

Computing with Large Populations

Olivier Bournez Pierre Fraigniaud Xavier Koenigler

Ecole Polytechnique
Laboratoire d'Informatique de l'X
Palaiseau, France

AlgoGT
June 20th 2011



Menu

Inspiration: Classical Population Protocols

Large Population Protocols

Discussion and Conclusion

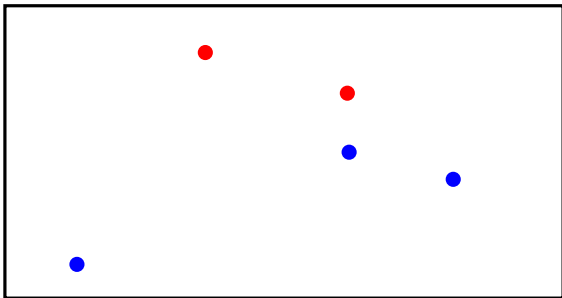
Inspiration: (classical) Population Protocols

Introduced by [Angluin, Aspnes, Diamadi, Fischer, Peralta 2004] in the context of distributed systems.

A model of anonymous (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.

Example 1: starting with 3 blues, 2red



Program:

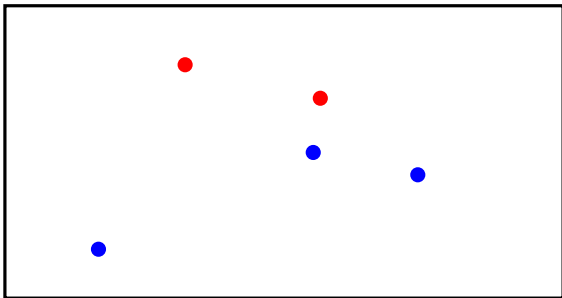
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

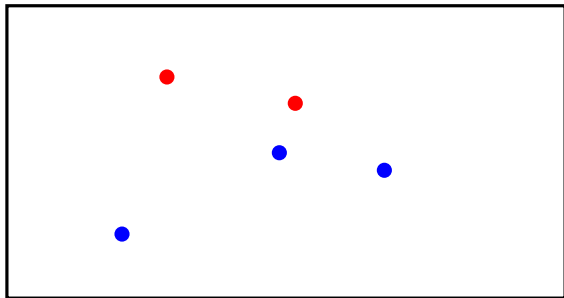
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

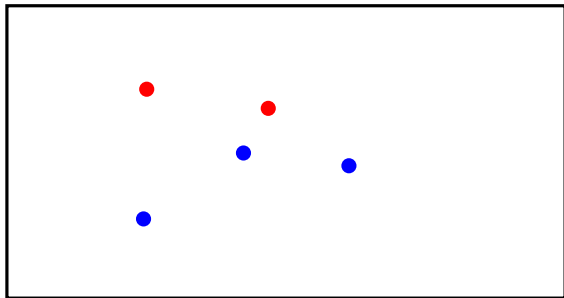
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

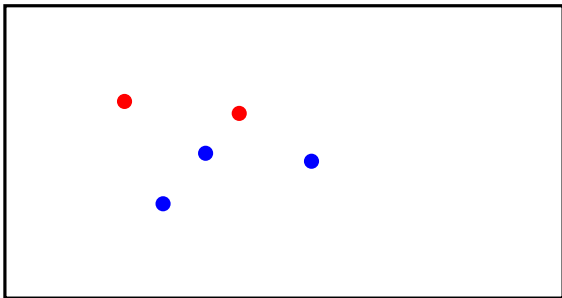
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

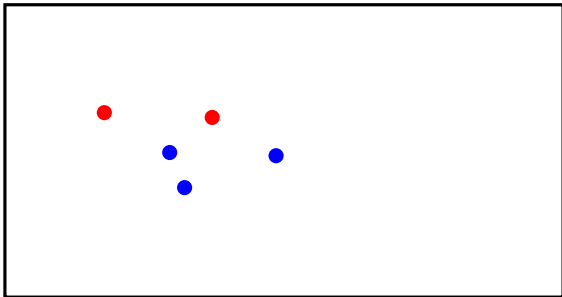
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

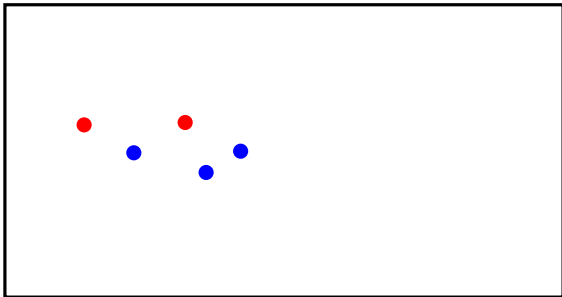
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

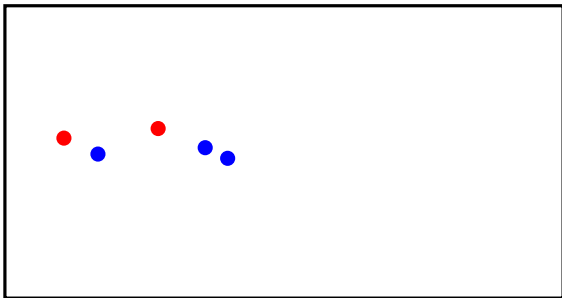
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

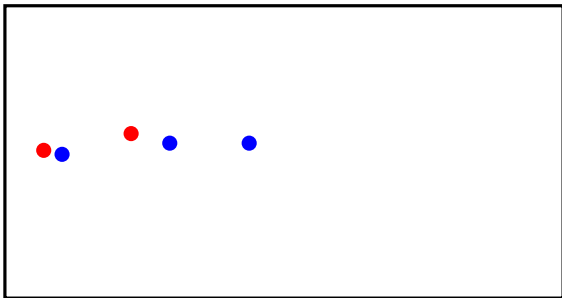
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

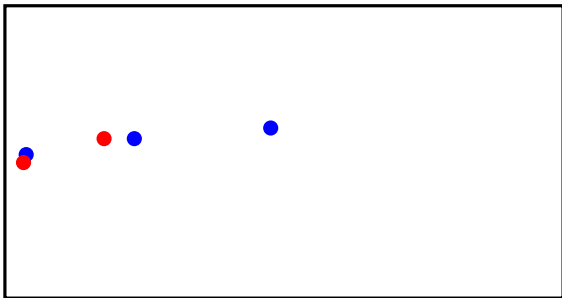
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

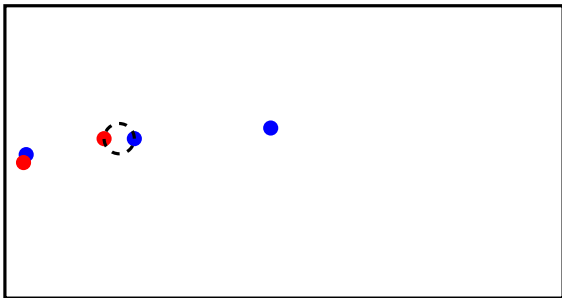
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

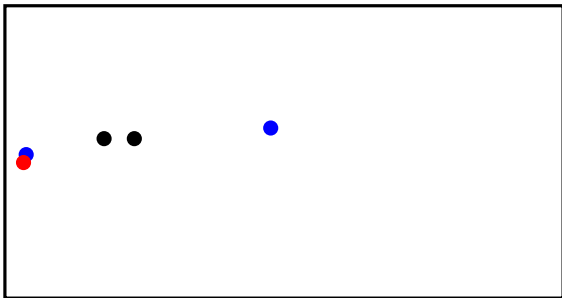
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

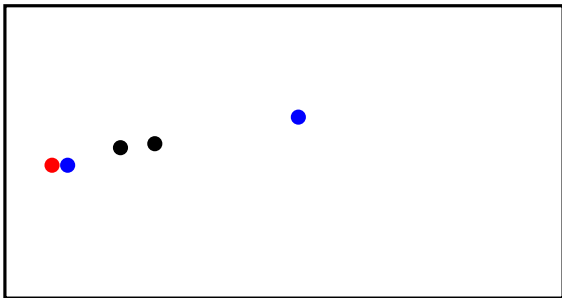
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

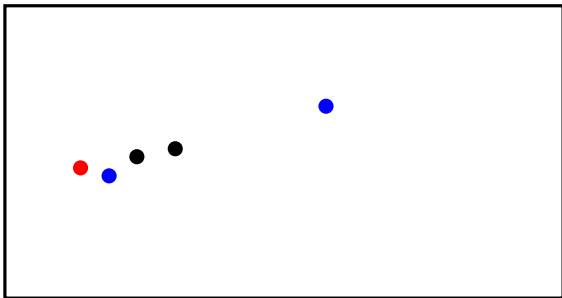
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

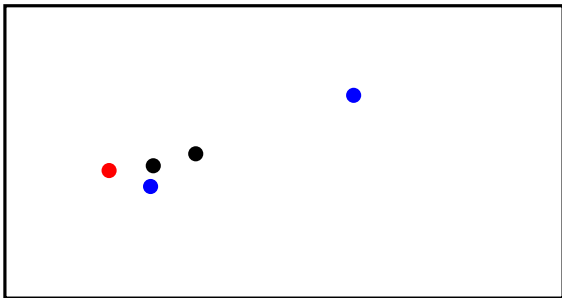
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

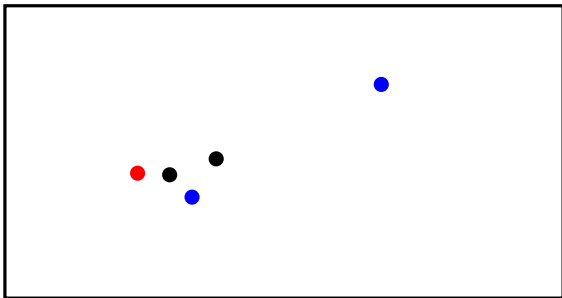
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

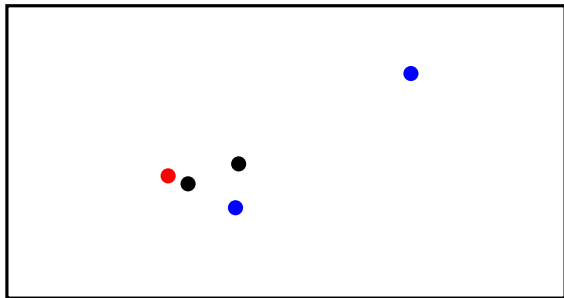
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

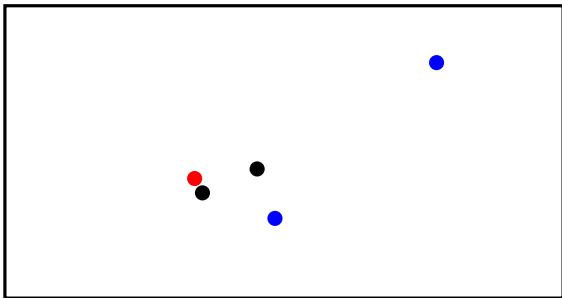
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

