ÉCOLE POLYTECHNIQUE

Laboratoire d'Informatique
de l'École Polytechnique

École Doctorale de l'École Polytechnique

THÈSE DE DOCTORAT

par **Mikaël Rabie**

soutenue le **31 août 2015**

pour obtenir le grade de **Docteur de l'École Polytechnique**
discipline **Informatique**

The Power of Weaknesses: What can be computed with Populations, Protocols and Machines

Contents

Part I

A Visit of the Infrastructure

Introduction

A Message Through Time

Dear you,

If you read this, it means that this thesis has not fallen (yet) into oblivion. This document, all these pages, (or this pdf, or this parchment, or this form of support that I could not even imagine the existence of) constitutes my message.

Can you imagine that thousands of years ago, to send a message to someone - apart from screaming at the top of your lungs - you had to walk (or ride, or run a marathon and drop dead) in order to tell him directly? To transmit an information rapidly, you had to hope the recipient was close by.

Hopefully, humanity looked for quicker ways to send messages. About three thousands years ago (if you read this document in 2015), people invented a way to use pigeons. Homing pigeons. This system had some limitations, as pigeons were only capable of returning to their initial home. It was used for example during the Greek Olympic games to announce the results. But even millennia later, as recently as World War II, this system of communication has been reused.

Pigeons was not the only way to carry messages. The postal system is much older than one might think. Using an intermediate service to transmit messages over long distances was already used in Ancient Egypt to broadcast new laws. The Persians apparently developed it in order that any citizen could reach any other throughout the empire.

Do you know the following paradox? Two generals plan to attack together a common enemy, one coming from south, the other from north. To have a chance of success, they need to engage battle at the same time. How to communicate an exact hour to strike and be sure that the other general will be there?

If you send a message setting a given hour and want to be sure that your ally has read it, you need your ally to send back an acknowledgment of receipt. But how could he be sure that you received it in return? By sending an acknowledgment of receipt back to you? It will be a loop with no end.

To transmit an information quasi-instantaneously, people had to be close enough. An Indian tribe could speak to another through smoke signals. Towns and castles used alarm systems to warn rapidly of pending danger.

To broadcast information, there were town criers and announcements written and published (on tracts or walls). Communication made a big step forward with the mastering of mechanical and electrical techniques. Broadcasting information through long distances instantly became possible via radios and TVs.

The invention of Morse code, then phone offered the possibility of instant communication between two persons. With the wiring of the whole planet, people at opposite ends of the world could converse together simultaneously.

Internet and smartphones even permitted to think bigger, with the capacity to reach more people quickly. Not only written messages can be send directly today, but also images, documents, videos...

Yet - today - we continue to use and rely upon old tools. Alarm sirens are tested every months to be sure that such warning systems still work. A wink still has the same purpose and the same interpretation. To avoid malicious interceptions of messages during wars, pigeons might make a comeback!

When I talk of ”today”, I have to admit I am being a bit self centered. This is my today - I know nothing of how yours might be. Perhaps in your day teleportation will have become a common mean of transportation, or maybe it is possible to engage in mental discussions? And if it is possible to communicate backward in time, please, make sure to warn me!

Did you know that in my days, scientists were already working on sending messages to future generations? In 2015, tons of nuclear waste are being buried. Wastes that need to stay under ground for hundreds of thousands of years. This raises a question: How can we make sure that future generations do not dig there too early? The Latin alphabet could no longer be used or even understood. The aim is to think up some universal warning message that will still be understood in one hundred millennia.

What will communication tools be in the future? What if innovation declines at some point? I hope that you found ways to reuse past tools. I hope that even with weaker tools than today’s, you are able to do a lot!

Sincerely yours,
Mikaël Rabie

Motivation

As I have tried to explain, communication has always been a strong tool for civilizations. Its uses are myriad, and meets incredibly variable needs. Some messages need to be sent instantaneously, for example to synchronize informations (like with the case of the paradox of the two generals). But, even nowadays, preforming synchronization among a great number of individuals is an extremely difficult task.

