Population protocols and Turing machines

LEFÈVRE Jonas

LIX

May 24, 2011

Introduction

A computation model introduced by Angluin, Aspnes, Diamadi, Fisher and Peralta in 2004.

It models very large networks of passively mobile and anonymous devices.

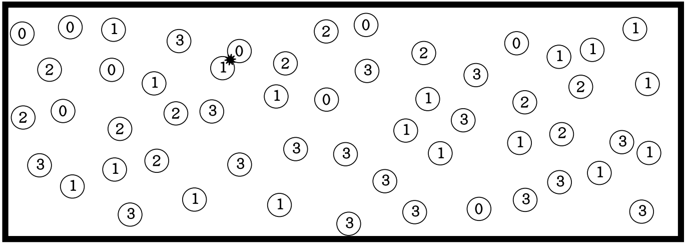
Plan

Plan

1. Population Protocols
2. Population Protocols and Turing Machines

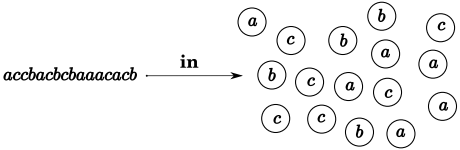
What is a population protocol ?

- A very large population of devices.
- The devices are passively mobile (no control on the walk), anonymous (no identification) and weak (no more computational power than a finite automaton).
- When they meet, they can interact.
- The answer is read on the stabilised population.

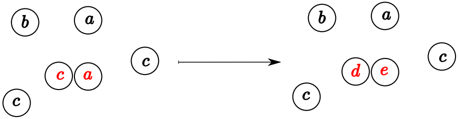


How does it work ?

- The input word ω is distributed on a population.

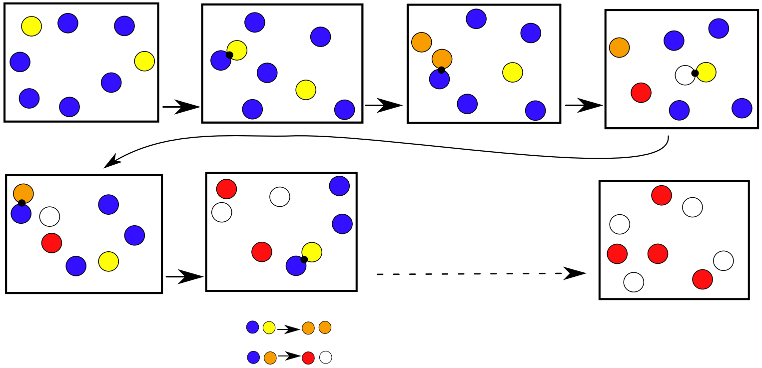


- Some devices interact.



- After some time, the population becomes output-stable : from this instant and forever, all the agent give the same output.

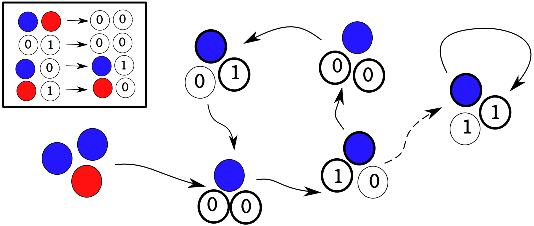
A computation



The fairness property

Definition (Fairness)

A fair execution is a sequence of configurations such that if a configuration C appears infinitely often in the execution and $C \to C'$ for some configuration C', then C' must appear infinitely often in the execution.



Population protocols

Definition (Population Protocol)

A population protocol is a 6-uplet $(Q, \Sigma, Y, in, out, \delta)$ where:

- *Q is the set of the states for the devices*
- *Σ the input alphabet and Y the output one*
- *$in : \Sigma \to Q$ the input function*
- *$out : Q \to Y$ the output function*
- *$\delta \subset Q^2 \times Q^2$ the transition relation*

Presburger Arithmetic

Definition (Presburger predicates)

A predicate of the Presburger Arithmetic is a predicate that can be written only with $\wedge, \vee, \neg, \exists x, \forall x, +, \leq, =, 1$ *and* 0.

Examples :

- $P(x) = [3 + 2x \geq 7]$

Presburger Arithmetic

Definition (Presburger predicates)

A predicate of the Presburger Arithmetic is a predicate that can be written only with $\wedge, \vee, \neg, \exists x, \forall x, +, \leq, =, 1$ *and* 0.

