Population Protocols

Eric Ruppert
York University

MiNEMA Winter School
Göteborg, Sweden
March 24, 2009

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

How to Be Explorers

Imagine we are dropped into an unfamiliar landscape.

It is dark.

How should we discover our surroundings?

How to Be Explorers

Imagine we are dropped into an unfamiliar landscape.
It is dark.
How should we discover our surroundings?

Approach 1:
Everybody scatters, running off in all directions.

How to Be Explorers

Imagine we are dropped into an unfamiliar landscape.
It is dark.
How should we discover our surroundings?

Approach 1:
Everybody scatters, running off in all directions.
Some find cool stuff.
Some run into brick walls and fall off cliffs.

How to Be Explorers

Imagine we are dropped into an unfamiliar landscape.
It is dark.
How should we discover our surroundings?

Approach 1:
Everybody scatters, running off in all directions.
Some find cool stuff.
Some run into brick walls and fall off cliffs.

Approach 2:
Cover the ground systematically.

How to Be Explorers

Imagine we are dropped into an unfamiliar landscape.
It is dark.
How should we discover our surroundings?

Approach 1:
Everybody scatters, running off in all directions.
Some find cool stuff.
Some run into brick walls and fall off cliffs.

Approach 2:
Cover the ground systematically.
Draw a map as you go.
Make sure explorers talk to one another.

How to be Explorers

Advantages of scattering:

- Will discover some far-away things quickly.
- Can be more exciting.

Advantages of systematic approach:

- Will not miss nearby things.
- Does not waste re-exploring same area.
- Will fall off fewer cliffs.

Perhaps a mixture of the two approaches is ideal:
Some explorers run off into the distance,
while others explore local area systematically.

Theory of Mobile Computing

Theory is lagging behind practice in mobile computing.

Mobile systems are complex and difficult to prove theorems about.

We have not developed the theoretical tools to prove hard things about mobile systems.

How to Study Mobile Computing

Approach 1:

Many mobile computing papers different sets of model assumptions (and do not always state them clearly).

Some design cool systems and algorithms.

Some run into obstacles.

Approach 2:

Start with some simple models.

Characterize what can be done within the models.

Use common language to ensure models can be compared.

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.

Personal view: This work is less glamorous, but more satisfying.

Let's spend some time on this approach.

Identifying Assumptions

What sorts of assumptions do people make about mobile systems?

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

Stripping Away Assumptions

First, choose some aspects of the model to make absolutely minimal assumptions about.

Then, choose the weakest assumptions about the other aspects that avoid triviality.

Towards Population Protocols

Today: we choose to make 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.

Consequences For Other Aspects of Model

Mobility Pattern

Some fairness guarantee to avoid network partition.
(Details later.)

Energy

Total asynchrony implies no time complexity bounds.
Therefore, we cannot impose limits on energy.

Failures

This weak model cannot handle failures very well.
For now, assume no failures.
(We introduce failures later today.)

The Model (Informally)

Collection of identically programmed finite state machines.

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

Fairness guarantees that all possible interactions happen eventually.

How to Compute?

Encode each agent's input value in its initial state.
At any time, can interpret state of an agent as its output.

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

A Motivating Example: Birds

• Strap tiny, identical sensors to many birds in a flock.
(Or a colony of frogs).

• Sensors on two birds can interact when the birds are close together.

• Want to detect when (at least) five birds have elevated body temperatures, indicating possible epidemic.

What is an Algorithm?

An algorithm is described by

- a finite set of states,
- transition function: maps pairs of states to pairs of states,
- input encoding: maps inputs to multisets of states, and
- output interpretation: map from multisets of states to outputs.

Remark: Algorithm is independent of size of population.

Simplest Example: Computing OR of Input Bits

Each agent gets an input of 0 or 1.

Eventually every agent should learn the OR of the input bits.

Simplest Example: Computing OR of Input Bits

Each agent gets an input of 0 or 1.
Eventually every agent should learn the OR of the 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.

Another Motivating Scenario: Chemical Computation

Recipe for Population Protocols:

- Represent each possible state by a different type of molecule.
- Choose molecules so that chemical reactions between them represent the transitions.
- Measure out quantities of molecules according to desired input.
- Combine all in a test tube and shake well.