As people are not necessarily plugged into devices 24 hours a day, broadcasting messages is also a powerful tool. It is important to be able to perform broadcasting as fast as possible, trying to loose the least information possible. Identification is a strong tool. If we communicate to the right persons, we can share messages efficiently. But sometimes, it is hard to differentiate everyone. When we do not know the whole population, we can give an identity that is unique locally but can be shared with people farther away.

Today, we are alerted to leak issues. The increase of technology increases the risk that our data can be read by external people. Putting private data in the cloud, for example, has enabled some people to hack it and access to private pictures, messages... Sometimes, people will prefer to use weaker tools, which they are capable to understand, and so be able trust in what they are using. Anonymity also helps people to feel safer.

If we want to develop tools that can be used in the distant future, the most advanced technology might not be the most relevant. We are already looking for simpler alternatives to gas and oil, as our resources are limited. Einstein said that World War IV's weapons might be sticks and stones. He implies that our current technologies may be destroyed. Finding ways to communicate and work with weaker tools might be useful for an unknown future far ahead of us.

In this document, we will discuss variants of models, using weakened tools. The primary objective is to investigate the potential power of computation with the imposition of new restrictions. Each time, motivation can be seen as an utility for today's and tomorrow's needs.

Population Protocol is a way to communicate pairwise with anonymous agents. It permits to everyone not to give too much information about themselves. This model is already known and can compute global functions about the population itself. For example, we can make boundaries tests on the size of the population, or find the proportion of ill individuals in a flock.

We will work on two restrictions within this model, corresponding to issues of trust. The first gives a more rational view of the interactions, as agents' interactions are seen as a game. When two players meet, they will try to improve their strategic choice if they lost too much. The second one is based upon a notion of trust, as it is more instinctive for agents to keep its opinion when they meet agents sharing the same opinion.

Community Protocol is a more powerful model than the previous one, as agents have now a unique identity. For example, it can be a phone number, and interactions can be seen as being performed by phones when two individuals are close enough. An input can be seen as a word composed of each element's input letter sorted according to the identifiers' order. This model has been shown to simulate computers using a small space of computation.

We will work on several restrictions within this model. The first is the question of speed. The simulations of computers are actually slow, and agents might want to find an answer quickly. The best way to communicate rapidly is with broadcasts of informations, requests... we want to have a reasonable number of broadcasts. We then look at what happens in a situation when identifiers are not unique. Finally, we have some studies of what happens when identities are no longer ordered.

Turing Machines correspond to the basic model of computer science. We are used to see what can be computed within a small space, or with a time restriction. Extensions of the model, with non determinism, oracles... have also been explored. We are used to compare other models to Turing Machines.

We will work on a new complexity approach. In the mechanism, changing the internal state has never be considered to be difficult (as far as we know). We will look at what happens when this operation can happen only a finite number of times, whatever the input is. The idea behind this is that the mechanism is rusted, and can only go in one way.

Organization of the Document

Chapter 1

The first chapter is here to present the three models that will be explored in this document. The three models are Population Protocols, Community Protocols and Passively Mobile Protocols.

In addition to the definitions, this chapter will include several examples and the most important results known. This chapter is essential to understand the landscape of models we will introduce and study in the next chapters and gives results used in later chapters.

Weakening of Population Protocols

We introduce two new restrictions of the model of Population Protocols.

Chapter 2

We show how to interpret population protocols' interactions as a two player symmetric games in Chapter 2, with ***Pavlovian Population Protocols***.

We first provide some results on variations of the model, in order to motivate our choices with the definitions of pavlovian population protocols. With asymmetric games, the model is as strong as the original one. If players choose the best strategy against the one they faced, without changing it if they lost, the model cannot check if there are at least 3 agents.

We prove that with pavlovian population protocols, it is possible to count if there are at least 3, or even 2^k agents in a given input state. Some other protocols are given. No characterization of what can or cannot be computed has been found, but the algorithm for counting at least 2^k agents proves that programming with this model is not at all a trivial task.