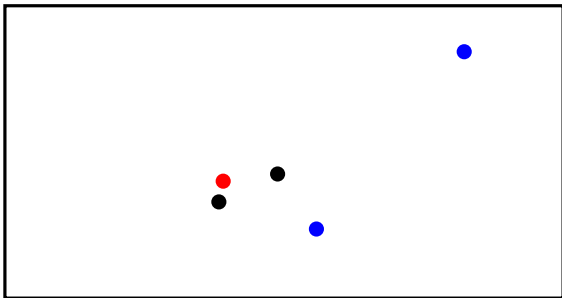
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

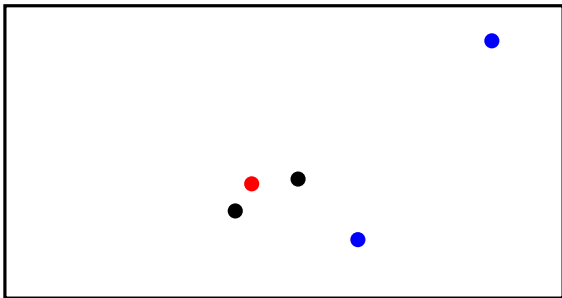
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

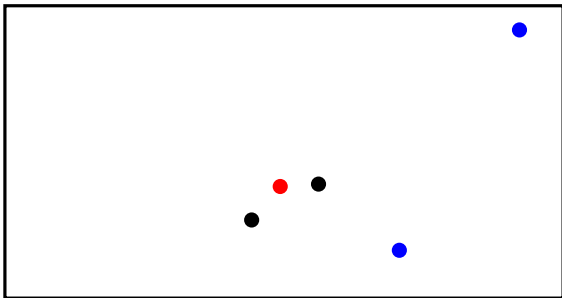
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

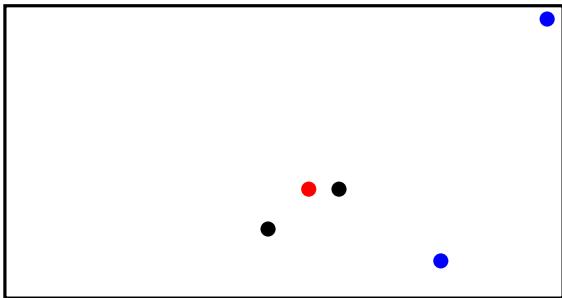
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

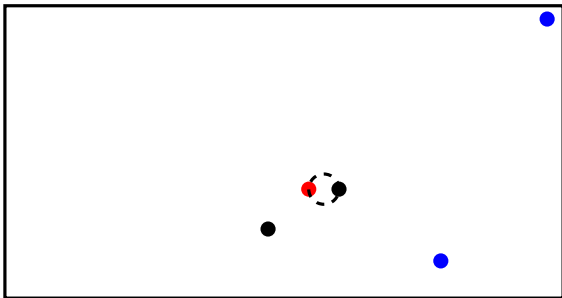
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

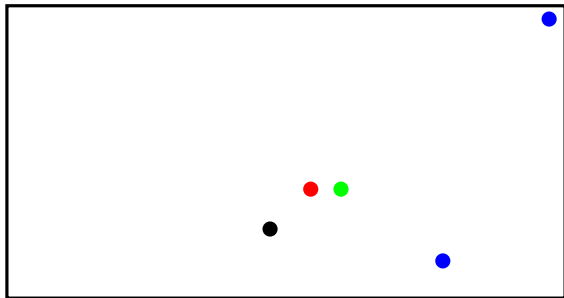
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

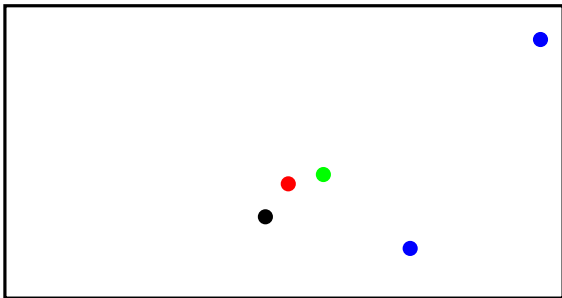
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

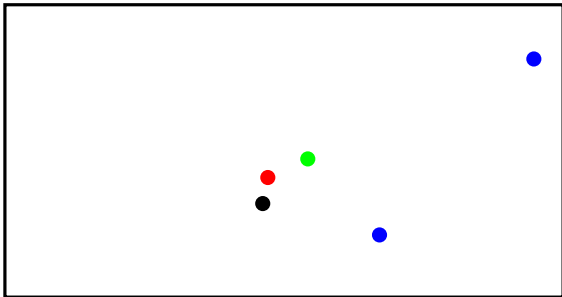
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

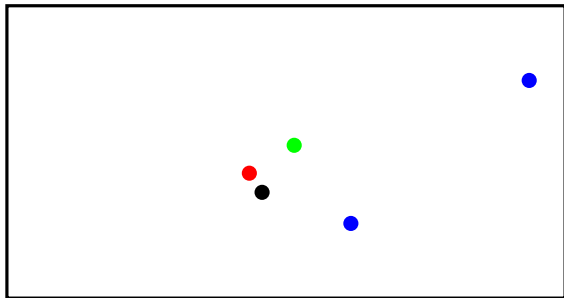
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

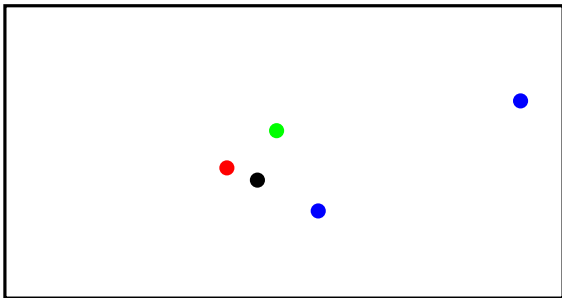
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

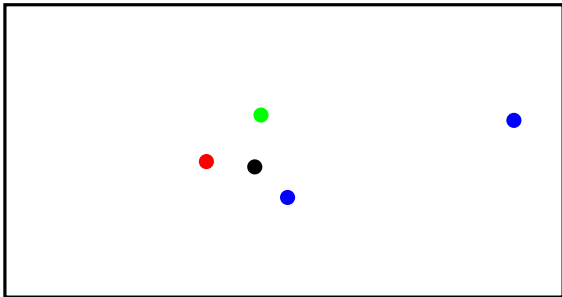
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

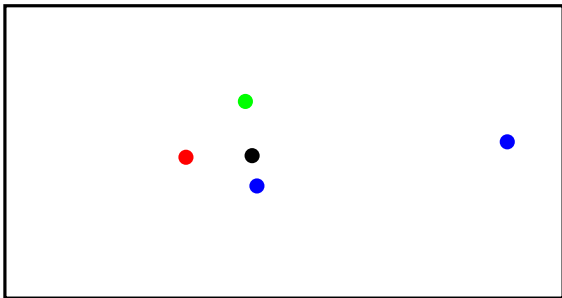
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

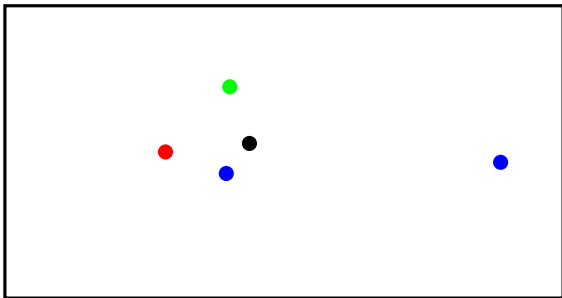
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

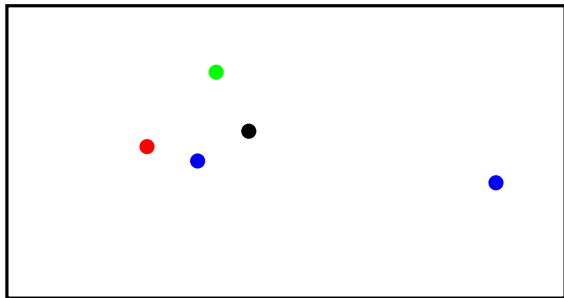
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

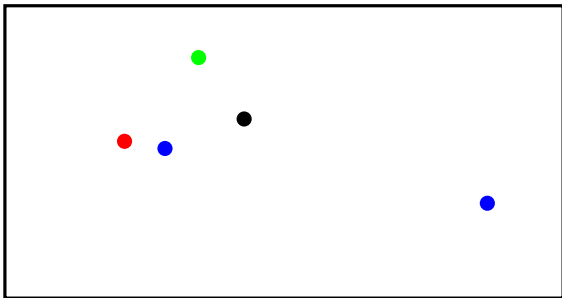
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

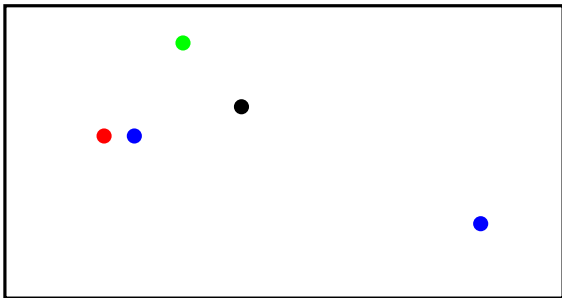
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

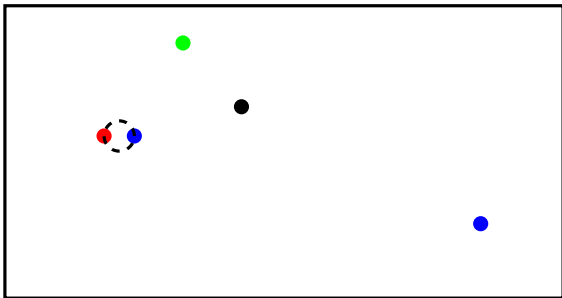
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

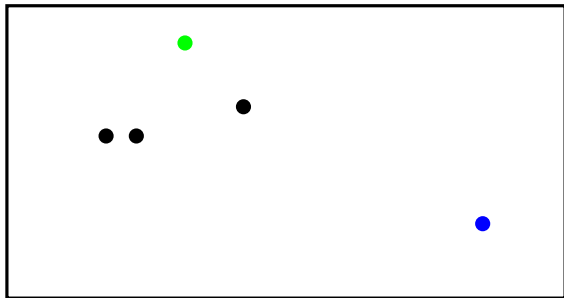
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