(Proposed by Banâtre and Le Métayer, 1986)

Executions

A configuration is a multiset of states.

We write $C \to C'$ if a single transition rule changes C to C'.

An execution is an infinite sequence of configurations $C_1, C_2, C_3, \ldots$, where $C_i \to C_{i+1}$ for all i.

Fairness

Anything that is always possible eventually happens.

If $C \to C'$ and C appears infinitely often in the execution,
then C' occurs infinitely often in the execution.

In one way, this is weaker than saying every pair of agents meet infinitely often:
There are fair executions in which some agents never meet.

In another way, fairness is stronger:
Avoids executions where agents are always in the "wrong" states to have an interaction whenever they meet each other.

Fairness: Motivation

Uniformity:

Definition of fairness works in many different variants of the model.

Captures Probabilistic Computation:

Suppose there is some (unknown) underlying probability distribution on interactions such that events are independent.

Then, unfair executions have probability 0.

Algorithms will work correctly with probability 1.

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.)

Example 1: Threshold Predicate

Use states $\{0, 1, 2, 3, 4, 5\}$. State 5 outputs 1; others output 0.

Accumulate number of birds in one agent:

Rule 1: $x, y \to \min(x + y, 5), 0$

Disseminate answer to all agents:

Rule 2: $5, * \to 5, 5$

If fewer than 5 sick birds, sum of states is invariant.
Otherwise, some agent reaches 5, then converts all agents to 5.

Example 2: Majority

Every agent is initially red or blue.

Determine whether # reds $>$ # blues.

Fairness Revisited

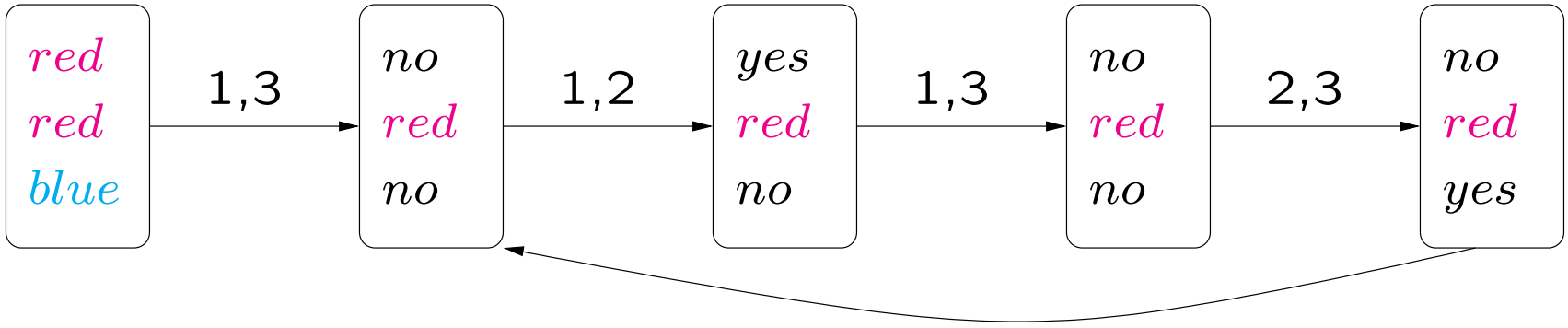
Consider following algorithm to decide if # reds $>$ # blues.

States: $\{red, blue, yes, no\}$.

Rules:

$red, blue$	$\rightarrow$	no, no	eliminates all $blues$ or all $reds$
red, no	$\rightarrow$	red, yes	red changes answers to yes
$blue, yes$	$\rightarrow$	$blue, no$	$blue$ changes answers to no
yes, no	$\rightarrow$	no, no	takes care of a tie

Is this execution a problem?



Example 2a: 40%

Each agent has input red or blue.

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

Example 2a: 40%

Each agent has input red or blue.

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

2 red agents cancel 3 blue agents.

(Details left as an exercise.)

