



# Outline

- Classical Population Protocols (Finite Populations)
  - Motivation
  - Some Algorithms
  - Computational Power
  
- Population Protocols with Huge Populations
  - Computing Algebraic Irrational Values : an unformal approach.
  - Computing Algebraic Irrational Values : a formal study.
  - Composing Protocols
  - Discussions
  
- Discussions and Conclusions

# Sources

- Classical Population Protocols :
  - Slides from talk “Population Protocols”, by Eric Ruppert (York University and EPFL), at MiNEMA Winter School February 15, 2007
  - itself based on the work of
    - Dana Angluin, James Aspnes, Melody Chan, Carole Delporte-Gallet, Zoë Diamadi, David Eisenstat, Michael J. Fischer, Hugues Fauconnier, Rachid Guerraoui, Hong Jiang, René Peralta, Eric Ruppert
- Population Protocols with Huge Populations :
  - **Work in Progress**
  - Based on the work of
    - Philippe Chassaing, Johanne Cohen, Pierre-Fraigniaud, Lucas Gerin, **Xavier Koegler**.

## Classical Population Protocols : Motivation

# Models of Mobile Computing

In mobile computing (and distributed computing in general), computability depends crucially on the model's parameters.

It is important to understand the effect of each model assumption

- in isolation, and
- in relation to other assumptions.

# Identifying Assumptions

What sorts of assumptions do people make about mobile systems?

- Computational sophistication of agents (robots carrying big hard drives vs nanotechnology)
- Infrastructure (e.g. fixed beacons, unique ids)
- Synchrony
- Communication range
- Mobility patterns (speed restrictions, probabilistic movement)
- Battery power
- Failure patterns

# Towards Population Protocols

A model of sensor networks with absolutely minimal assumptions about

- sophistication of mobile units : finite state machines
- infrastructure : none (not even unique ids)
- synchrony : totally asynchronous
- communication range : big enough that communication is occasionally possible between two agents.

## A Motivating Example : Birds

- Strap tiny, identical sensors to many birds in a flock.
- Sensors on two birds can interact when the birds are close together.
- Can we program sensors in order to detect when (at least) five birds have elevated body temperatures, indicating possible epidemic?
- Can we program sensors in order to detect when (at least) five percent of birds have elevated body temperatures, indicating possible epidemic?

## A Motivating Example : Birds

- Strap tiny, identical sensors to many birds in a flock.
- Sensors on two birds can interact when the birds are close together.
- Can we program sensors in order to detect when (at least) five birds have elevated body temperatures, indicating possible epidemic?
- Can we program sensors in order to detect when (at least) five percent of birds have elevated body temperatures, indicating possible epidemic?

## A Motivating Example : Birds

- Strap tiny, identical sensors to many birds in a flock.
- Sensors on two birds can interact when the birds are close together.
- Can we program sensors in order to detect when (at least) five birds have elevated body temperatures, indicating possible epidemic?
- Can we program sensors in order to detect when (at least) five percent of birds have elevated body temperatures, indicating possible epidemic?

# The Model (Informally)

Collection of **finitely many** identically programmed finite state machines.

When two get close together, they can interact and update their own states.

Some fairness hypotheses guarantee that all possible interactions happen eventually (captures the idea of probabilistic adversary : there is some (unknown) underlying probability distribution on interactions such that events are independent).

## How to Compute ?

Encode inputs in initial states of the agents. Interpret states of agents as an output.

For any input  $I$  , for any execution starting from  $I$  agents converge to the correct output for  $I$  .

# What is an Algorithm ?

An algorithm consists of

- a finite set of states
- transition rules, mapping pairs of states to pairs of states
- input encoding : mapping from inputs to multisets of states
- output interpretation : mapping from multiset of states to output.

Remark : Algorithm is independent of size of population !

# Simplest Example : Computing OR of Input Bits

States : 0, 1.

One transition rule :  $0, 1 \rightarrow 1, 1$ .

Output of an agent is its state.

If all inputs are 0, all agents will remain in state 0.

If some agent has input 1, eventually all will have state 1.

◀ Back

## Classical Population Protocols : Some Algorithms

# More Algorithms

- Computing OR of Input Bits
- Threshold Predicate
- Majority
- 5 %
- Arithmetic Operations
- Leader Election

## Classical Population Protocols : Computational Power

# Predicates

A predicate has yes/no output.

Assume every agent should eventually produce correct output

Predicate must be **symmetric** (order of inputs is unimportant).