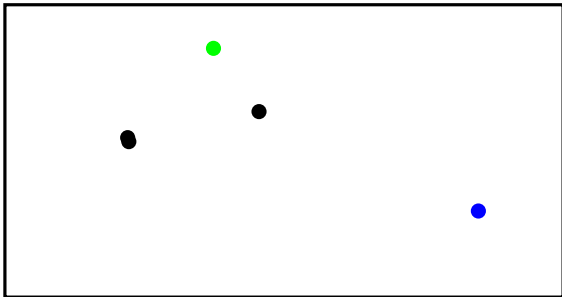
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

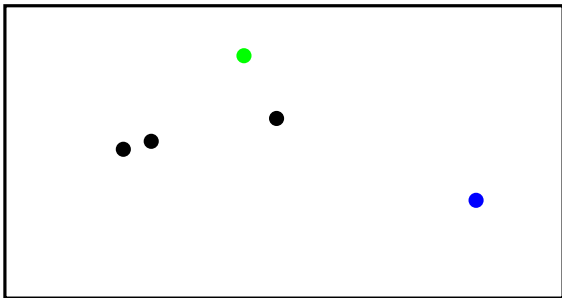
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

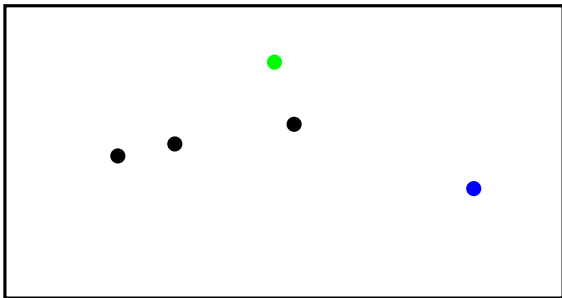
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

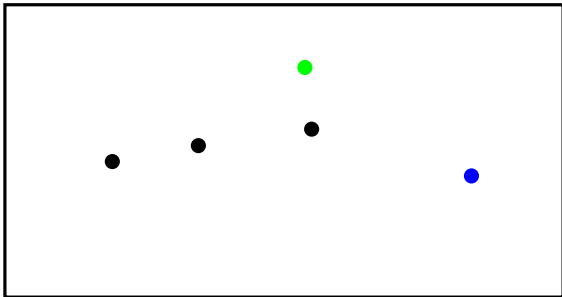
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

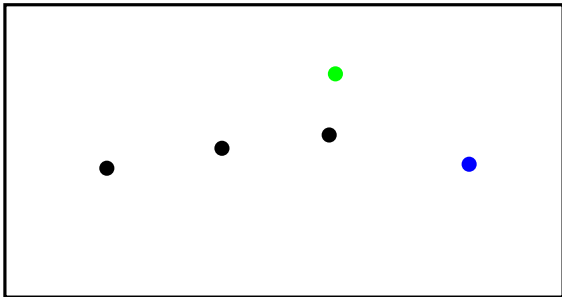
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

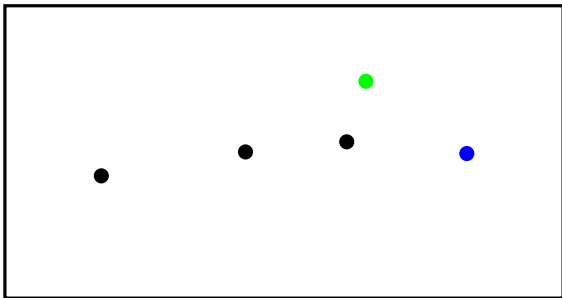
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

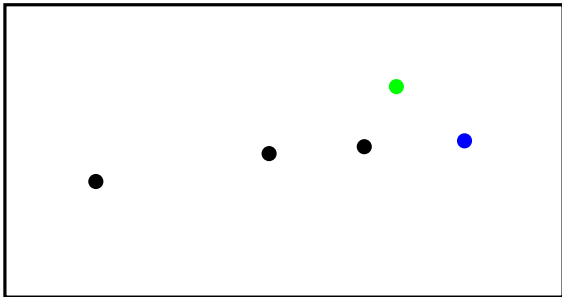
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

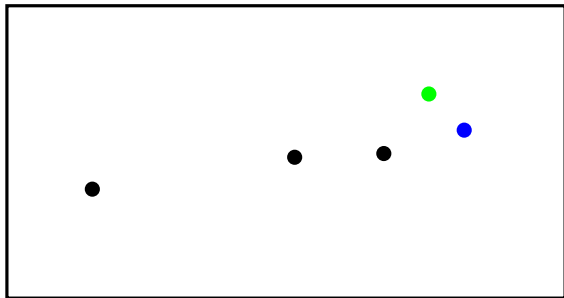
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

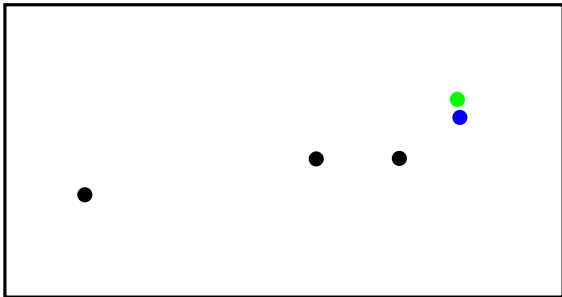
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

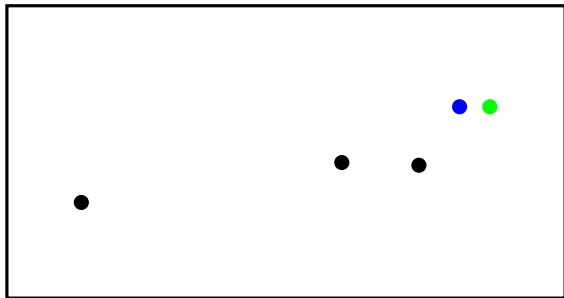
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

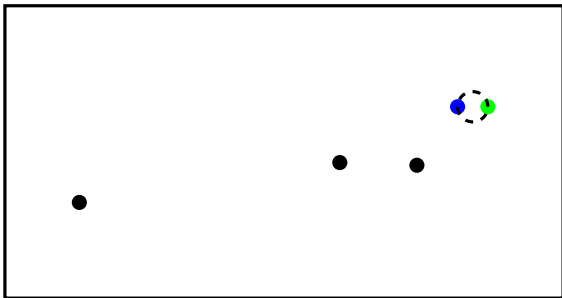
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

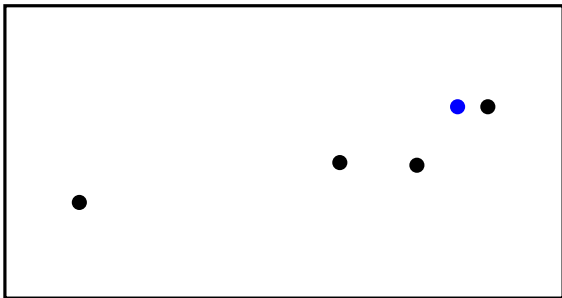
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: starting with 3 blues, 2red



Program:

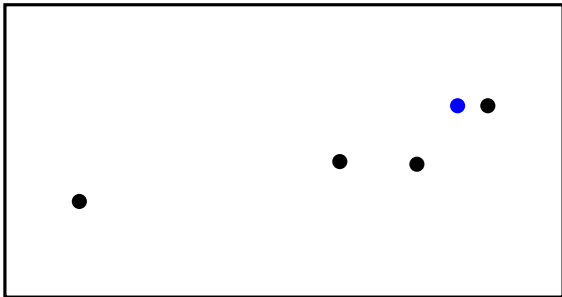
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 1: Final result



Program:

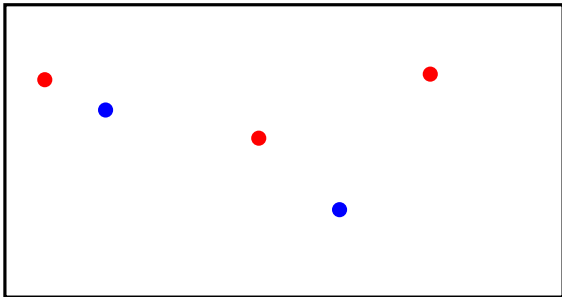
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

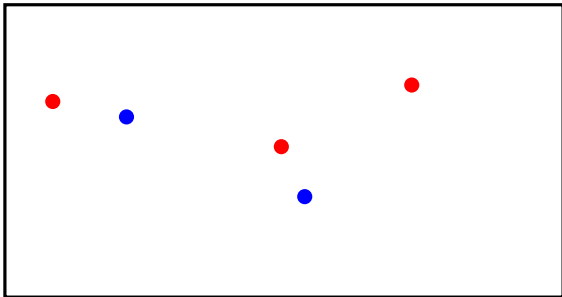
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

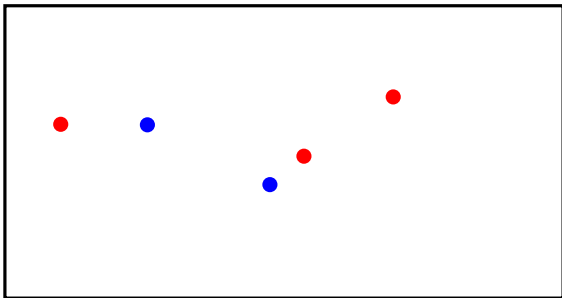
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

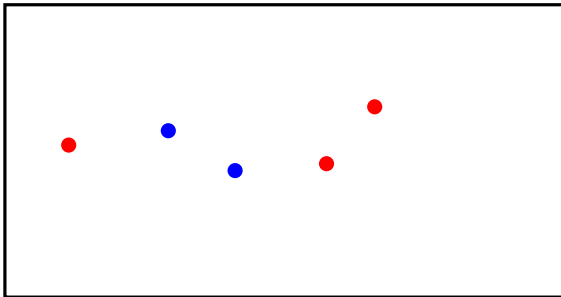
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

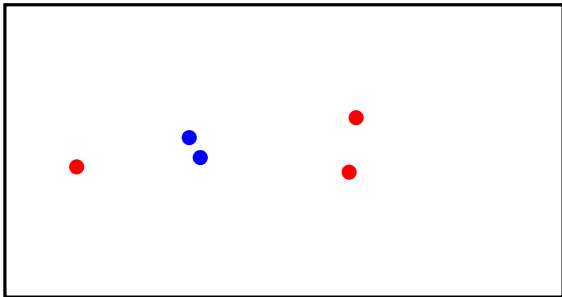
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