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: Leader Election

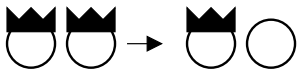
Initially, all agents in same state.

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

Example 6: Leader Election

Initially, all agents in same state.

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



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 $P(x_1, x_2, \ldots, x_k)$ where
$k =$ number of possible initial states,
$x_i =$ number of agents starting in ith 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

  (Generalization of threshold and majority)

- $\sum_{i=1}^{k} c_i x_i \equiv a \pmod{b}$ where a, b and c_i's are constants

  (Generalization of sum mod 7)

- Boolean combinations of the above predicates

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

Input convention: each agent with ith input symbol starts in state c_i.

Each agent has a leader bit governed by 

Let $m = \max(|a| + 1, |c_1|, \ldots, |c_k|)$.
Each agent also stores a value from $-m, -m+1, \ldots, m-1, m$.

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

$x, y \to x + y, 0$ if $0 \leq x + y \leq m$

$x, y \to m, x + y - m$ if $x + y > m$

$x, y \to -m, x + y + m$ if $x + y < -m$

(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 (mod b) into one agent, then disseminate answer.

How to Do Boolean Operators?

Suppose P and Q are computable predicates.

How to compute $\neg P$?

How to compute $P \wedge Q$?

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

We have proved $\Leftarrow$.

Alternate Characterization: Presburger Arithmetic

A predicate is computable iff it can be expressed in first-order logic using the symbols $+, 0, 1, \wedge, \vee, \neg, \forall, \exists, =, <, (,)$ 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% red: $\neg(x_1 + x_1 < x_0 + x_0 + x_0)$

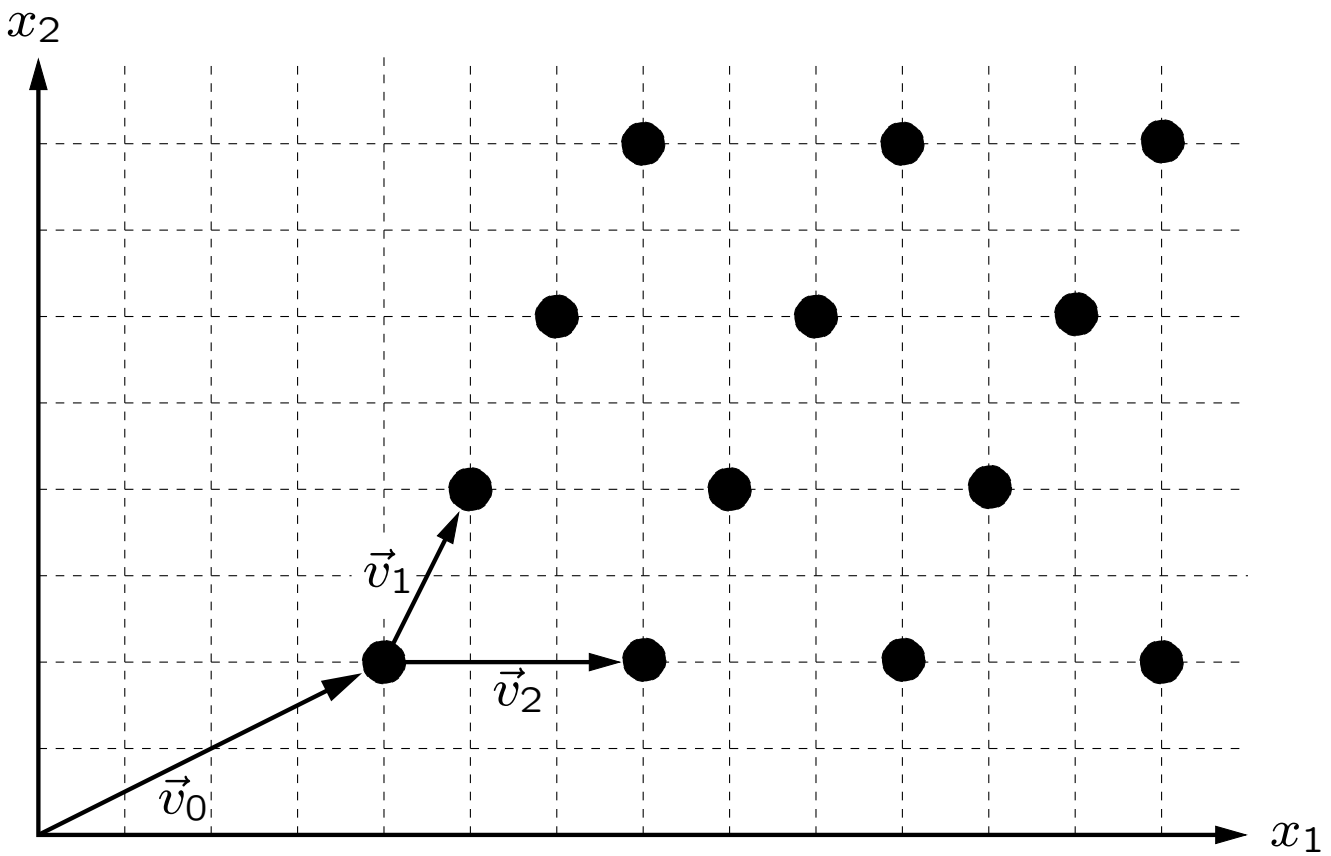
Alternate Characterization: Semilinear Sets