Chapter 3

Chapter 3 is based on the intuitive idea that when two agents meet in a population protocol, if they share the same opinion, they should keep there opinion after the interaction. This is the ***Trustful Population Protocol*** model.

We introduce two versions, a *strong* and a *weak*. We got an exact characterization of what these restrictions compute. Both versions compute exactly the partitions such that each partitioned set can be defined as a boolean combination of 0-threshold predicates.

To obtain this result, we study the restriction where opinion and output are the same.

We prove that the model is frequency based, which means that a protocol accepts an input if and only it accepts this input multiplied by any positive integer. This result permits to prove that modulo predicates and some threshold predicates cannot be computed.

Finally, we study an extended version where agents can carry a Turing Machine (as in the Passively Mobile Protocols). We prove that, with enough space on each Machine, the proportions of each input symbol can be computed.

Weakening of Community Protocols and Some Extensions

Two new restrictions of Community Protocols have been studied.

Chapter 4

Using works about computing in $O(n \log^6 n)$ expected interactions with Population Protocols, we introduce in Chapter 4 the new class $\boldsymbol{CPPOLYLOG}$. It corresponds to what can be computed with Community Protocols with $O(n \log^k n)$ expected interactions for any $k > 0$.

We prove that some computable functions of Community Protocols are no longer computable. In particular, the rational language $(ab)^*$ is not in the set.

We provide some computable functions. The most important is the ability to encode the size of the population and use it for computation purpose. It permits to provide a tight comparison with Turing Machines.

Some set inclusions with known computational classes of classical complexity are also provided.

Chapter 5

Chapter 5 follows a new restriction: what happens with community protocols when a single identifier may be shared by several agents. This is the ***Homonym Population Protocol*** model.

We prove that with $\log n$ identifiers, if agents know when two identifiers are consecutive, it is possible to write the size of the population in binary, and it is possible to simulate a non deterministic Turing Machine on a tape of size $\log n$.

With $f(n)$ identifiers, we are able to create for any $k \in \mathbb{N}$, $f(n)^k$ identifiers, by the intermediate of k-tuples.

We provide a hierarchy depending on the number $f(n)$ of identifiers present in the population. The hierarchy is summarized by the following table:

$f(n)$ identifiers	Computational power
$O(1)$	Semilinear Sets Theorem 5.25
$\Theta(\log^r n)$ with $r \in \mathbb{R}_{>0}$	$\bigcup_{k\in\mathbb{N}} MNSPACE\left(\log^k n\right)$ Theorem 5.22
$\Theta(n^\epsilon)$ with $\epsilon > 0$	$MNSPACE(n \log n)$ Theorem 5.23
n	$NSPACE(n \log n)$ [GR09]

$MNSPACE(f(n))$ corresponds to non deterministic Turing Machines working on space $f(n)$ with an adapted input fit for this model.

Chapter 6

In Chapter 6, we provided a few results on some other variations.

Turing Machines with a Space of $O(\log \log n)$

Results with homonym population protocols permit to fill the gap of the question "What can be computed when each agent carries a Turing Machine with a space $O(\log \log n)$?" in the Passively Mobile Protocols.

We prove that with this space of computation, we compute exactly the union over k of the non-deterministic Turing Machines on tapes of length $\log^k n$ that have inputs stable under permutations.

Unordered Identifiers

In Section 6.2, we consider community protocols where identifiers cannot be compared. This time, agents can only do one action between two identifiers: check if there are equal or not.

Even if it looks to be too restrictive, it actually can still simulate a tape of size $O(n \log n)$. The only difference is that the input can only be seen as a multiset over the input alphabet. The computable sets are stable under permutation.

With the trick of the tuples used with homonym protocols, we are able to create more identifiers if there is homonymy.

Stack

We consider in Section 6.3 what happens when agents can carry a stack of tokens. We prove formally that this model can simulate any non deterministic Turing Machine, by showing how to handle a major problem with Population Protocols: When an agent finally starts to interact with the rest of the population.

Weakening of Turing Machines

Chapter 7

We introduce in Chapter 7 a new complexity element to consider in the theory of Turing Machines. What happens when we limit the number of times when we restrict the number of times a Turing Machine changes its internal state.