So, we can write predicate as  $P(x_1, x_2, \dots, x_k)$  where

$k$  = number of possible initial states,

$x_i$  = number of agents starting in  $i$ th state.

# Computable Predicates

## Theorem

*A predicate is computable iff it is on the following list.*

- $\sum_{i=1}^k c_i x_i \geq a$ , where  $a, c_i$ 's are integer constants
- $\sum_{i=1}^k c_i x_i \equiv a \pmod{b}$  where  $a, b$  and  $c_i$ 's are constants
- Boolean combinations of the above predicates

▶ Bonus

## Alternate Characterization : Presburger Arithmetic

A predicate is computable iff it can be expressed in first-order logic using the symbols  $+$ ,  $0$ ,  $1$ ,  $\forall$ ,  $\wedge$ ,  $\neg$ ,  $\exists$ ,  $=$ ,  $<$ ,  $(, )$  and variables.

(This system is known as Presburger Arithmetic [1929].)

Note : no multiplication.

### Examples :

majority :  $x_1 < x_2$

divisible by 3 :  $\exists y : y + y + y = x_1$

at least 40% 1's :  $\neg(x_1 + x_1 < x_0 + x_0 + x_0)$

## Alternate Characterization : Semilinear Sets

A predicate is computable iff it the set of inputs with output yes is semilinear.

A set of vectors  $\vec{x} = (x_1, x_2, \dots, x_k) \in \mathbb{N}^k$  is **linear** if it is of the form  $\{\vec{v}_0 + c_1\vec{v}_1 + c_2\vec{v}_2 + \dots + c_m\vec{v}_m : c_1, \dots, c_m \in \mathbb{N}\}$

A set of vectors is **semilinear** if it is a finite union of linear sets.

◀ Picture

# What Do We Know ?

We know exactly which predicates are computable in the classical basic model.

- They are boolean combinations of threshold and mod predicates.
- They are semilinear.
- They are expressible in Presburger arithmetic.

Note : Authorizing probabilities on right hand sides

Ex :  $+ - \rightarrow 1/2+$ ;  $1/2 : -$   
doesn't change their power.

# Variants of the (Finite Population) Model

The basic classical model is now already understood pretty well.

Variants considered in literature :

- Limited interaction graph
- One-way communication
- Failures

## Population Protocols with Huge Populations

# Population Protocols with Huge Populations

Inspired by Evolutionary Game Theory,

considering rules of interactions as games,

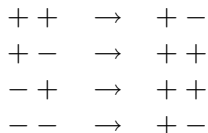
we want to consider

- a huge population of agents (instead of finitely many agents)
- proportions of agents in a given state (instead of counts)
- protocols as computing functions (not only predicates)

# An Example

States :  $Q = \{+, -\}$ .

Rules of interactions :



What can we say about

$$p_+ = \frac{\text{number of } +}{\text{number of } ++ + \text{number of } -}?$$

## When the number $n$ of agents is fixed.

- The interaction rules describe an homogeneous Markov chain with  $2^n$  states.
- Special configuration :  $(-, -, \dots, -)$  is left in one step.
- Any other configuration is reachable with positive probability from any other configuration.

+	+	→	+	-
+	-	→	+	+
-	+	→	+	+
-	-	→	+	-

## When the number $n$ of agents is fixed : continued

- By ergodic theorem, whatever the initial distribution is, the distribution of  $+$ , and  $-$  will converge
- to the unique stationary distribution of the Markov chain,
- and hence,

$p_+$  will converge to some (computable) rational value.

When the number  $n$  of agents is huge?

++	→	+-
+-	→	++
-+	→	++
--	→	+-

- What is the limit of  $p_+$  when  $n$  goes to infinity?

## Unformal Approach

# An Unformal Approach

$$\begin{array}{lcl} ++ & \rightarrow & +- \\ +- & \rightarrow & ++ \\ -+ & \rightarrow & ++ \\ -- & \rightarrow & +- \end{array}$$

- The mean number of + created,

$$-1 * p_+^2 + 1 * p_+(1-p_+) + 1 * p_+(1-p_+) + 1 * (1-p_+)^2 = 1 - 2p_+^2$$

- must be equal, at the limit to  $p_+$ ,
- and hence

$$p_+ = \frac{\sqrt{2}}{2},$$

at the limit.

# An Unformal Approach

$$\begin{array}{lcl} + + & \rightarrow & + - \\ + - & \rightarrow & + + \\ - + & \rightarrow & + + \\ - - & \rightarrow & + - \end{array}$$

- The mean number of + created,

$$-1 * p_+^2 + 1 * p_+(1 - p_+) + 1 * p_+(1 - p_+) + 1 * (1 - p_+)^2 = 1 - 2p_+^2$$

- must be equal, at the limit to  $p_+$ ,
- and hence

$$p_+ = \frac{\sqrt{2}}{2},$$

at the limit.