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

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

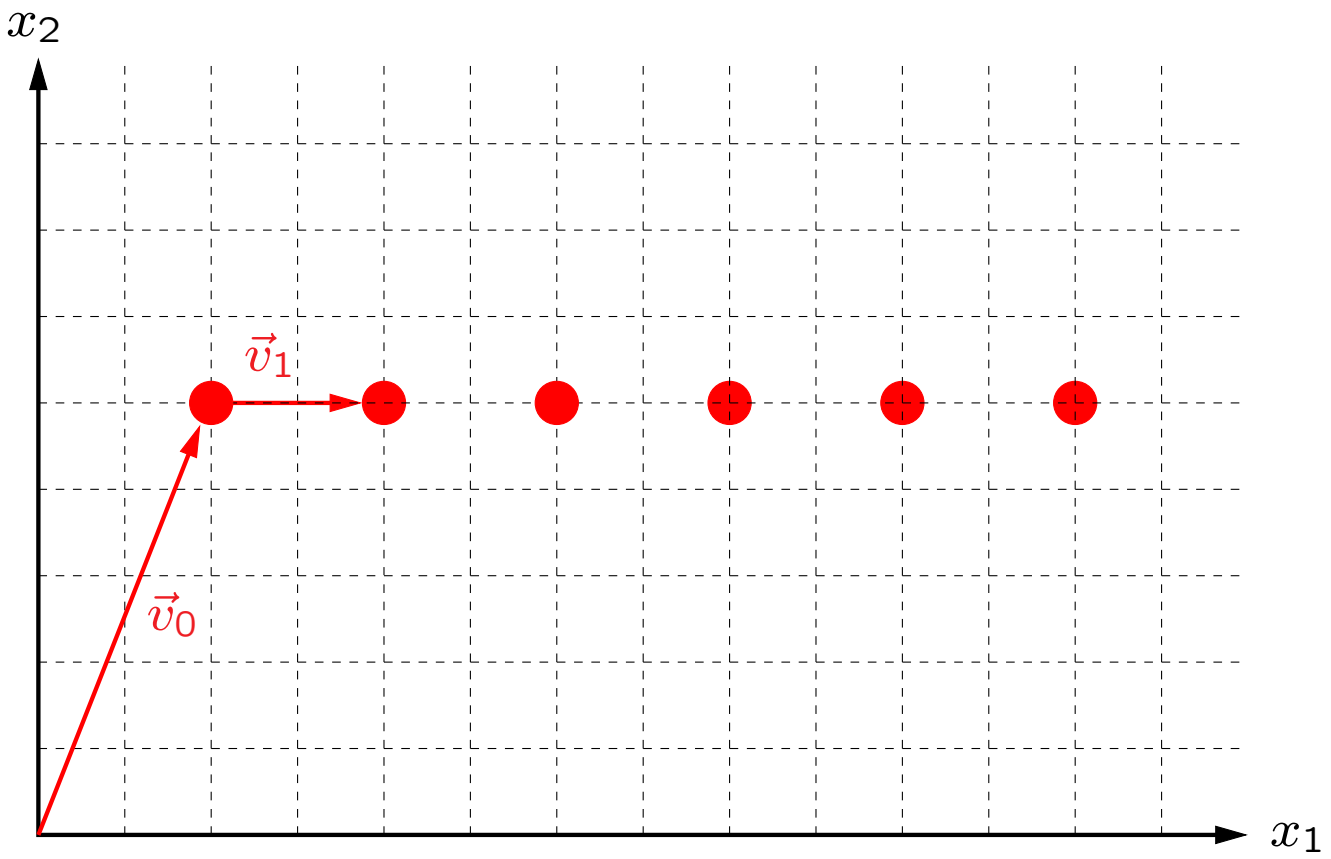
A set of vectors is semilinear if it is a finite union of linear sets.