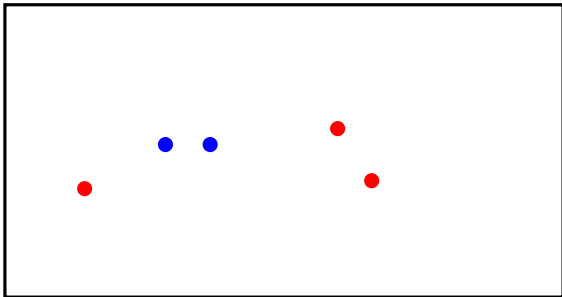
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

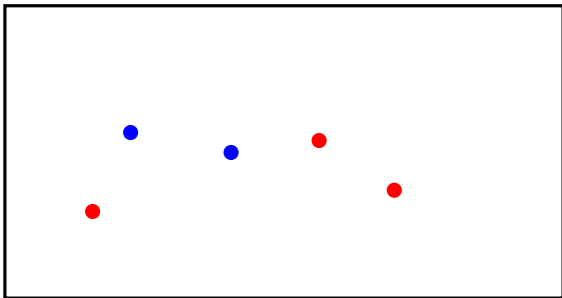
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

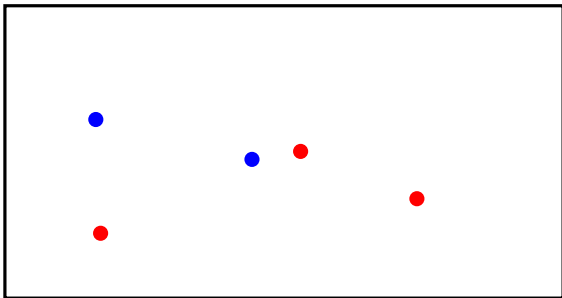
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

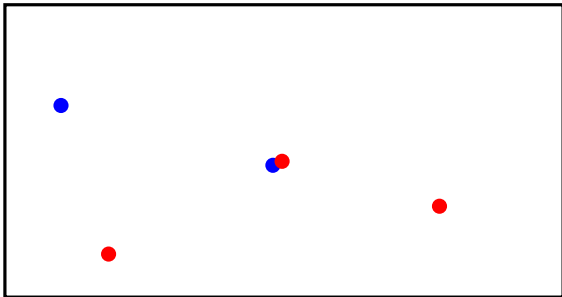
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

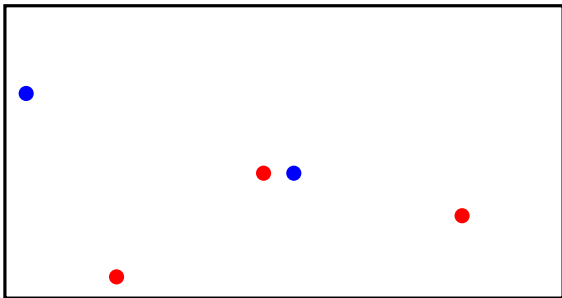
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

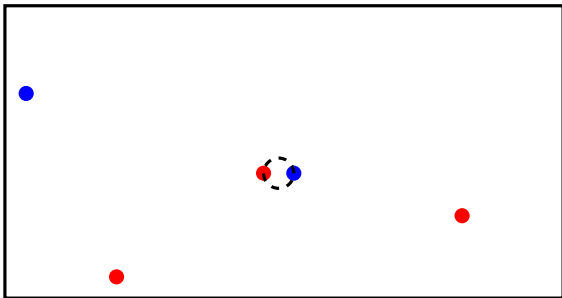
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

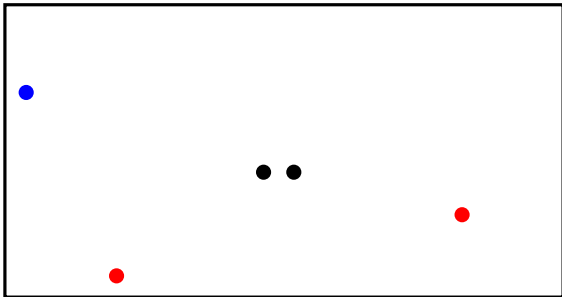
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

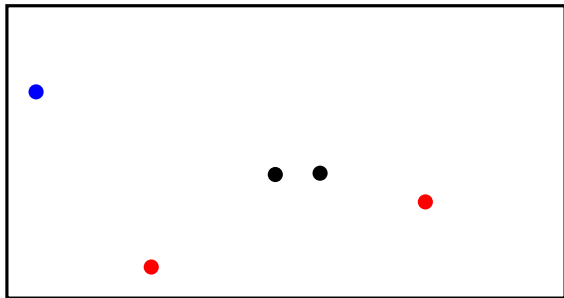
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

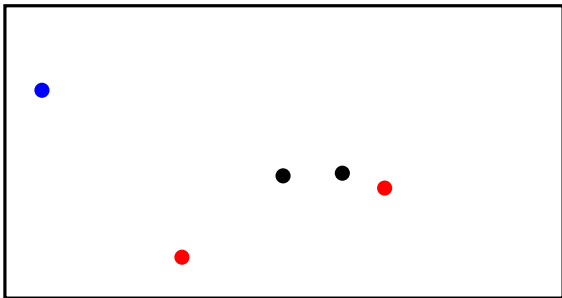
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

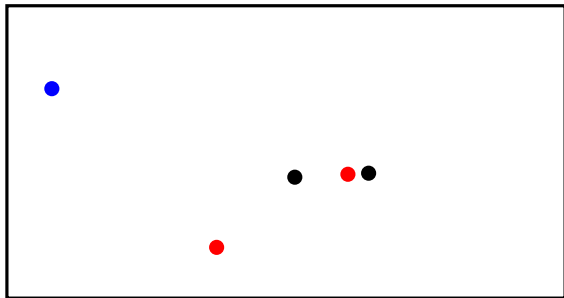
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

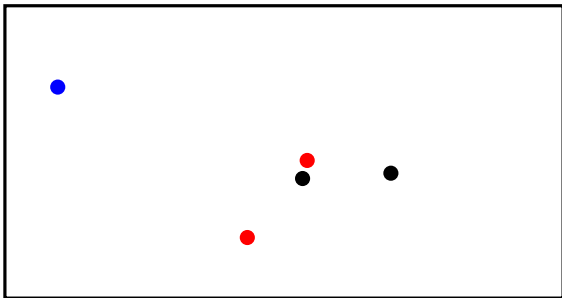
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

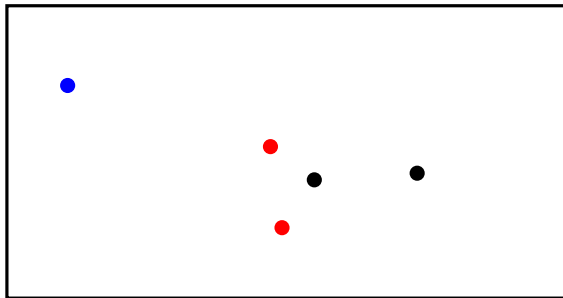
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

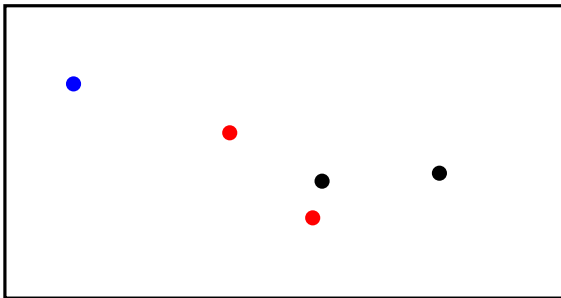
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

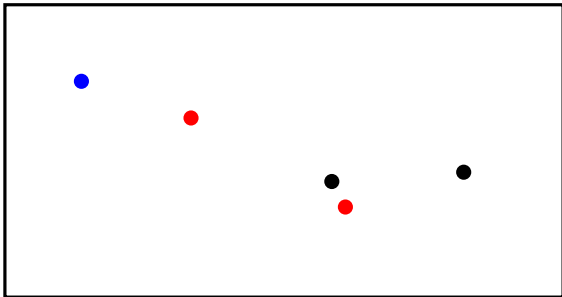
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

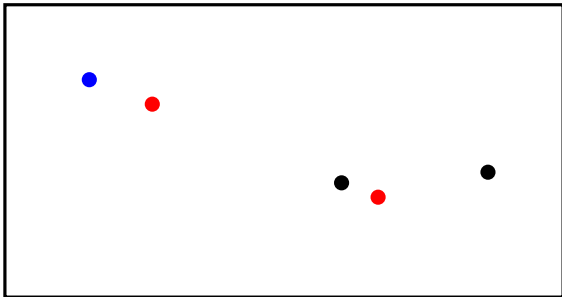
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

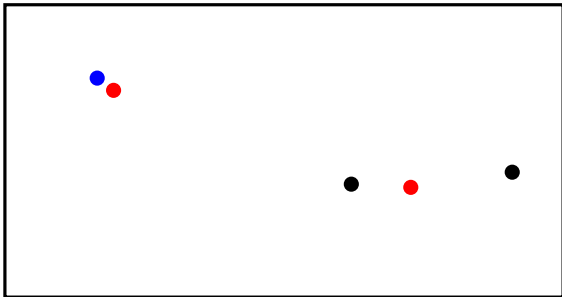
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

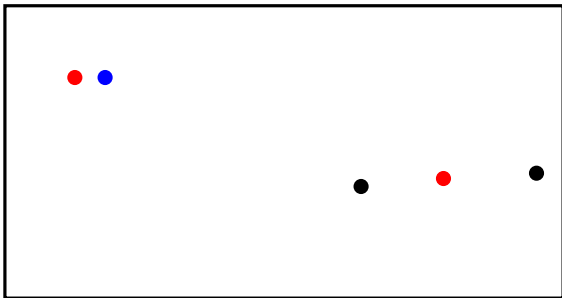
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

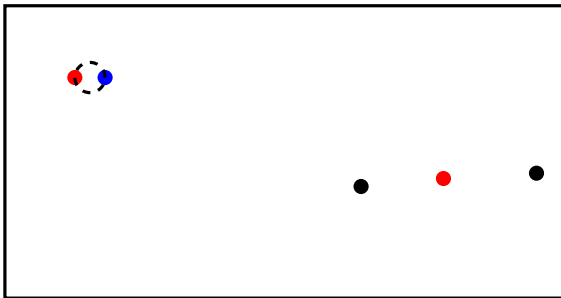
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

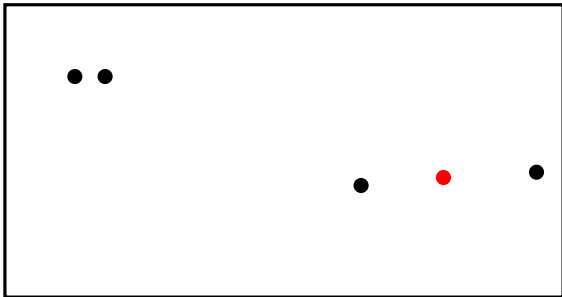
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