A Turing Machine transition is pivotal if it changes the internal state of the machine. We then work on RTM, the class of ***Rusted Turing Machines***, machines using a constant number of pivotal transitions, whatever the input is.

Even if the model looks weak, we provide machines that computes:

- The size of the input written in binary.
- The semilinear sets.
- Some languages of the form $(u+v)^*$.

We prove in this chapter that the model cannot be compared to regular languages, context-free languages, and that star height of computable regular languages is not bounded.

We prove that the halting problem for this class of machines is decidable. We then give an upper bound of the Busy Beaver problem of the model.

Chapter 1

Population, Community and Passively Mobile Protocols

We will introduce the models and the results used in this document. The models will be used in next chapters.

We first provide definitions and main results on Population Protocols, including the tools to compute quickly from [AAD$^+$04]. We then introduce the Community Protocols [GR09], and next the Passively Mobile Protocols [CMN$^+$11].

None of the results in this section are original. They can be found in the literature or can be deduced from.

1.1 Introduction

Population Protocols have been introduced by Angluin *et al.* in 2004 [AAD$^+$04]. They correspond to a model of devices with a weak memory interacting pairwisely to communicate and compute a global result. Predicates computable by population protocols have been characterized as being precisely the semi-linear predicates; which is equivalent to be definable in first-order Presburger arithmetic. Semi-linearity was shown to be sufficient in [AAD$^+$04], and necessary in [AAER07].

Many works on population protocols have concentrated latter on characterizing what predicates on the input configurations can be stably computed in different variants of the models and under various assumptions. Variants of the original model considered so far include restriction to one-way communications [AAER07], restriction to particular interaction graphs [AAC$^+$05]. Various kinds of fault tolerance have been considered for population protocols [DFGR06], including the search for self-stabilizing solutions [AAFJ05]. Works also includes the Probabilistic Population Protocol model that makes a random scheduling assumption for interactions [AAD$^+$04]. We refer to [AR07, CMS10] for a more comprehensive survey or introduction.

Some works extend this model. On the one hand, the edges of the interaction graph may have states that belong to a constant-size set. This model called the *mediated population* *protocol* is presented in [MCS11]. The addition on Non-Determinism has been studied in [BBRR12]. The research of Self-Stabilization (over the fairness assumption) has been explored in [BBK09]. An extension with sensors offering a *cover-time* notion was also studied [BBBD11, MCS12, BBB13].

The population protocol model shares many features with other models already considered in the literature. In particular, models of pairwise interactions have been used to study the propagation of diseases [Het00], or rumors [DK65]. In chemistry, the chemical master equation has been justified using (stochastic) pairwise interactions between the finitely many molecules [Gil92, Mur02]. The variations over the LOCAL model [FKL07, FKP11] can be seen as a restriction over the interactions (using a graph) but with a set of possible improvements in agents' capacities.

Among many variants of population protocols, the *passively mobile (logarithmic space) protocols* introduced by Chatzigiannakis *et al.* [CMN+11] constitutes a generalization of the population protocol model where finite state agents are replaced by agents that correspond to arbitrary Turing machines with $O(S(n))$ space per-agent, where n is the number of agents. An exact characterization [CMN+11] of computable predicates is given: this model can compute all symmetric predicates in $NSPACE(nS(n))$ as long as $S(n) = \Omega(\log n)$. Chatzigiannakis *et al.* establish that with a space in agents in $S(n) = o(\log \log n)$, the model is equivalent to population protocols, i.e. to the case $S(n) = O(1)$.

In parallel, *community protocols* introduced by Guerraoui and Ruppert [GR09] are closer to the original population protocol model, assuming *a priori* agents with individual very restricted computational capabilities. In this model, each agent has a unique identifier and can only remember $O(1)$ other agent identifiers, and only identifiers from agents that it met. Guerraoui and Ruppert [GR09] using results about the so-called storage modification machines [Sch80], proved that such protocols simulate Turing machines: Predicates computed by this model with n agents are precisely the predicates in $NSPACE(n \log n)$.