# An Unformal Approach

$$\begin{array}{lcl} ++ & \rightarrow & +- \\ +- & \rightarrow & ++ \\ -+ & \rightarrow & ++ \\ -- & \rightarrow & +- \end{array}$$

- The mean number of + created,

$$-1 * p_+^2 + 1 * p_+(1 - p_+) + 1 * p_+(1 - p_+) + 1 * (1 - p_+)^2 = 1 - 2p_+^2$$

- must be equal, at the limit to  $p_+$ ,
- and hence

$$p_+ = \frac{\sqrt{2}}{2},$$

at the limit.

# An Unformal Approach

$$\begin{array}{lcl} + + & \rightarrow & + - \\ + - & \rightarrow & + + \\ - + & \rightarrow & + + \\ - - & \rightarrow & + - \end{array}$$

- The mean number of + created,

$$-1 * p_+^2 + 1 * p_+(1 - p_+) + 1 * p_+(1 - p_+) + 1 * (1 - p_+)^2 = 1 - 2p_+^2$$

- must be equal, at the limit to  $p_+$ ,
- and hence

$$p_+ = \frac{\sqrt{2}}{2},$$

at the limit.

# An Unformal Approach

$$\begin{array}{lcl} + + & \rightarrow & + - \\ + - & \rightarrow & + + \\ - + & \rightarrow & + + \\ - - & \rightarrow & + - \end{array}$$

- The mean number of + created,

$$-1 * p_+^2 + 1 * p_+(1 - p_+) + 1 * p_+(1 - p_+) + 1 * (1 - p_+)^2 = 1 - 2p_+^2$$

- must be equal, at the limit to  $p_+$ ,
- and hence

$$p_+ = \frac{\sqrt{2}}{2},$$

at the limit.

## Formal Approach

## A Formal Approach : convergence

### Theorem

*We have for all  $t$ ,*

$$p(\lfloor nt \rfloor) = \frac{\sqrt{2}}{2} + Z_n(t),$$

*where  $Z_n(t)$  converges in law when  $n$  goes to infinity to the (deterministic) solution of ordinary differential*

$$dX(t) = (1 - 2X^2)dt.$$

*Solutions of this ordinary differential equation go to 0 at infinity.*

# A Formal Approach : Asymptotic Development

## Theorem

*We have for all  $t$ ,*

$$p(\lfloor nt \rfloor) = \frac{\sqrt{2}}{2} + \frac{1}{\sqrt{n}} A_n(t),$$

*where  $A_n(t)$  converges in law to the unique solution of stochastic differential equation (Orstein-Uhlenbeck process)*

$$dX(t) = -2\sqrt{2}X(t)dt + dB(t),$$

*and hence to the Gaussian  $\mathcal{N}(0, \frac{\sqrt{2}}{8})$  when  $t$  goes to infinity.*

## Computational Power?

# Computing With Huge Populations

- In other words, this protocol **computes** real number  $\frac{\sqrt{2}}{2}$ .
- Which real numbers can be computed in the above sense?

# Computing with Huge Populations

Which real numbers can be computed in the above sense?

- Computable real numbers belongs to  $[0, 1]$ .
- Computable real numbers must be algebraic.
- **Can all algebraic real of  $[0, 1]$  be computed?**

# Computing with Huge Populations

Which real numbers can be computed in the above sense?

- Computable real numbers belongs to  $[0, 1]$ .
- Computable real numbers must be algebraic.
- Can all algebraic real of  $[0, 1]$  be computed?

# Computing with Huge Populations

Which real numbers can be computed in the above sense?

- Computable real numbers belongs to  $[0, 1]$ .
- Computable real numbers must be algebraic.
- Can all algebraic real of  $[0, 1]$  be computed?

# Computing with Huge Populations

Which real numbers can be computed in the above sense?