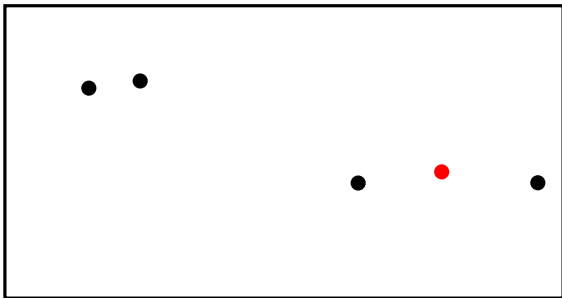
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

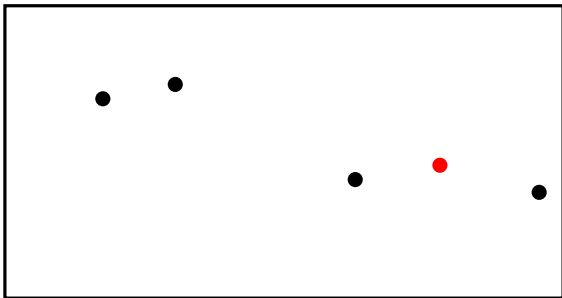
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

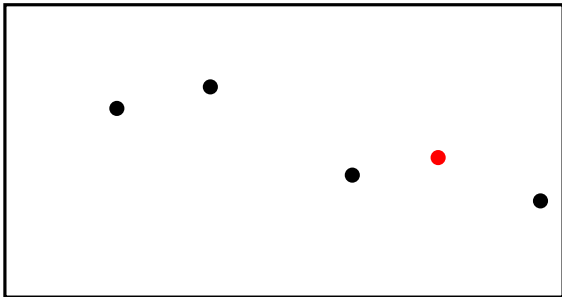
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

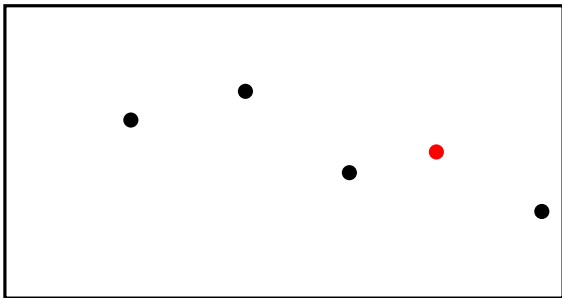
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3 red



Program:

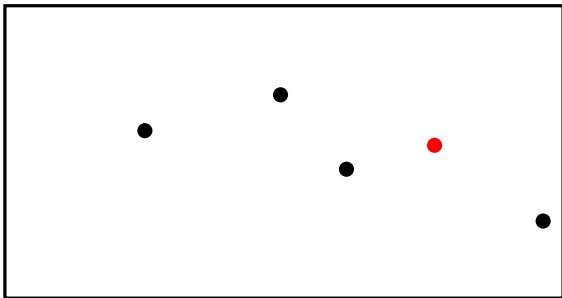
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3 red



Program:

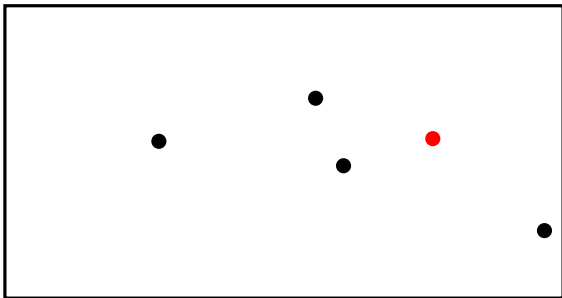
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

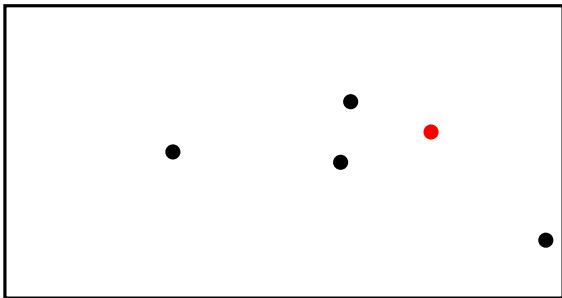
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

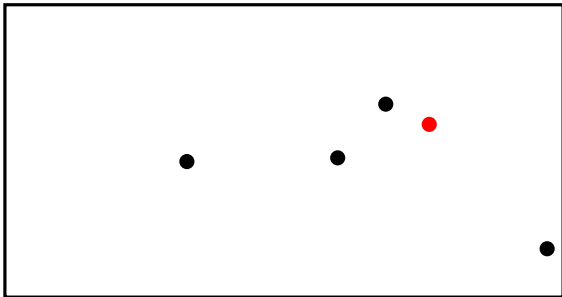
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

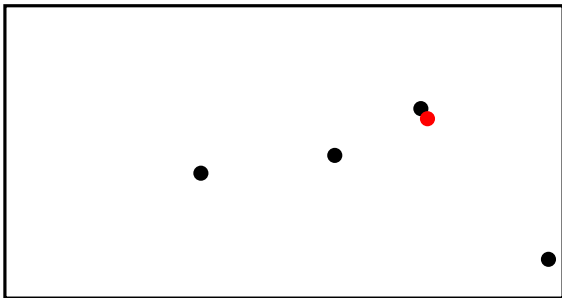
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

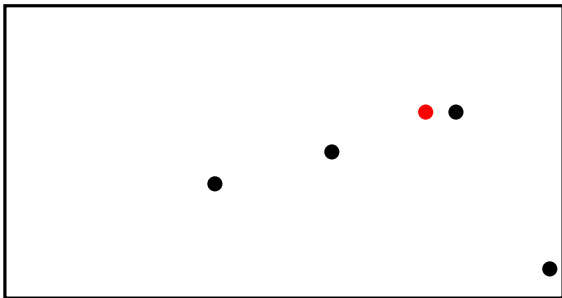
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3 red



Program:

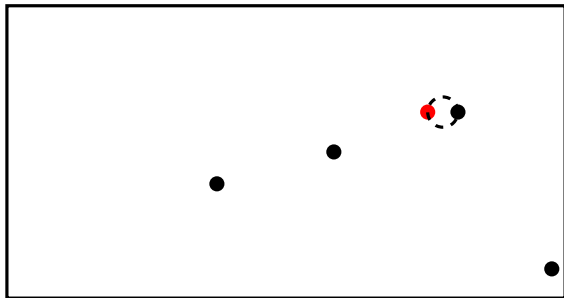
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

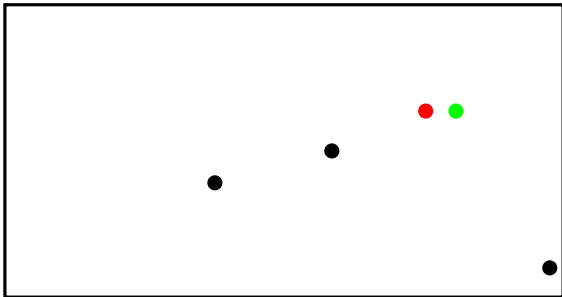
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3 red



Program:

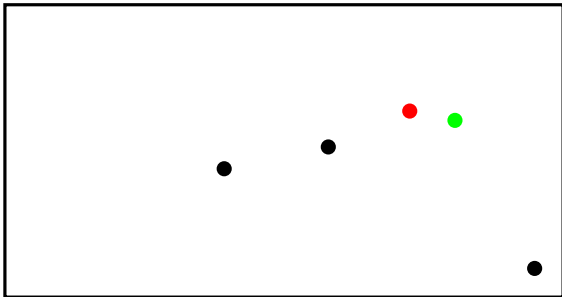
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

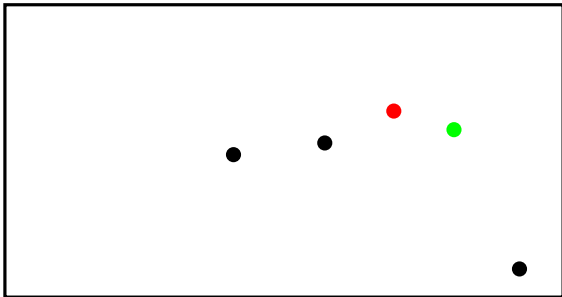
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

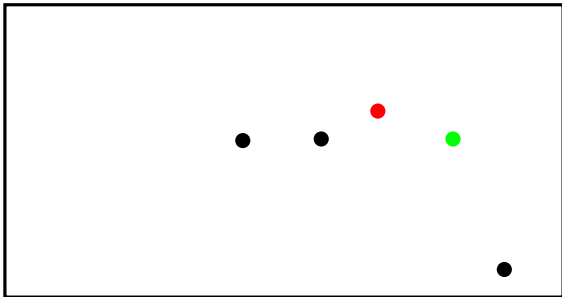
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

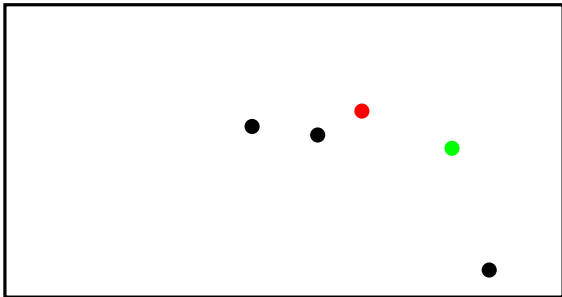
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

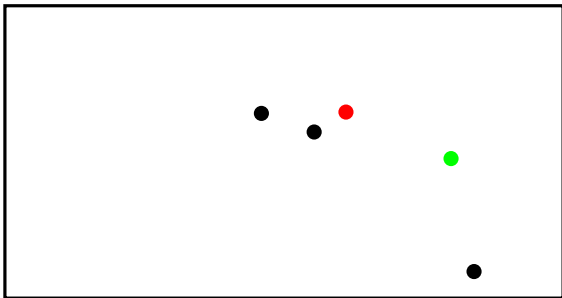
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

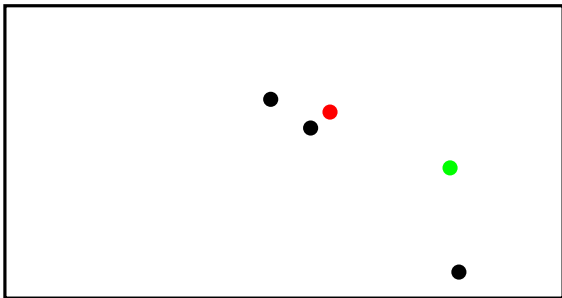
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