In this chapter, we will first introduce the Population Protocols model. We then gradually improve the power of individual agents by adding identifiers, with the Community Protocols model. We finish with a even more powerful model: the Passively Mobile Protocols model.

1.2 Population Protocols

1.2.1 Population Protocol Definition

We introduce population protocols [AAER07, AAD+04]: basically, this is a model of computation over a finite set of agents. In this model, each agent starts with an input in Σ and has its state in Q where Σ is a finite alphabet, and Q is a finite set of states. The population evolves by pairwise interactions. Each such interaction has the effect of (possibly) changing the state of the two agents according to some program, that is to say a function δ that maps couple of states (previous states of involved agents) to couple of states (next states of these agents).

A protocol computes a function (a predicate) from multisets over Σ to a finite set Y: encoding and decoding is given by two functions ι and ω. Given some input, that is to say a multiset over Σ, the initial state of the population is given by applying function $\iota : \Sigma \to Q$ element-wise. A computation can provide several outputs, each possible one being in Y. To interpret the output of an agent, it is sufficient to apply ω to its current state.

More formally:

Definition 1.1 *A* **Population Protocol** *is given by six elements $(Q, \Sigma, \iota, Y, \omega, \delta)$ where:*

- *Q is a finite set of states.*
- *Σ is the finite set of entry symbols.*
- *ι is a function $\Sigma \to Q$.*
- *Y is the finite set of possible outputs.*
- *ω is a function $Q \to Y$.*
- *δ is a function $Q^2 \to Q^2$.*

A ***Configuration*** is a multiset over Q. As agents are assumed to be anonymous, a configuration can also be described by only counting the number of agents in each state. In other words, a configuration, as it is a multiset, can also be considered as an element of $\mathbb{N}^{|Q|}$. An ***Input*** is a multiset of Σ (i.e. an element of $\mathbb{N}^{|\Sigma|}$). The initial configuration is computed by applying ι to each agent, i.e. apply ι elementwise to the input. A ***Step*** is the transition between two configurations $C_1 \to C_2$, where all the agents but two do not change: we apply to the two agents a_1 and a_2 the rule corresponding to their respective state q_1 and q_2, i.e. if $\delta(q_1, q_2) = (q_1', q_2')$ (also written by rule $q_1\ q_2 \to q_1'\ q_2'$), then in C_2 the respective states of a_1 and a_2 are q_1' and q_2'. All other agents have the same state in C_1 and C_2.

Remark 1.2

1. *Choosing Y arbitrary will be used in the Trustful Population Protocols Chapter (Chapter 3). In all other chapters, we will assume that $Y = \{True, False\}$, and will not even mention Y.*

2. *We suppose that our protocol rules are deterministic, as δ is assumed to be a function. The case where δ can transform a couple (p, q) into two (or more) different possible couples is quickly treated in Section 1.2.5.*

Example 1.3 (Leader Election) *This protocol is one of the most important protocol and will be used and revisited a lot of time in this document: we want, from an initial population, to differentiate an agent from the others. More concretely, we initially have all agents in state L, and we want to reach a point where all agents but one are in state N, the last one staying in state L. This agent will be then called the* **Leader***.*

The idea of the protocol is simple: when two agents in state L meet, exactly one changes its state into N.

More formally, the Leader Election Protocol is the following:

- *$Q = \{L, N\}$.*
- *$\Sigma = \{L\}$.*
- *$\iota(L) = L$.*
- *$Y = \{True\}$.*
- *$\omega(L) = \omega(N) = True$.*
- *δ is such that the rules are:*

$$\begin{array}{rl} L\ L \rightarrow & L\ N \\ L\ N \rightarrow & L\ N \\ N\ L \rightarrow & N\ L \\ N\ N \rightarrow & N\ N \end{array}$$

With these rules, the number of L can only decrease, and this number cannot be 0. We need to introduce an hypothesis that permits to ensure, even if an adversary choses the interacting agents, that we can reach an expected configuration at some point.

The fairness has been introduced in order to deal with this issue.