- Computable real numbers belongs to  $[0, 1]$ .
- Computable real numbers must be algebraic.
- **Can all algebraic real of  $[0, 1]$  be computed?**

# Some Closure Properties : Square Root

## Theorem

*From a protocol computing  $p_2$ , one can build a protocol computing  $\sqrt{p_2}$ .*

$(1, 1)(., 1)$	$\rightarrow$	$(1, 1)$
$(1, 1)(., 0)$	$\rightarrow$	$3/4(1, 1); 1/4(0, 1)$
$(1, 0)(0, 0)$	$\rightarrow$	$1/2(1, 0); 1/2(0, 0)$
$(1, 0)(0, 1)$	$\rightarrow$	$1/4(1, 0); 3/4(0, 0)$
$(1, 0)(1, 0)$	$\rightarrow$	$1/4(1, 0); 3/4(0, 0)$
$(1, 0)(1, 1)$	$\rightarrow$	$(1, 0)$
$(0, 1)(1, .)$	$\rightarrow$	$1/2(1, 1); 1/2(0, 1)$
$(0, 1)(0, .)$	$\rightarrow$	$1/4(1, 1); 3/4(0, 1)$
$(0, 0)(1, .)$	$\rightarrow$	$1/4(1, 1); 3/4(0, 1)$
$(0, 0)(0, .)$	$\rightarrow$	$(0, 0)$

$$p_1^{n+1} - p_1^n = -1/4((p_1^n)^2 - p_2)$$

## Some Closure Properties : Square Root

### Theorem

*From a protocol computing  $p_2$ , one can build a protocol computing  $\sqrt{p_2}$ .*

$(1, 1)(., 1)$	$\rightarrow$	$(1, 1)$
$(1, 1)(., 0)$	$\rightarrow$	$3/4(1, 1); 1/4(0, 1)$
$(1, 0)(0, 0)$	$\rightarrow$	$1/2(1, 0); 1/2(0, 0)$
$(1, 0)(0, 1)$	$\rightarrow$	$1/4(1, 0); 3/4(0, 0)$
$(1, 0)(1, 0)$	$\rightarrow$	$1/4(1, 0); 3/4(0, 0)$
$(1, 0)(1, 1)$	$\rightarrow$	$(1, 0)$
$(0, 1)(1, .)$	$\rightarrow$	$1/2(1, 1); 1/2(0, 1)$
$(0, 1)(0, .)$	$\rightarrow$	$1/4(1, 1); 3/4(0, 1)$
$(0, 0)(1, .)$	$\rightarrow$	$1/4(1, 1); 3/4(0, 1)$
$(0, 0)(0, .)$	$\rightarrow$	$(0, 0)$

$$p_1^{n+1} - p_1^n = -1/4((p_1^n)^2 - p_2)$$

# Some Closure Properties : Square Root

## Theorem

*From a protocol computing  $p_2$ , one can build a protocol computing  $\sqrt{p_2}$ .*

$(1, 1)(., 1)$	$\rightarrow$	$(1, 1)$
$(1, 1)(., 0)$	$\rightarrow$	$3/4(1, 1); 1/4(0, 1)$
$(1, 0)(0, 0)$	$\rightarrow$	$1/2(1, 0); 1/2(0, 0)$
$(1, 0)(0, 1)$	$\rightarrow$	$1/4(1, 0); 3/4(0, 0)$
$(1, 0)(1, 0)$	$\rightarrow$	$1/4(1, 0); 3/4(0, 0)$
$(1, 0)(1, 1)$	$\rightarrow$	$(1, 0)$
$(0, 1)(1, .)$	$\rightarrow$	$1/2(1, 1); 1/2(0, 1)$
$(0, 1)(0, .)$	$\rightarrow$	$1/4(1, 1); 3/4(0, 1)$
$(0, 0)(1, .)$	$\rightarrow$	$1/4(1, 1); 3/4(0, 1)$
$(0, 0)(0, .)$	$\rightarrow$	$(0, 0)$

$$p_1^{n+1} - p_1^n = -1/4((p_1^n)^2 - p_2)$$

# Some Closure Properties : Products

## Theorem

*From a protocol computing  $p_2$ , and a protocol computing  $p_3$ , one can build a protocol computing  $p_2p_3$ .*

$$(\cdot, 1, 1)(\cdot, \cdot, 1) \rightarrow (1, 1, 1)$$

$$(\cdot, 1, 0)(\cdot, \cdot, 1) \rightarrow (1, 1, 0)$$

$$(\cdot, 1, 1)(\cdot, \cdot, 0) \rightarrow (0, 1, 1)$$

$$(\cdot, 1, 0)(\cdot, \cdot, 0) \rightarrow (0, 1, 0)$$

$$(\cdot, 0, 1)(\cdot, \cdot, \cdot) \rightarrow (0, 0, 1)$$

$$(\cdot, 0, 0)(\cdot, \cdot, \cdot) \rightarrow (0, 0, 0)$$

$$p_{(1,\dots)}^{n+1} - p_{(1,\dots)}^n = (p_2p_3) - p_1^n$$

# Some Closure Properties : Products

## Theorem

*From a protocol computing  $p_2$ , and a protocol computing  $p_3$ , one can build a protocol computing  $p_2p_3$ .*