Examples :

- $P(x) = \left[1+1+1+1+1+1+1 \leq 1+1+1+x+x\right]$

Presburger Arithmetic

Definition (Presburger predicates)

A predicate of the Presburger Arithmetic is a predicate that can be written only with $\wedge, \vee, \neg, \exists x, \forall x, +, \leq, =, 1$ *and* 0.

Examples :

- $P(x) = \left[3 + 2x \geq 7\right]$
- $Q(x, y) = \left[x \equiv y \bmod 5\right]$

Presburger Arithmetic

Definition (Presburger predicates)

A predicate of the Presburger Arithmetic is a predicate that can be written only with $\wedge, \vee, \neg, \exists x, \forall x, +, \leq, =, 1$ *and* 0.

Examples :

- $P(x) = \left[3 + 2x \geq 7\right]$
- $Q(x, y) = \left[\exists k(x = y + k + k + k + k + k)\right]$

Presburger Arithmetic

Definition (Presburger predicates)

A predicate of the Presburger Arithmetic is a predicate that can be written only with $\wedge, \vee, \neg, \exists x, \forall x, +, \leq, =, 1$ *and* 0.

Examples :

- $P(x) = \left[3 + 2x \geq 7\right]$
- $Q(x, y) = \left[x \equiv y \bmod 5\right]$
- $R(x, y, z) = \left[\exists k(x + y = 4.k) \wedge (z \leq x - 2y)\right]$

Presburger Arithmetic

Definition (Presburger predicates)

A predicate of the Presburger Arithmetic is a predicate that can be written only with $\wedge, \vee, \neg, \exists x, \forall x, +, \leq, =, 1$ *and* 0.

Examples :

- $P(x) = \left[3 + 2x \geq 7\right]$
- $Q(x, y) = \left[x \equiv y \bmod 5\right]$
- $R(x, y, z) = \left[\exists k(x + y = k + k + k + k) \wedge (z + y + y \leq x)\right]$

Presburger Arithmetic

Definition (Presburger predicates)

A predicate of the Presburger Arithmetic is a predicate that can be written only with $\wedge, \vee, \neg, \exists x, \forall x, +, \leq, =, 1$ *and* 0.

Examples :

- $P(x) = \left[3 + 2x \geq 7\right]$
- $Q(x, y) = \left[x \equiv y \bmod 5\right]$
- $R(x, y, z) = \left[\exists k(x + y = 4.k) \wedge (z \leq x - 2y)\right]$
- but neither $A(x, y, z) = \left[x.y = z\right]$
  nor $B(x, y) = \left[y = 0 \bmod x\right] = \left[\exists k(x.k = y)\right]$

Computed Set

A semi-linear set is a finite union of

$$\{x + k_1 v_1 + \cdots + k_p v_p | k_1, \ldots, k_p \in \mathbb{N}\}$$

where $x, v_1, \ldots, v_p$ are vectors of $\mathbb{N}^d$.

Theorem

The computable subset of $\mathbb{N}^d$ by population protocols are exactly the semi-linear set.
Those set can be described with Presburger predicates.

Semi-linear set

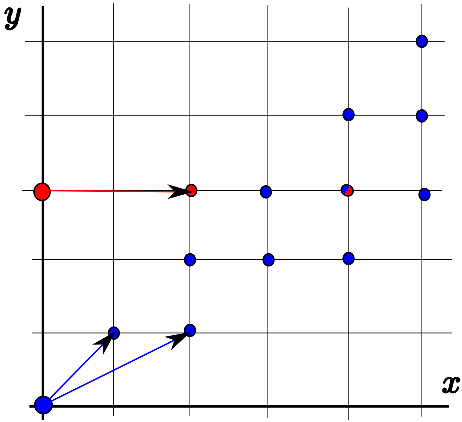


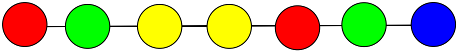
Figure: $\big((y = 3) \wedge (x = 0 \bmod 2)\big) \vee \big((x \leq y) \wedge (y \leq 2x)\big)$

Plan

Population protocols on strings

Definition

A population protocol on a string is population protocol where the devices are on the vertices of a string graph. Two devices can interact only if they are bound by an edge.



We call $PP(C_n)$ the set of the function computable by a population protocol on a string.