A Linear Set



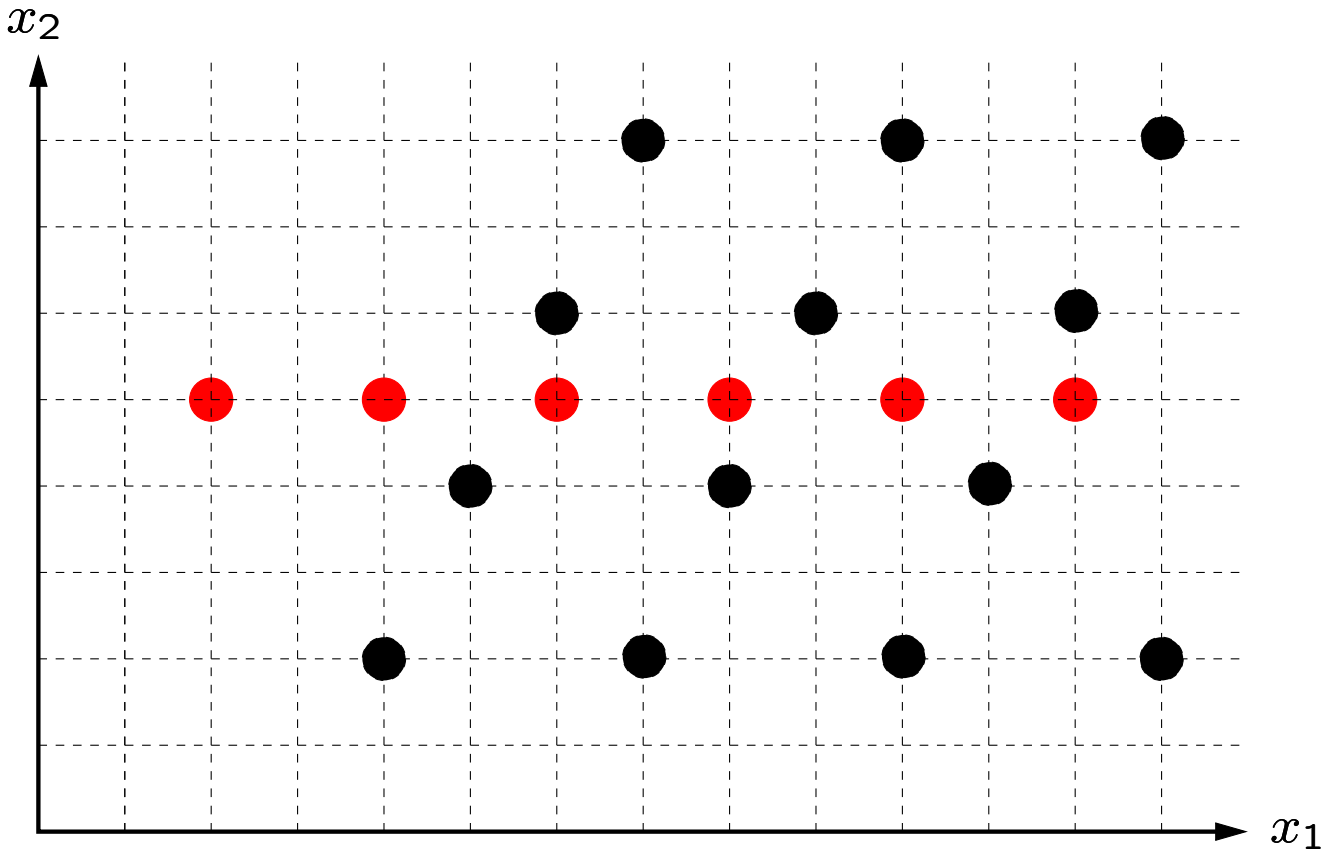
$$S = \{\vec{v}_0 + c_1\vec{v}_1 + c_2\vec{v}_2 : c_1, c_2 \in \mathbb{N}\}$$