$$(\cdot, 1, 1)(\cdot, \cdot, 1) \rightarrow (1, 1, 1)$$

$$(\cdot, 1, 0)(\cdot, \cdot, 1) \rightarrow (1, 1, 0)$$

$$(\cdot, 1, 1)(\cdot, \cdot, 0) \rightarrow (0, 1, 1)$$

$$(\cdot, 1, 0)(\cdot, \cdot, 0) \rightarrow (0, 1, 0)$$

$$(\cdot, 0, 1)(\cdot, \cdot, \cdot) \rightarrow (0, 0, 1)$$

$$(\cdot, 0, 0)(\cdot, \cdot, \cdot) \rightarrow (0, 0, 0)$$

$$p_{(1, \dots)}^{n+1} - p_{(1, \dots)}^n = (p_2 p_3) - p_1^n$$

# Some Closure Properties : Products

## Theorem

*From a protocol computing  $p_2$ , and a protocol computing  $p_3$ , one can build a protocol computing  $p_2p_3$ .*

$$(\cdot, 1, 1)(\cdot, \cdot, 1) \rightarrow (1, 1, 1)$$

$$(\cdot, 1, 0)(\cdot, \cdot, 1) \rightarrow (1, 1, 0)$$

$$(\cdot, 1, 1)(\cdot, \cdot, 0) \rightarrow (0, 1, 1)$$

$$(\cdot, 1, 0)(\cdot, \cdot, 0) \rightarrow (0, 1, 0)$$

$$(\cdot, 0, 1)(\cdot, \cdot, \cdot) \rightarrow (0, 0, 1)$$

$$(\cdot, 0, 0)(\cdot, \cdot, \cdot) \rightarrow (0, 0, 0)$$

$$p_{(1,\dots)}^{n+1} - p_{(1,\dots)}^n = (p_2p_3) - p_1^n$$

# Some Closure Properties : Half Sum

## Theorem

*From a protocol computing  $p_2$ , and a protocol computing  $p_3$ , one can build a protocol computing  $(p_2 + p_3)/2$ .*

$$\begin{aligned}(1, 1, 1)(., ., .) &\rightarrow (1, 1, 1) \\(0, 1, 1)(., ., .) &\rightarrow (1, 1, 1) \\(1, 0, 1)(., ., .) &\rightarrow 1/2(1, 0, 1); 1/2(0, 0, 1) \\(0, 0, 1)(., ., .) &\rightarrow 1/2(1, 0, 1); 1/2(0, 0, 1) \\(1, 1, 0)(., ., .) &\rightarrow 1/2(1, 1, 0); 1/2(0, 1, 0) \\(0, 1, 0)(., ., .) &\rightarrow 1/2(1, 1, 0); 1/2(0, 1, 0) \\(1, 0, 0)(., ., .) &\rightarrow (0, 0, 0) \\(0, 0, 0)(., ., .) &\rightarrow (0, 0, 0)\end{aligned}$$

$$p_1^{n+1} - p_1^n = 1/2(p_2^n + p_3^n) - p_1^n$$

# Some Closure Properties : Half Sum

## Theorem

*From a protocol computing  $p_2$ , and a protocol computing  $p_3$ , one can build a protocol computing  $(p_2 + p_3)/2$ .*

$$\begin{aligned}(1, 1, 1)(., ., .) &\rightarrow (1, 1, 1) \\(0, 1, 1)(., ., .) &\rightarrow (1, 1, 1) \\(1, 0, 1)(., ., .) &\rightarrow 1/2(1, 0, 1); 1/2(0, 0, 1) \\(0, 0, 1)(., ., .) &\rightarrow 1/2(1, 0, 1); 1/2(0, 0, 1) \\(1, 1, 0)(., ., .) &\rightarrow 1/2(1, 1, 0); 1/2(0, 1, 0) \\(0, 1, 0)(., ., .) &\rightarrow 1/2(1, 1, 0); 1/2(0, 1, 0) \\(1, 0, 0)(., ., .) &\rightarrow (0, 0, 0) \\(0, 0, 0)(., ., .) &\rightarrow (0, 0, 0)\end{aligned}$$

$$p_1^{n+1} - p_1^n = 1/2(p_2^n + p_3^n) - p_1^n$$

# Some Closure Properties : Half Sum

## Theorem

*From a protocol computing  $p_2$ , and a protocol computing  $p_3$ , one can build a protocol computing  $(p_2 + p_3)/2$ .*