Notice that, using the computation definitions that will be provided in Definition 1.6, this protocol does not compute any non trivial predicate: It accepts every input.

Remark 1.4

- *A configuration will also be sometimes seen as a word of Σ^*. For example, in the leader election, from the initial configuration L^n (where n is the number of agents in the population), we can only reach configurations of the form $L^k N^{n-k}$ with $k > 0$.*

  The notion of fairness will then be defined to ensure that from the configuration $L^k N^{n-k}$, as we can reach the configuration LN^{n-1}, we will reach it at some point.

- *The rules $L\ N \rightarrow L\ N$, $N\ L \rightarrow N\ L$ and $N\ N \rightarrow N\ N$ do not change the configuration. Rules such as $\delta(q_1, q_2) = (q_1, q_2)$ will be called* **Trivial**, *and will often not be provided in protocol's description.*

- *A configuration will be said* **Stable** *if, for any pair of agents in states q_1 and q_2, the corresponding rule is trivial. A stable configuration can never change.*

1.2.2 Computation

Definition 1.5 *A* **Sequence of Configurations** *is a sequence $(C_i)_{i \in \mathbb{N}}$ such that for all i, $C_i \rightarrow C_{i+1}$, C_0 being an initial configuration. We say that a sequence is* **Fair** *if, for each configuration C appearing infinitely often in the sequence and for each configuration C' such as $C \rightarrow C'$, C' also appears infinitely often in the sequence.*

The notion of fairness is here to avoid some "undesired" adversaries: for example, an adversary that would choose the same pair at each time to interact. It is also motivated by giving a very weak constraint that covers the case of (non-degenerated) Markovian random interactions: if pairs of agents are chosen randomly and if C appears infinitely often and $C \to C'$, as the number of agents is fixed (and hence bounded), the probability for C' to appears infinitely often is 1, and hence sequence of configurations are fair almost surely.

We need to say explicitly how configurations are interpreted, in particular in our settings. This is rather natural:

Definition 1.6 *A configuration has an* **Interpretation** $y \in Y$ *if for every state q present in the configuration, $\omega(q) = y$. If there is no $y \in Y$ such that the configuration has interpretation y (i.e. there are q and q' in C such that $\omega(q) \neq \omega(q')$), the configuration is considered to have no interpretation.*

A population protocol **Computes** y *on some input S, if for every fair sequence $(C_i)_{i\in\mathbb{N}}$ starting from the initial configuration corresponding to S, there exists some integer J such that, for all $j > J$, the configuration C_j has the interpretation y.*

A Protocol **Computes a Function** $f : \mathbb{N}^{|\Sigma|} \to Y$ *if for every input $S \in \mathbb{N}^{|\Sigma|}$, the protocol computes $f(S)$ on S.*

A Protocol **Computes a Set** $A \subset \mathbb{N}^{|\Sigma|}$ *if $Y = \{True, False\}$ and $f^{-1}(True) = A$.*

Example 1.7 (Majority Protocol) *We want to compute an election: At the beginning, each agent votes for A or B. We create a protocol such that, we can be sure that at some point, each agent knows if there was strictly more A than B or not in the input. We want to compute the sets $\{x \in \Sigma^* | x_A > x_B\}$ and $\{x \in \Sigma^* | x_A \leq x_B\}$, where x_A is the number of A in x and x_B the number of B.*

The idea of the protocol is simple: when two agents in state A and B interact, they "cancel" each other. We add two states a and b corresponding on the belief of the winner.

Here is the formal Majority Protocol:

- $Q = \{A, B, a, b\}$.
- $\Sigma = \{A, B\}$.
- $\forall x \in \Sigma, \iota(x) = x$.
- $Y = \{A, B\}$.
- $\omega(A) = \omega(a) = A$, $\omega(B) = \omega(b) = B$.
- *δ is such that the non trivial rules are:*

$$\begin{array}{l} A\ B \to b\ b \\ A\ b \to A\ a \\ B\ a \to B\ b \\ a\ b \to b\ b \end{array}$$

The stable configurations (i.e. configurations that will not change, whatever pairs you chose interact) are exactly of the form A^*a^* and B^*b^*.