Another Linear Set



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

A Semilinear Set



$S \cup T$

Proving Converse

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

Take any algorithm that computes a predicate.

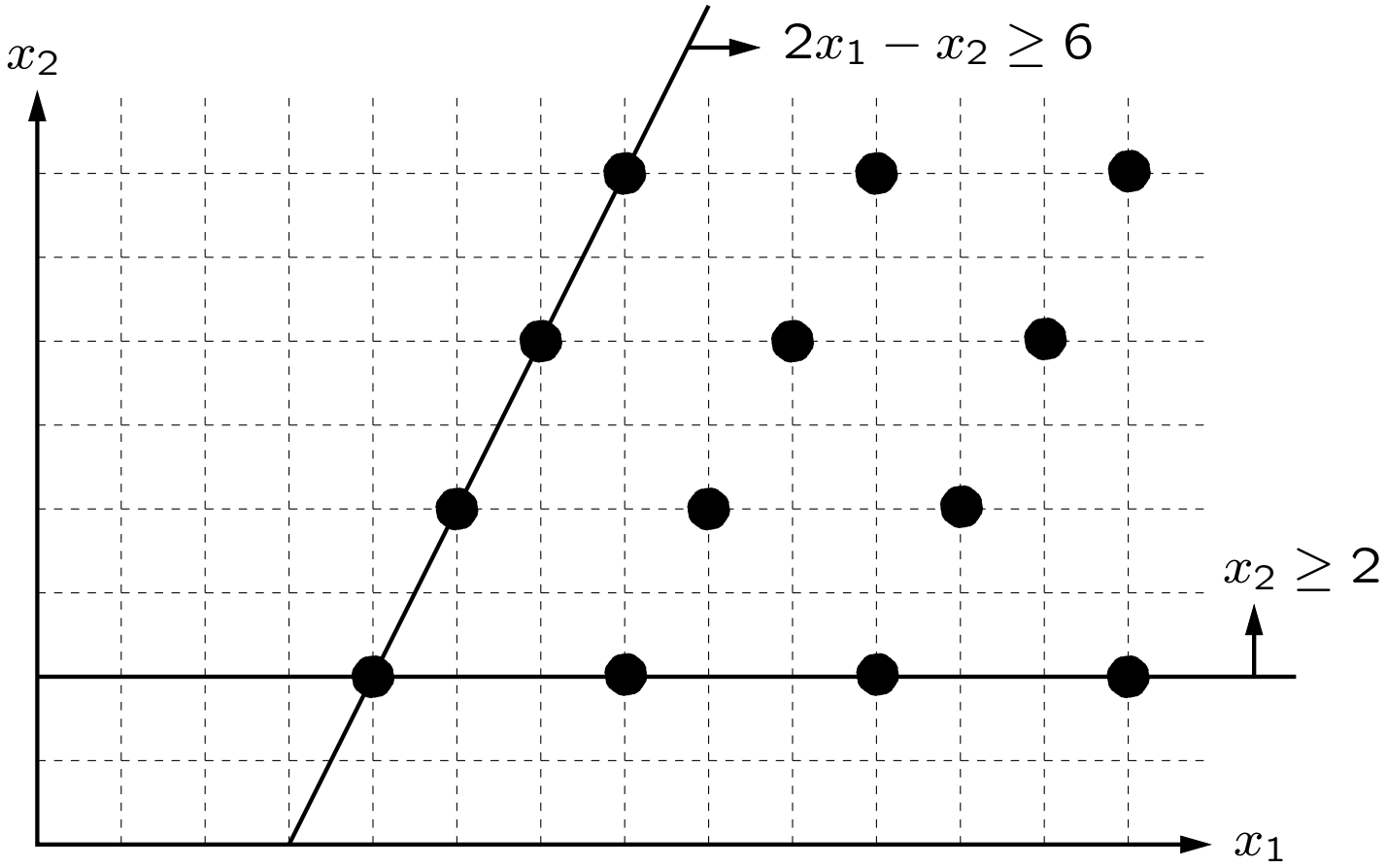
Step 1: Show set of inputs that produce output Yes is semilinear.
This step is hard!

