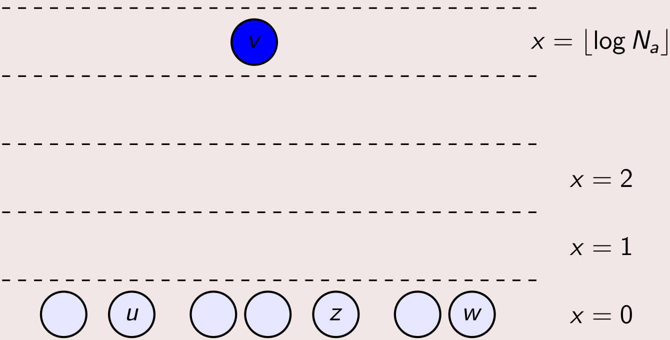
Proof.

- Whenever $x_v = x_v + 1$ for some v, there are at least 2^{x_v+1} a's in the population.
- $x_v \neq 0$ **for only one** $v \iff 2^{x_v+1}$.
- **Max**$(x_v) = \log N_a \leq \log n \implies O(\log\log n)$ **space**.



Proof.

- Whenever $x_v = x_v + 1$ for some v, there are at least 2^{x_v+1} a's in the population.
- $x_v \neq 0$ **for only one** $v \iff 2^{x_v+1}$.
- **Max**$(x_v) = \log N_a \leq \log n \implies O(\log \log n)$ **space**.



Conclusions - Further Research

- **Our contribution:**
  - We have presented a new model to study passive mobility in interaction-based, distributed, anonymous systems.
  - We have given a space hierarchy for functions $\Omega(\log n)$.
  - We have proved an interesting threshold in $o(\log \log n)$.
    - Tight.
- **Further research:**
  - Computational characterization between $\log \log n$ and $\log n$.
  - Fault tolerance.
  - Probabilistic assumptions & time complexity.
  - Adversarial perspective.

Thank You!

SEVENTH FRAMEWORK PROGRAMME