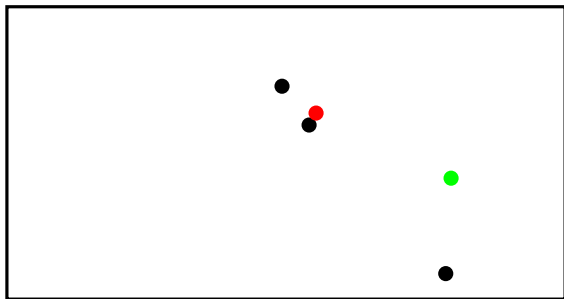
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

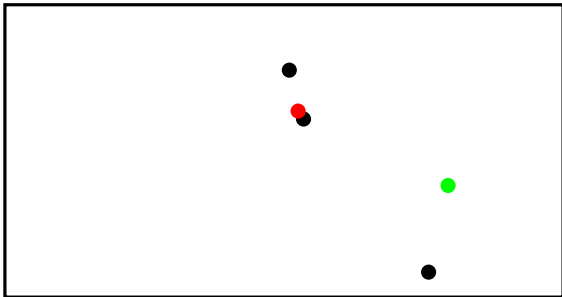
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

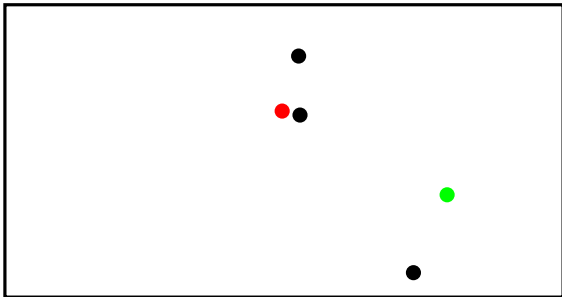
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

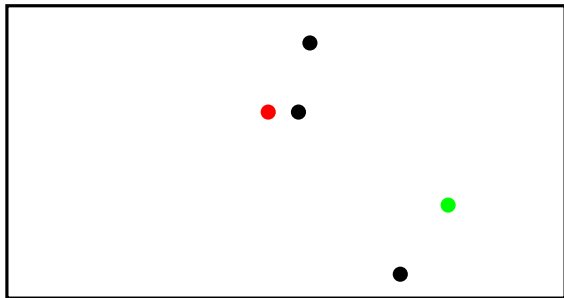
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3 red



Program:

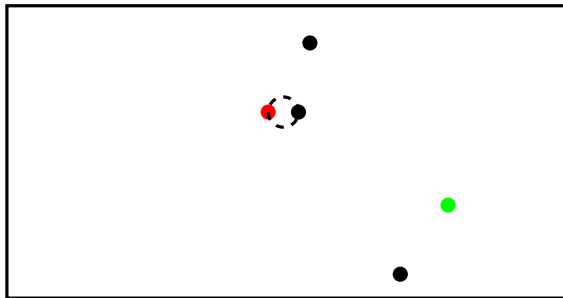
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

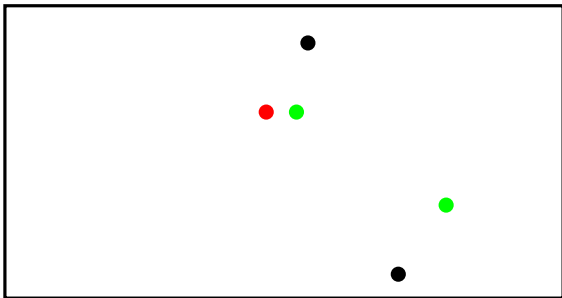
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

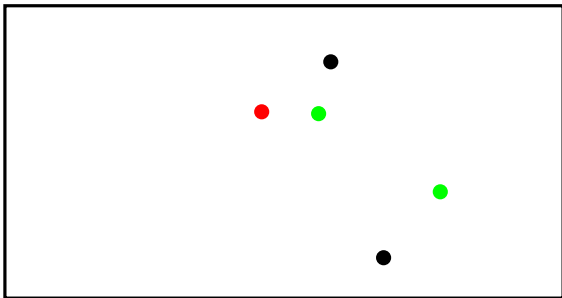
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

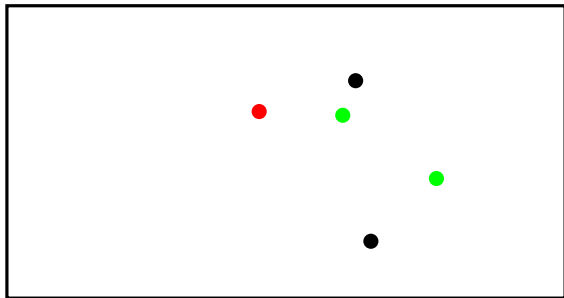
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

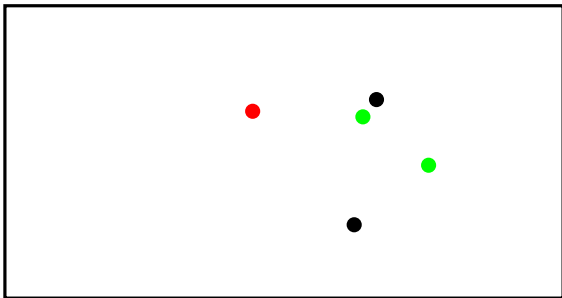
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

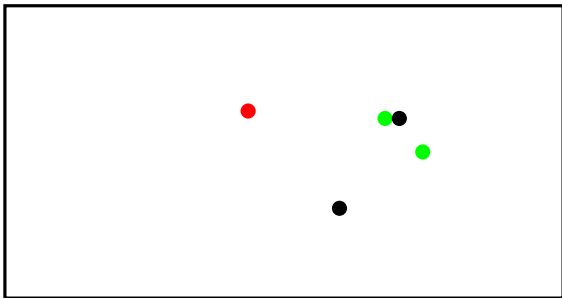
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

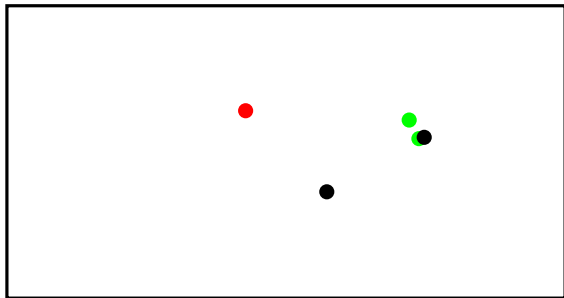
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

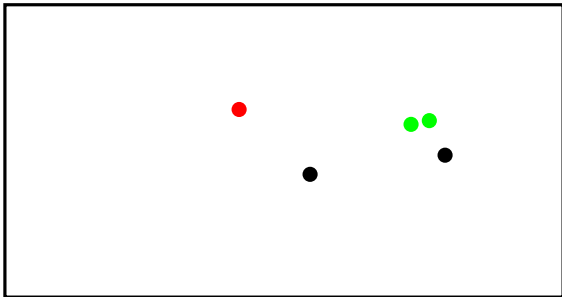
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

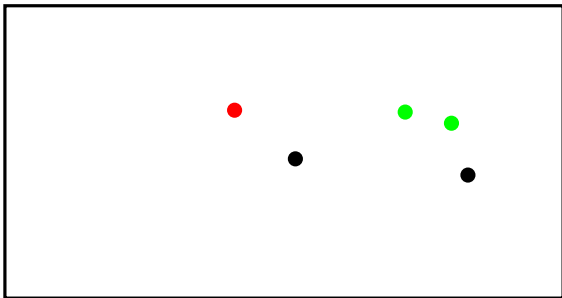
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

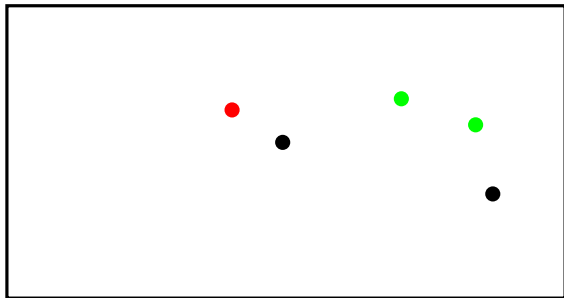
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

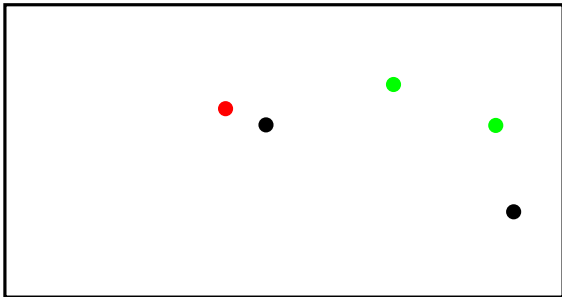
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

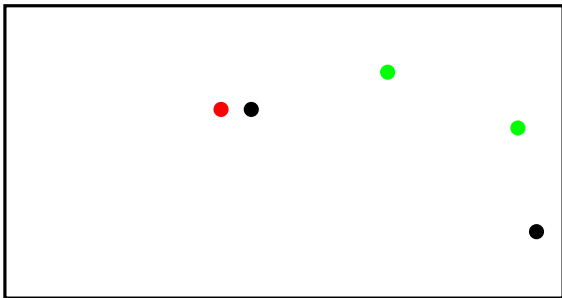
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

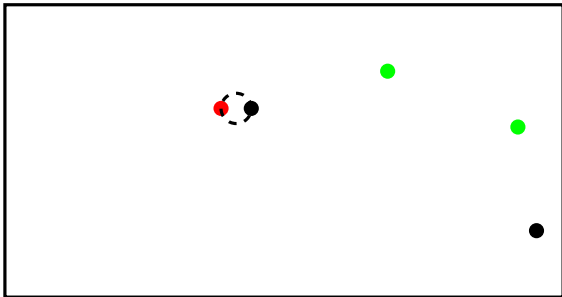
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

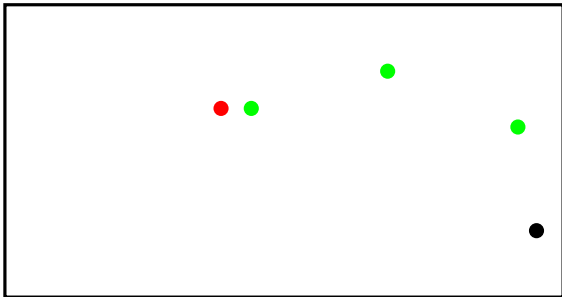
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

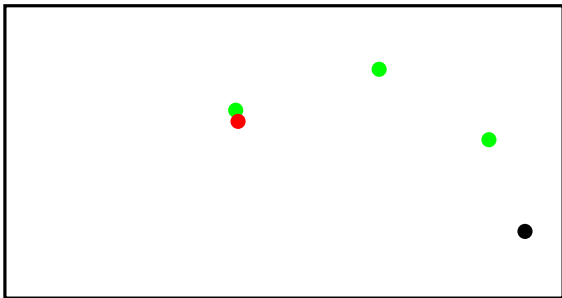
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