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 $2x_1 - x_2 \equiv 0 (\text{mod } 6)$.

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.

What Do We Know?

Now we know exactly which predicates are computable in the basic population protocol model.

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

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.

Variants of the Model

We now understand basic model pretty well.

Variants to be considered:

- Limited interaction graph
- Failures
- Agents with unique ids (and slightly larger memories)
- One-way communication

Variant #1

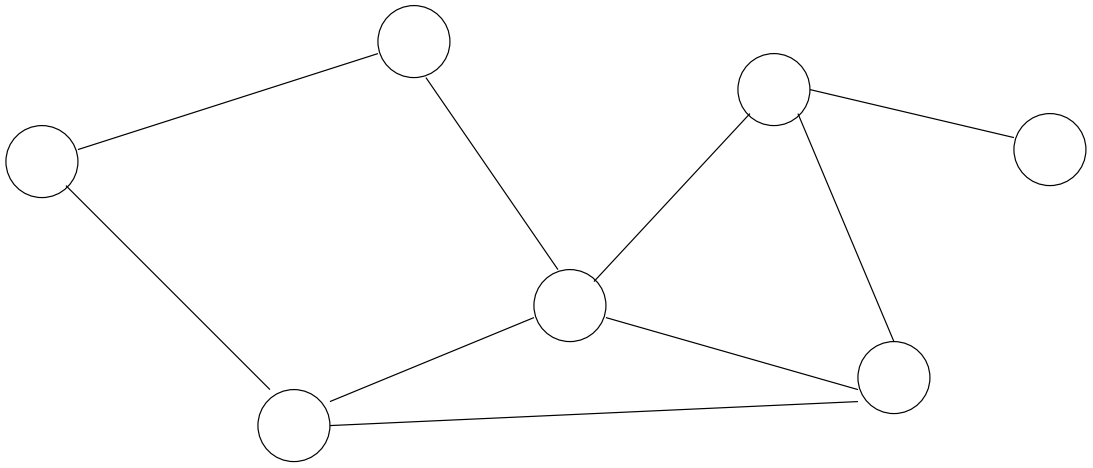
Limited Interaction Graph

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.

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.
$\Rightarrow$ Movement restrictions determine interaction graph.

Agents might want to discover what interaction graph looks like.
Idea: pre-programme agents to determine graph properties once they are deployed.

For example, a population protocol can determine whether the interaction graph has degree < 7.

Variant #2

Failures

(And back to complete interaction graph.)

Failures

What if those sick birds drop dead (along with their sensors)?

Useful computations can be done in the presence of failures.

Different types of failures have been considered:

- Crash failure: an agent crashes without warning.
- Transient failure: an agent's state is corrupted.
- Byzantine failure: faulty agents can behave arbitrarily badly.

Main Result for Crash Failures

There is a general construction to transform any protocol for the failure-free model to a protocol that tolerates crash failures.

(Transient failures can be handled similarly.)

Making the Problem Specification Fault-Tolerant

What if failures happen right away, obscuring some inputs?

Solution: Add preconditions on possible inputs.

Majority Example:
Must determine whether more inputs are red or blue.

Can tolerate c crash failures, if you know $|\#\text{reds} - \#\text{blues}| > c$.
(Even if c agents crash, the winner is still clear.)