$(1, 1, 1)(., ., .)$	$\rightarrow$	$(1, 1, 1)$
$(0, 1, 1)(., ., .)$	$\rightarrow$	$(1, 1, 1)$
$(1, 0, 1)(., ., .)$	$\rightarrow$	$1/2(1, 0, 1); 1/2(0, 0, 1)$
$(0, 0, 1)(., ., .)$	$\rightarrow$	$1/2(1, 0, 1); 1/2(0, 0, 1)$
$(1, 1, 0)(., ., .)$	$\rightarrow$	$1/2(1, 1, 0); 1/2(0, 1, 0)$
$(0, 1, 0)(., ., .)$	$\rightarrow$	$1/2(1, 1, 0); 1/2(0, 1, 0)$
$(1, 0, 0)(., ., .)$	$\rightarrow$	$(0, 0, 0)$
$(0, 0, 0)(., ., .)$	$\rightarrow$	$(0, 0, 0)$

$$p_1^{n+1} - p_1^n = 1/2(p_2^n + p_3^n) - p_1^n$$

# Discussions

For now, we don't have a precise characterization of the computable real numbers.

We know that :

- they are algebraic
- they contain rational numbers
- they are closed by square root
- they are closed by product
- they are closed by half sum

# Current and Future Work

- Characterize precisely computable numbers.
- Coming back to the “game” motivation :
  - can we characterize rules that correspond to a game ?
  - influence of allowed rules : symmetric, one-way, ...
- Going Further :
  - Dynamics versus Dynamics of Evolutionary Game Theory ?
  - Can we characterize computable predicates ?

## Example 1 : Threshold Predicate

Suppose each agent starts with input 0 or 1.

Want to determine whether at least five agents have input 1.

Output convention : Each state has an associated output.  
Eventually, all agents reach states with the correct output.

(This was the problem of detecting bird flu epidemic.)

◀ Back

## Example 2 : Majority

Every agent is initially red or blue.

Determine whether  $\# \text{ reds} > \# \text{ blues}$ .



## Example 2a : 5%

Each agent has input 0 or 1.

Determine whether at least 5% of the inputs are 1.

## Example 2a : 5%

Each agent has input 0 or 1.

Determine whether at least 5% of the inputs are 1.

Similar to majority, except each 1 can cancel 19 0's.

◀ Back

## Example 2b : 40%

Each agent has input 0 or 1.

Determine whether at least 40% of the inputs are 1.

This is a bit trickier.

(Left as an exercise.)

◀ Back

## Example 3 : mod

Each agent initially has state 0 or 1.

Determine whether the number of 1's is divisible by 7.

(All agents must produce correct output.)

## Example 4 : Addition

Representing **addition** in the population protocol model.

Cannot use binary representation of inputs,  
since there is no structure on the agents.

Use **unary** to represent the problem of computing  $m + n$ .

Initially, we have  $m$  red agents and  $n$  blue agents.

Eventually, we have exactly  $m + n$  red agents.

## Example 5 : Division by 7

Initially, exactly  $m$  agents are red, rest are blue.

Eventually, exactly  $\lfloor m/7 \rfloor$  agents are red, rest are blue.

◀ Back

## Example 6 : Leader Election

Initially, all agents in same state.

Eventually, exactly one agent is in a special leader state.

◀ Back

## Example 6 : Leader Election

Initially, all agents in same state.

Eventually, exactly one agent is in a special leader state.



◀ Back

## How to Compute $\sum_{i=1}^k c_i x_i \geq a$

Input convention : each agent with  $i$ th input symbol starts in state  $c_i$ .

Each agent has a **leader bit** governed by 

Let  $m = \max(|a| + 1, |c_1|, \dots, |c_k|)$ .

Each agent also stores a value from  $-m, -m + 1, \dots, m - 1, m$ .

If a leader meets a non-leader, their values change as follows :

$$\begin{aligned}x, y &\rightarrow x + y, 0 && \text{if } 0 \leq x + y \leq m \\x, y &\rightarrow m, x + y - m && \text{if } x + y > m \\x, y &\rightarrow -m, x + y + m && \text{if } x + y < -m\end{aligned}$$

(In each case first agent on right hand side is the leader.)

Each agent also remembers **output** of last leader it met.

## Correctness

Sum of agents' values is invariant.