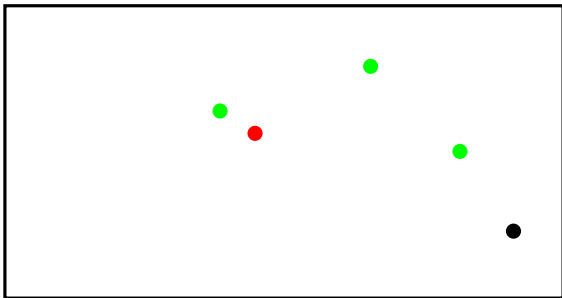
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

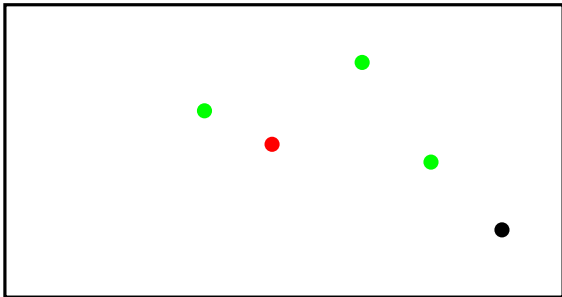
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

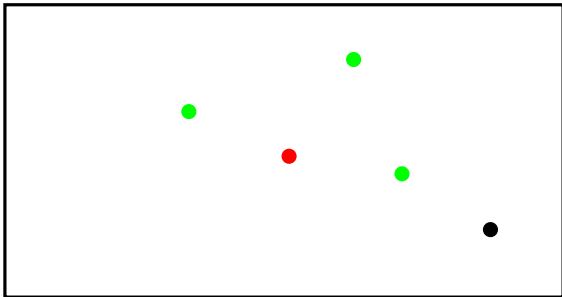
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

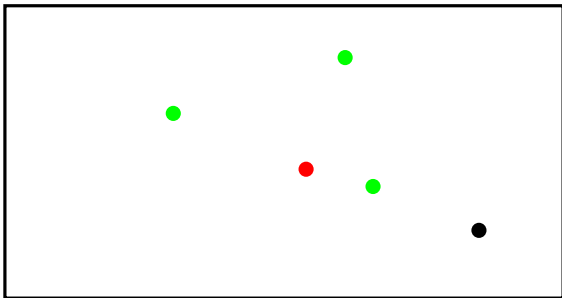
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

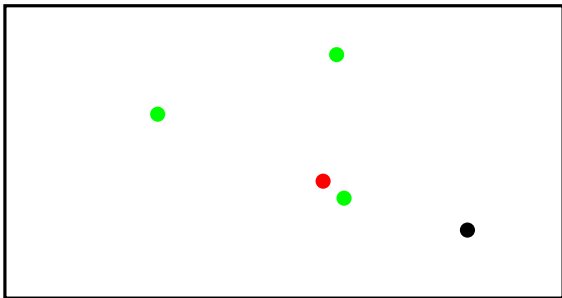
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

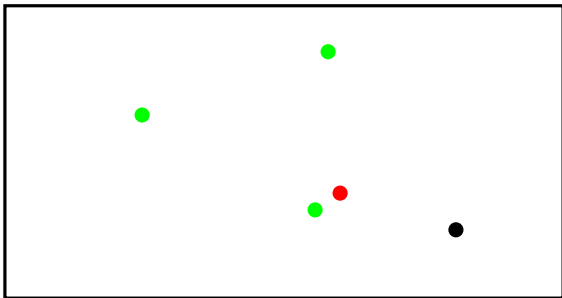
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

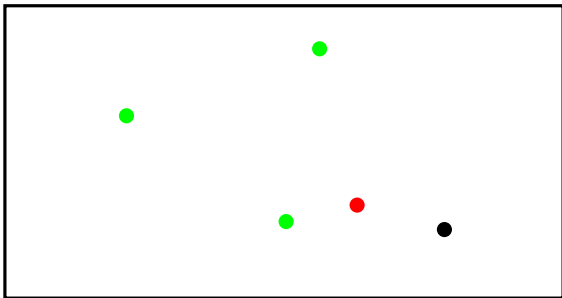
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

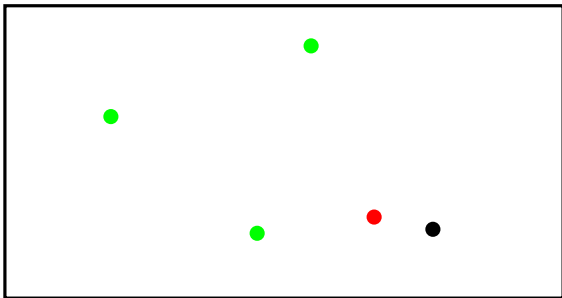
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

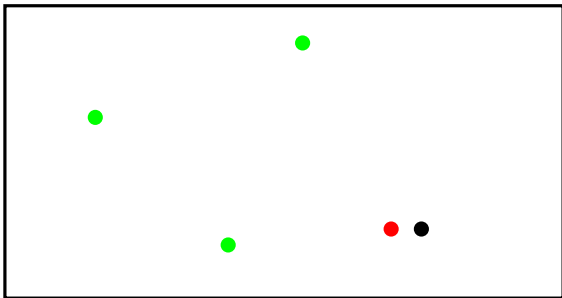
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

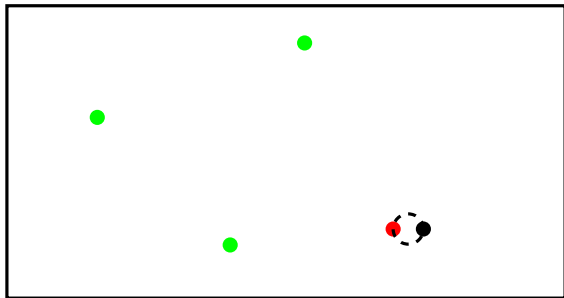
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

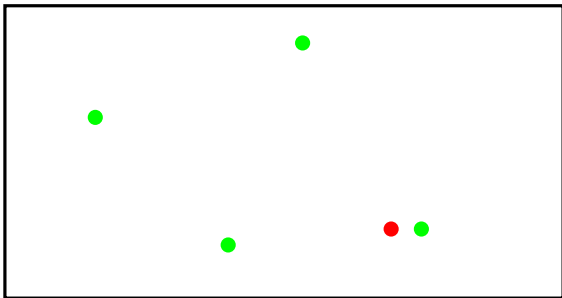
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: starting with 2 blues, 3red



Program:

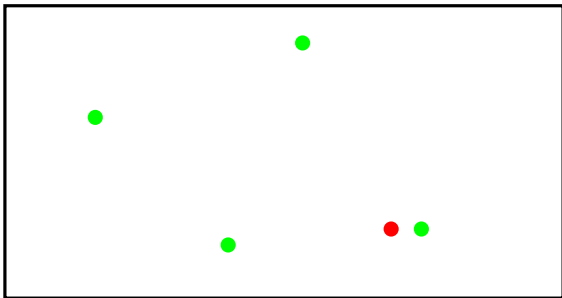
● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

Example 2: Final result



Program:

● ● → ● ●

● ● → ● ●

● ● → ● ●

● ● → ● ●

What is computed?

Let's interpret

- ● and ● by *yes*.
- ● and ● by *no*.
- Whatever the initial state is, ultimately, all agents will agree.
- They agree on *yes* iff the initial population of ● is strictly greater than the initial population of ●.

What is computed?

Let's interpret

- ● and ● by *yes*.
- ● and ● by *no*.
- Whatever the initial state is, ultimately, all agents will agree.
- They agree on *yes* iff the initial population of ● is strictly greater than the initial population of ●.
- Hence, it is computing **majority**,

What is computed?

Let's interpret

- ● and ● by *yes*.
- ● and ● by *no*.
- Whatever the initial state is, ultimately, all agents will agree.
- They agree on *yes* iff the initial population of ● is strictly greater than the initial population of ●.
- Hence, it is computing **majority**,
i.e.

$$n_{\bullet} > n_{\bullet},$$

What is computed?

Let's interpret

- ● and ● by *yes*.
- ● and ● by *no*.
- Whatever the initial state is, ultimately, all agents will agree.
- They agree on *yes* iff the initial population of ● is strictly greater than the initial population of ●.
- Hence, it is computing **majority**,
i.e.

$$n_{\bullet} > n_{\bullet},$$

that is to say

$$n_{\bullet} - n_{\bullet} > 0.$$

Classical Population Protocols: Computable Predicates

Theorem (Angluin et al. 2006)

A predicate is computable iff it is on the following list:

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

Alternate Characterization: Presburger Arithmetic

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

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

Note: no multiplication.

Examples:

majority: $x_0 < x_1$

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

at least 40%: $x_0 + x_0 < x_1 + x_1 + x_1$

Menu

Inspiration: Classical Population Protocols

Large Population Protocols

Discussion and Conclusion

Population protocols with large populations

We want to consider

- a large 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)

Sub-menu

Large Population Protocols

My favorite example

The model

Convergence to an ordinary differential equation

Asymptotic Development

Computing with large population protocols

Our main result

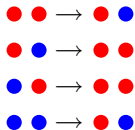
A scenario

- Consider a population of individuals provided with mobile phones, moving in a somewhat restricted environment.
- Whenever two individuals meet, i.e., whenever they get within the transmission range of each-other, their respective mobile devices get connected, and potentially exchange data of common interest.
- In order to limit the amount of energy spent in such a process, one may want to limit the frequency of exchanges between nodes:
 - ▶ one could let two connected nodes exchange information only if they are both in some specific state, called *active*
 - ▶ say we want the frequency (or probability) of exchanges to be $p = \frac{1}{2}$

An example

States: $\{\bullet, \bullet\}$

Rules of interactions:

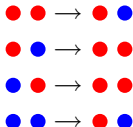


What can we say about

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

Exercise 0: when the number n of agents is fixed.

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

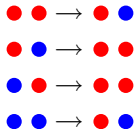


Exercise 0: When the number n of agents is fixed: continued

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

$E[p_{\bullet}]$ will converge to some (computable) rational value.

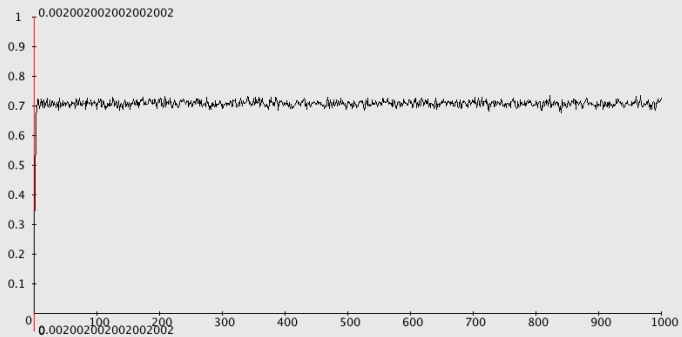
When the number n of agents is large?