Definitions

Definition

$M \in RSPACE(s)$ *if*

- $M(x)$ *uses at most* $s(|x|)$ *squares and ends with probability 1.*
- $\forall x \in L, P(M(x) = 1) \geq 1/2$
- $\forall x \notin L, P(M(x) = 1) = 0$

$ZPSPACE(s) = RSPACE(s) \cap coRSPACE(s)$

Definition

$NSPACE(s)$ *is the set of non-deterministic Turing machines working with a space bound by* s

Definitions

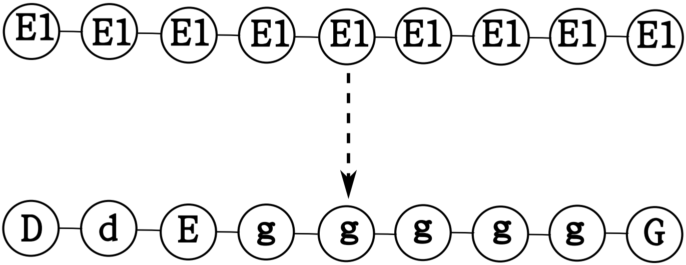
Definition

$XSPACE_{sym}(f) = \{L \in XSPACE(f) | \forall \pi \textit{ permutation of positions } L(x) = L(\pi(x))\}.$

The protocol *Organise*

Theorem

There is a population protocol on string Organise which transforms a uniform input into an organised string.



How does it work ?

- Each agent initially contains a head.
- Each head tries to construct an organised area around it.
- The heads oscillate into their respective areas and try to extend them.
- When two heads meet (at a border), one dies and resets its area.
- The fairness assures we obtain an organised string.

Population Protocols on strings and Turing Machines

Theorem

$ZPSPACE_{sym}(n) \subseteq PP(C_n)$

Theorem

Every population protocols on strings can be simulated by a Turing machine of $NSPACE_{sym}(n)$

$NSPACE_{sym}(n) = ZPSPACE_{sym}(n)$

Proposition

$$NSPACE_{sym}(n) = RSPACE_{sym}(n) = ZPSPACE_{sym}(n)$$

Caracterization of $PP(C_n)$

$$\begin{aligned} PP(C_n) &\subseteq NSPACE_{sym}(n) \\ ZPSPACE_{sym}(n) &\subseteq PP(C_n) \\ ZPSPACE_{sym}(n) &= NSPACE_{sym}(n) \end{aligned}$$

Caracterization of $PP(C_n)$

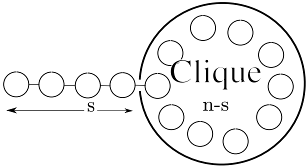
$$\begin{array}{rcl} PP(C_n) & \subseteq & NSPACE_{sym}(n) \\ ZPSPACE_{sym}(n) & \subseteq & PP(C_n) \\ ZPSPACE_{sym}(n) & = & NSPACE_{sym}(n) \end{array}$$

Theorem

$$PP(C_n) = ZPSPACE_{sym}(n) = NSPACE_{sym}(n)$$

Conclusion

- Classical population protocols compute the Presburger arithmetic.
- Population protocols on strings are equivalent to non-deterministic Turing machines using linear space.
- Those results and methods can be used on intermediate situations :



- Further studies : other restricted communication graph, partial identification of the devices,etc.

References I

D. Angluin, J. Aspnes, Z. Diamadi, M. Fisher, and R. Peralta.
Computation in networks of passively mobile finite-state sensors.
pages 290–299, 2004.

D. Angluin, J. Aspnes, Z. Diamadi, M. Fisher, and R. Peralta.
Computation in Networks of Passively Mobile Finite-State Sensors .
Distributed Computing, (18(4)):235–253, 2006.

D. Angluin, J. Aspnes, and D. Eisenstat.
Stably Computable Predicates are Semilinear.
in Proc. 25th Annual ACM Symposium on Principles of Distributed Computing, pages 292–299, 2006.

References II

D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert.
The computational power of population protocols .
Springer-Verlag, 2007.

D. Angluin and E. Ruppert.
An Introduction to Population Protocols.
2007.

S. Ginsburg and E. Spanier.
Semigroups, Presburger formulas, and languages.
Pacific Journal of Mathematics, (16):285–296, 1966.

References III

M. Saks.

Randomization and derandomization in space-bounded computation.

In *Computational Complexity, 1996. Proceedings., Eleventh Annual IEEE Conference on*, pages 128–149. IEEE, 1996.