Sum is eventually gathered into the unique leader  
(up to maximum absolute value of  $m$ ) :

If  $sum > m$ , leader has value  $m \Rightarrow$  Output Yes.

If  $sum < -m$ , leader has value  $-m \Rightarrow$  Output No.

If  $-m \leq sum \leq m$ , leader's value is the actual sum  $\Rightarrow$  Output depends on sum.

In each case, leader knows output and tells everyone else.

How to Compute  $\sum_{i=1}^k c_i x_i \equiv a \pmod{b}$

Very similar (and easier) :  
gather sum into one agent, then disseminate answer.

# Characterization

## Theorem

*A predicate is computable iff it is on the following list.*

- $\sum_{i=1}^k c_i x_i \geq a$ , where  $a, c_i$ 's are integer constants  
(Generalization of Examples 1 and 2)
- $\sum_{i=1}^k c_i x_i \equiv a \pmod{b}$  where  $a, b$  and  $c_i$ 's are constants  
(Generalization of Example 3)
- Boolean combinations of the above predicates

We have proved  $\Leftarrow$ .

# Proving Converse

Want to show that all computable predicates are in the list.

Take any algorithm that computes a predicate.

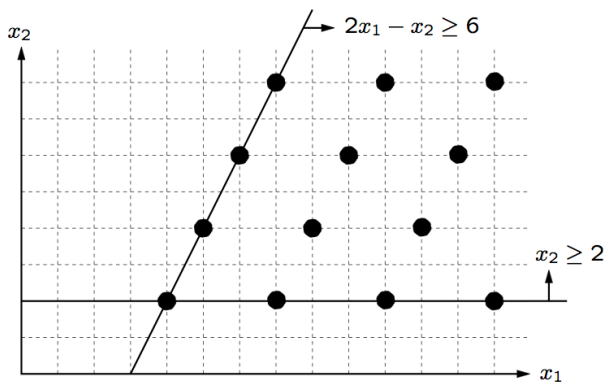
Step 1 : Show set of inputs that produce output Yes is semilinear.  
This step is hard !

Step 2 : Show that all linear sets are in the list.  
This is easy.

Step 3 : Show that all semilinear sets are in the list.  
This is trivial : semilinear is  $\vee$  of finite number of linear sets.

Conclusion : The computed predicate is on the list !

## Step 2 : Recognizing a Linear Set



**Algorithm:** Check  $x_2 \geq 1$ ,  $2x_1 - x_2 \geq 6$  and  $2x_1 - x_2 \equiv 0 \pmod{6}$ .

# Basic Ingredients of Step 1

- Higman's Lemma [1913] (subsets of  $\mathbb{N}^k$  closed under  $\leq$  have finitely many minimal elements).
- A Pumping Lemma.
- Use lots of abstract algebra on monoid cosets to decompose inputs that produce answer Yes into finitely many sets that look almost like linear sets.
- Tools from convex geometry to banish irrational numbers.

# Functions Beyond Predicates

What if we want every agent to converge to the value of function with **non-binary** output?

Note : output domain is finite.

Computable **iff**, for each possible output value  $v$ , the set of inputs that have output  $v$  is semilinear.

**Proof :**

( $\Rightarrow$ ) Just change output map : If function output is  $v$ , output 1. Otherwise, output 0.

( $\Leftarrow$ ) Compute all the predicates in parallel.  
Use their outputs to determine the function output.

# Variant #1

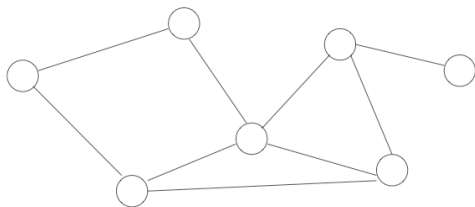
Limited Interaction Graph

## Limited Interaction Graph

Now, each agent sits at a node of a (connected) graph  $G$ .

Possible interactions are denoted by edges.

The basic model corresponds to the complete graph.



Does the interaction graph make the model weaker or stronger?

# Stronger !

Suppose the graph is a line.



Can simulate a linear-space Turing machine  
(if states are appropriately initialized).

Each agent simulates one square of the Turing machine tape.

State of agent represents the contents of the square.

The agent for the square currently containing the TM head also stores the TM state.

This is **much stronger** than the basic population protocol model !  
For example, it can compute the predicate  $x_1 = x_2 \cdot x_3$ .

(Can also do this simulation in any bounded-degree graph)

## At Least as Strong

Any connected interaction graph  $G$  can simulate the complete graph  $K$ .