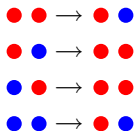
- What is the limit of p_{\bullet} when n goes to infinity?

données **graphe** histogramme

- moyenne
- + $\frac{V}{V}$
- $\frac{V}{V}$



An informal approach



- The mean number of ● created,

$$\begin{aligned} b(p_{\bullet}) &= -1 * p_{\bullet}^2 + 1 * p_{\bullet}(1 - p_{\bullet}) + 1 * p_{\bullet}(1 - p_{\bullet}) + 1 * (1 - p_{\bullet})^2 \\ &= 1 - 2p_{\bullet}^2 \end{aligned}$$

- must be equal, at the limit to 0,
- and hence

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

at the limit.

Computing with large populations

- In other words,

this protocol **computes** real number $\frac{\sqrt{2}}{2}$.

- Which real numbers can be computed in the above sense?

A scenario

- Consider a population of individuals provided with mobile phones, moving in a somewhat restricted environment.
- Whenever two individuals meet, i.e., whenever they get within the transmission range of each-other, their respective mobile devices get connected, and potentially exchange data of common interest.
- In order to limit the amount of energy spent in such a process, one may want to limit the frequency of exchanges between nodes:
 - ▶ one could let two connected nodes exchange information only if they are both in some specific state, called *active*
 - ▶ say we want the frequency (or probability) of exchanges to be $p = \frac{1}{2}$

Sub-menu

Large Population Protocols

My favorite example

The model

Convergence to an ordinary differential equation

Asymptotic Development

Computing with large population protocols

Our main result

General Case: Algorithm

An algorithm consists of

- a finite set of states $Q = \{q_1, q_2, \dots, q_k\}$.
- transition rules, mapping pairs of states to pairs of states

Executions are given by:

- instantaneous configuration: $(x_1, x_2, \dots, x_k) \in [0, 1]^k$, with $\sum_i x_i = 1$.
- transitions between configuration: two agents are picked, and updated according to the rules of the algorithm.
- some states are marked.
- we want to consider the proportion of agents in marked states.

Remark: Algorithm are assumed independent of size of population!

Rescaling...

- Let $Y_i^{(n)}(t)$ be the random variables equal to the numbers of agents in state q_i at time t in a population of size n .
- Let

$$\bar{Y}_i^{(n)}(t) = \frac{1}{n} \cdot Y_i^{(n)}(t).$$

$$\bar{Y}^{(n)}(t) = (\bar{Y}_1^{(n)}(t), \dots, \bar{Y}_{|Q|}^{(n)}(t)).$$

$$X^{(n)}(t) = \bar{Y}^{(n)}(\lfloor nt \rfloor) + (nt - \lfloor nt \rfloor) \cdot \left(\bar{Y}^{(n)}(\lfloor nt + 1 \rfloor) - \bar{Y}^{(n)}(\lfloor nt \rfloor) \right).$$

- By definition,
 - ▶ $X^{(n)}(t)$ is a continuous-time Markov chain obtained by linear interpolation of $\bar{Y}^{(n)}$ with a time-acceleration of factor n .
 - ▶ After rescaling, the number of interactions occurring in one time unit is proportional to the number of agents in the population.

Sub-menu

Large Population Protocols

My favorite example

The model

Convergence to an ordinary differential equation

Asymptotic Development

Computing with large population protocols

Our main result

Convergence to an ordinary differential equation

Proposition

For any initial conditions $Y^{(n)}(0)$ such that $Y^{(n)}(0) \rightarrow x$ when $n \rightarrow \infty$, the sequence of random processes $X^{(n)}$ converges in law to the solution of the stochastic differential equation (with degenerated brownian motion):

$$dX(t) = b(X(t))dt \quad \text{with } X(0) = x. \quad (1)$$

where

■ $b : \mathbb{R}^{|\mathcal{Q}|} \rightarrow \mathbb{R}^{|\mathcal{Q}|}$ is the “balance” equation

► Formally:

$$b(x) = \sum_{(q_1, q_2) \in \mathcal{Q}^2} \left(x_{q_1} x_{q_2} \left(-e_{q_1} - e_{q_2} + \sum_{(q_3, q_4) \in \mathcal{Q}^2} \delta_{q_1, q_2, q_3, q_4} (e_{q_3} + e_{q_4}) \right) \right) \quad (2)$$

where $\delta_{q_1, q_2, q_3, q_4} = 1$ if $q_1, q_2 \rightarrow q_3, q_4$, and 0 otherwise.

Sub-menu

Large Population Protocols

My favorite example

The model

Convergence to an ordinary differential equation

Asymptotic Development

Computing with large population protocols

Our main result

Asymptotic Development

Theorem

Assume that the ordinary differential equation $dX(t)/dt = b(X(t))$ has a (exponentially) stable equilibrium $b(x^*) = 0$.

Then there exists a neighborhood of x^* such that, whenever $\bar{Y}^{(n)}(0)$ belongs to this neighborhood, we have

$$\bar{Y}^{(n)}(t) \approx x^* + \frac{1}{\sqrt{n}} \mathcal{N}(0, \chi)$$

when $t \rightarrow \infty$,

where $\mathcal{N}(0, \chi)$ is the centered gaussian distribution with covariance matrix χ , for some χ .

Sub-menu

Large Population Protocols

My favorite example

The model

Convergence to an ordinary differential equation

Asymptotic Development

Computing with large population protocols

Our main result

Computation

A real number ν is said to be computable if

- there exists a LPP \mathcal{P} , admitting finitely many equilibria,
- some of these finitely many equilibria $x^* = (x_1, x_2, \dots, x_k) \in [0, 1]^k$ is stable
- and $\sum_{q_i \in Q^+} x_i = \nu$ where Q^+ is the set of marked states for \mathcal{P} .

Sub-menu

Large Population Protocols

My favorite example

The model

Convergence to an ordinary differential equation

Asymptotic Development

Computing with large population protocols

Our main result

Main result







Theorem

ν is computable by an LPP

if and only if

$\nu \in [0, 1]$ is algebraic.

Sketch of proof

- Direct sense: 
 - ▶ A computable number ν must correspond to a x^* with $b(x^*) = 0$.
 - ▶ b componentwise polynomial implies ν algebraic.
- Undirect sense: 
 - ▶ All rational numbers are computable. 
 - ▶ Derandomization: probabilistic large population protocols can be simulated by large population protocols. 
 - ▶ Constructing equilibria. 
 - ▶ Enforcing stability. 

Menu

Inspiration: Classical Population Protocols

Large Population Protocols

Discussion and Conclusion

Conclusion

- We proposed a formalization of population protocols with large populations.
- We proved convergence in the general case.
- We gave an asymptotic development of the convergence in some cases.
- We characterize all computable real numbers to be precisely algebraic numbers of $[0, 1]$.

Current and future work

- Variants on the model.
- Going further:
 - ▶ We characterize computable numbers.
 - ▶ Can we characterize computable predicates?

Sub-menu

Sketch of proof

Direct sense

Lemma

For every $\nu \in [0, 1]$, if ν is computable by a LPP then ν is algebraic.

- a computable number must correspond to an equilibrium x^* of b : i.e. $b(x^*) = 0$.
- b is componentwise polynomial
- any solution of a system of polynomial equations which has finitely many solutions is component-wise algebraic.

Undirect sense: step 1

Computing all rationals

Lemma

Let $\nu \in [0, 1]$ be a rational number. There exists a LPP computing ν .

- To compute $1/k$, consider the protocol \mathcal{M} over states $Q_k = \{1, \dots, k\}$ given by the following transitions rules:

$$i j \rightarrow (i + 1) (j + 1)$$

- To compute $p/q \in \mathbb{Q}$, set marked states in an appropriate way.

Undirect sense: step 2

Derandomization

We focus on probabilistic LPPs, where the transition rules are defined by

$$q_i q_j \rightarrow \alpha_{i,j,k,l} q_k q_l$$

where all $\alpha_{i,j,k,l}$ are rational numbers.

Lemma

Let $\nu \in [0, 1]$, and assume that there exists a probabilistic LPP computing ν , with rational probabilities. Then there exists a (deterministic) LPP computing ν .

Undirect sense: step 3

Constructing Equilibria

Lemma

For any algebraic number $\nu \in [0, 1]$, there exists $\delta \in \mathbb{N}$, $\lambda \in \mathbb{Q}$, and a protocol \mathcal{P} such that $(\nu, \lambda\nu^2, \lambda^2\nu^3, \dots, \lambda^{\delta-2}\nu^{\delta-1}, 1 - \sum_{i=1}^{\delta-1} \lambda^{i-1}\nu^i)$ is an equilibrium of \mathcal{P} .

Undirect sense: step 3

Constructing Equilibria

Lemma

For any algebraic number $\nu \in [0, 1]$, there exists $\delta \in \mathbb{N}$, $\lambda \in \mathbb{Q}$, and a protocol \mathcal{P} such that $(\nu, \lambda\nu^2, \lambda^2\nu^3, \dots, \lambda^{\delta-2}\nu^{\delta-1}, 1 - \sum_{i=1}^{\delta-1} \lambda^{i-1}\nu^i)$ is an equilibrium of \mathcal{P} .

- Let $\nu \in (0, 1]$ be an algebraic number, and let $P(X) = \sum_{i=0}^{\delta} a_i X^i$, be a polynomial such that $P(\nu) = 0$, and $P(0) > 0$.
- Build \mathcal{P}_ϵ with

$$dx_1 = \epsilon \left(a_0 + a_1 x_1 + \sum_{i=2}^{\delta-1} \frac{a_{i+1}}{\lambda^{i-1}} x_{i-1} x_1 \right)$$

$$dx_i = \lambda x_1 x_{i-1} - x_i \quad \text{for every } i = 2, \dots, \delta - 1$$

$$dx_\delta = - \sum_{i=1}^{\delta-1} dx_i.$$

where $\lambda > 0$, and $\sum_{i=1}^{\delta-1} \lambda^{i-1} \nu^i \leq 1$.

Undirect sense: step 4

Enforcing stability

Lemma

Every algebraic number $\nu \in [0, 1]$ is computable by an LPP.

◀ back

Undirect sense: step 4

Enforcing stability

Lemma

Every algebraic number $\nu \in [0, 1]$ is computable by an LPP.

- Establish stability using Routh-Hurwitz criterion. . .
- by choosing ϵ sufficiently small.

◀ back