The Main Idea

- Start with any protocol that does not tolerate failures.

- Use replication:

Divide agents into $O(1)$ groups.

Each group simulates a run of the entire system.

- Ensure each failure interferes with at most two simulated runs.

- Use enough simulated runs that a majority will be correct.

Difficulties

• Limiting the effects of failures.

Elements of the solution:

- A single agent should not have a critical role: no leaders.
- Replication works even if groups are only roughly equal in size.
- Ensure each failure only "pollutes" the simulations of 1 or 2 groups.

• Agents do not know when a protocol (or part of it) is complete: hidden agents could suddenly appear, requiring more computation.
Solution: Allow phases to overlap.

• Careful bookkeeping required to overcome anonymity and $O(1)$ space per agent restriction.

Results for Crash Failures

We know exactly what is computable with a small number of crash failures.

We also know good bounds on what is computable with small number of crash and transient failures.

What about Byzantine Failures?

It is easy to prove that even 1 Byzantine agent can wreak havoc on a population protocol.

Essentially, nothing is computable.

However, we can beat Byzantine agents in two ways:

- Move to probabilistic model. (Majority is doable. What else?)
- Give agents unique, unforgeable ids.

Variant #3

Community Protocol Model.

Community Protocols

Each agent has a unique id.

Agents can compare two ids to see which one is bigger.

Each agent can store a constant number of other agents' ids (plus its usual constant number of bits of memory).

Additional Power of Community Protocols

For the failure-free case, the ids allow the agents to organize themselves to simulate a Turing machine.

Anything that is computable is Turing machine using $O(n \log n)$ space is computable in a community protocol.

(Requires a fairly tricky simulation, with lots of coordination between agents.)

Ids help against Byzantine Failures

The replication and simulation idea used for crash failures also works against Byzantine failures if

- agents have unique ids,
- appropriate preconditions are added, and
- Byzantine agents are not allowed to forge their own id.

Essentially, everything Turing-computable using $O(n \log n)$ space can be computed in this way.

Variant #4

One-Way Communication

(and back to population protocol model: no ids)

One-Way Communication

Basic model assumes a pairwise interaction can update both agents' states simultaneously.

We now focus on one-way interactions:
Information can flow from sender to receiver, but not vice versa.

Modelling Issues

(1) Is sender aware that it has sent a message?

Transmission model: sender actively sends message to (unknown) receiver.
Observation model: sender passively observed by receiver.

(2) Does the information flow instantaneously?

Immediate transmission: message delivered instantly.
Immediate observation: receiver sees sender's current state.
Delayed transmission: unpredictable delay in delivery.
Delayed observation: receiver sees an old state of sender.

(3) Can incoming messages be queued?

The Pirahã Tribe

The Pirahã tribe of Brazil speak a language that has only the following words for numbers:

- hói: one
- hoí: two
- baagi: many
- aibai: many

Experiments suggest that they cannot really distinguish numbers larger than two.

Reference: Peter Gordon. Numerical cognition without words: Evidence from Amazonia. *Science*, volume 306, pages 496-499, 2004.

One-Way Models: Overview

We have exact characterizations of computable predicates in one-way models.

- Delayed Observation: can count up to 2.
- Immediate Observation: can count up to any constant.
- Immediate and Delayed Transmission: characterization exists.
  - Strictly stronger than observation (can compute mods).
  - Strictly weaker than two-way (cannot compute majority).
- Queued transmission is equivalent to two-way interactions.

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)

There are scattered results for other variants.

- mostly algorithms here and there.

A good start to mapping the terrain of population protocols.

Further Reading

The first paper:
Angluin *et al.* Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(3), 2006.

Survey: in your textbook.

James Aspnes's site `www.cs.yale.edu/~aspnes` has many papers.

Where to Explore Next?

- Push the assumptions (slowly) towards a more realistic model.
- Study complexity. (Some limited results exist.)
- Look at more realistic stochastic models of interactions.

Where to Explore Next?

Beyond population protocols:

Where do you think theory needs to catch up to practice?

The Last Slide

These slides are at `www.cse.yorku.ca/~ruppert/goteborg.pdf`

Thanks!