Each agent in  $G$  simulates one agent in  $K$ .

Add interactions that allow adjacent agents to swap states.  
This allows simulated agents to “move around” freely and interact with non-adjacent simulated agents.

(This assumes non-determinism, but can be made deterministic.)

## At Least as Strong

Any connected interaction graph  $G$  can simulate the complete graph  $K$ .

Each agent in  $G$  simulates one agent in  $K$ .

Add interactions that allow adjacent agents to swap states.

This allows simulated agents to “move around” freely and interact with non-adjacent simulated agents.

(This assumes non-determinism, but can be made deterministic.)

# Interaction Graphs : An Example

Imagine agents are scattered across the landscape.

Each one only moves within a certain bounded area.

⇐ Movement restrictions determine interaction graph.

Agents might want to discover what the interaction graph looks like.

Idea : pre-programme agents to determine graph properties once they are deployed.

## Example : Does the Interaction Graph Have Low Degree ?

How to determine whether every vertex has degree  $< 3$  ?

Idea : Assume answer is Yes, until finding a violation.

All agents start out blue, with output Yes.

All agents start out as leaders.

If you are a leader :

Try to change 3 neighbours from blue to red.

(Always remember how many neighbours you have set to red.)

If you succeed, change your output to No.

Otherwise, after a while, reset neighbours back to blue and swap states with your neighbour. (Thus, leader moves around graph.)

When two leaders meet, one becomes non-leader, and both reset output to Yes, reset neighbours back to blue.

## Example : Does the Interaction Graph Have Low Degree ?

How to determine whether every vertex has degree  $< 3$  ?

Idea : Assume answer is Yes, until finding a violation.

All agents start out blue, with output Yes.

All agents start out as leaders.

If you are a leader :

Try to change 3 neighbours from blue to red.

(Always remember how many neighbours you have set to red.)

If you succeed, change your output to No.

Otherwise, after a while, reset neighbours back to blue and swap states with your neighbour. (Thus, leader moves around graph.)

When two leaders meet, one becomes non-leader, and both reset output to Yes, reset neighbours back to blue.

# Summing Up

The population protocol model was introduced in 2004.

We understand some things perfectly :

- basic model
- some variants (e.g. one-way communication, crash failures)

We understand some variants well :

- e.g. transient failures

There are scattered results for other variants.

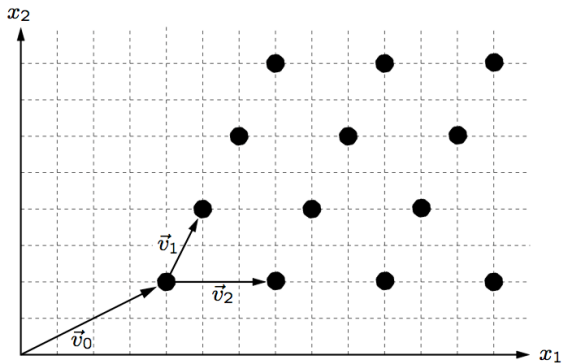
- mostly algorithms here and there.

A good start to mapping the terrain of population protocols.

## Where to Explore Next ?

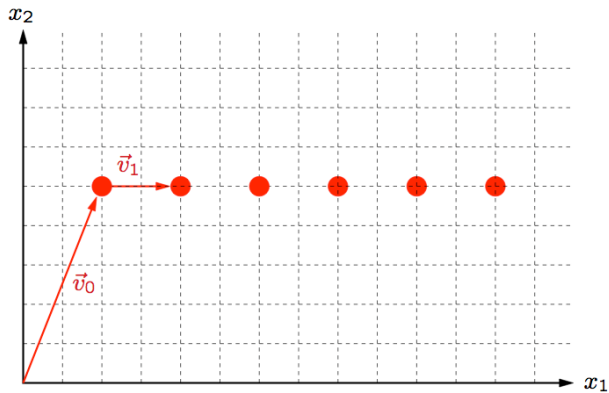
- Push the assumptions (slowly) towards a more realistic model.
- Study complexity.  
(There is a randomized simulation of register machines, represented in unary with  $b$ -bit registers running in  $O(b^4)$  time per simulated step.)
- Look at more realistic stochastic models of interactions.

# A Linear Set



$$S = \{\vec{v}_0 + c_1\vec{v}_1 + c_2\vec{v}_2 : c_1, c_2 \in \mathbb{N}\}$$

## Another Linear Set



$$T = \{\vec{v}_0 + c_1\vec{v}_1 : c_1 \in \mathbb{N}\}$$