Let $(C_i)_{i\in\mathbb{N}}$ be a fair sequence of configurations. We will prove that it reaches the good output.

First, we can notice that there are as many A as B that disappear. If at some point there are no more A (resp B), there were less A than B (resp less B than A) in the input.

By fairness, we can be sure that as long as an A and a B are in the population, the first rule will eventually occur. Hence, there exists some $j \in \mathbb{N}$ such that configuration C_j no longer contains A or B. We work now with the sequence of configurations $(C_i)_{i\geq j}$.

If there is at least one A, by repeating the second rule, we can reach a configuration with only As and as, which has the good interpretation, as we had strictly more A than B.

If there is no A, there is at least one B or one b (as the only rule that can make disappear the last b implies the presence of an A, and the one that can make disappear the last B gives to the agent the state b). By repeating the third or fourth rule, we can delete all states a and reach a configuration with only Bs and bs, which has the good interpretation, as we did not have strictly more A than B.

As A^*a^* and B^*b^* are stable, once they are reached, the population will not evolve. Hence, as these configurations have an interpretation (and the desired one), from each input, the protocol provides the right output for each configuration at some point.

This protocol computes the Majority Protocol.

Definition 1.8 *A* **Predicate** *over a multiset $\mathbb{N}^\Sigma$ is a subdivision of this multiset into two parts: that is to say a partition in two classes; one class corresponding to multisets where the predicate is $True$ and the other where it is $False$.*

When we have an explicit definition of the predicate, we will note $\mathcal{P} = [\mathcal{F}(x_1, \dots, x_n)]$ (or $\mathcal{P} = [\mathcal{F}(x_{s_1}, \dots, x_{s_n})]$), where $\Sigma = \{s_1, \dots, s_n\}$ and $\mathcal{F} : \mathbb{N}^\Sigma \to \{True, False\}$. An element $X = (x_1, \dots, x_n) \in \Sigma^*$ verifies $\mathcal{P}$ if and only if $\mathcal{F}(X) = True$.

Remark 1.9 *The Majority Protocol corresponds to the predicate $[x_A > x_B]$.*

Example 1.10 (Computing 2^k) *Let k be a positive integer. We want to check, from a population of A and B if there is at least 2^k agents, i.e compute the predicate $[x_A \geq 2^k]$. We provide here an efficient protocol, as it uses only $k+1$ states.*

This protocol will help to understand a different version, more complicate as it will use a Game Theory constraint, see Section 2.3.4.

The idea of the protocol is the following: Each agent stores an integer. At the beginning, a starting A stores 1, a starting B stores 0. When two agent storing the same integer meet, one stores the sum, the other stores 0. The state 2^k is absorbing: It cannot disappear and each agent meeting the state 2^k receives the state 2^k.

Here is the formal Protocol:

- $Q = \{0\} \cup \{2^i | i \in [0, k]\}$.
- $\Sigma = \{A, B\}$.

- *$\iota(A) = 1$, $\iota(B) = 0$.*
- *$Y = \{True, False\}$.*
- *$\omega(x) = True \Leftrightarrow x = 2^k$.*
- *δ is such that the non trivial rules are:*

$$\begin{array}{llll} 1 & 1 \rightarrow 0 & 2 & \\ 2^i & 2^i \rightarrow 0 & 2^{i+1} & \textit{with } i < k \\ 2^k & X \rightarrow 2^k & 2^k & \textit{with } X \in Q \end{array}$$

The correctness of this protocol follows by noticing these following claims:

- *As long as no state 2^k appears, the sum of the integers stored in the agents does not change.*
- *Hence, if there were less than 2^k agents with input A at the beginning, the state 2^k will never appear.*
- *If there were at least 2^k agents A at the beginning, from any configuration, the state 2^k can be obtained. Hence, by fairness, a state 2^k will appear, and the third rule will ensure that all agents get the correct output.*

Remark 1.11 *In some protocols, agents will store integers in their state. It will often happen that we will talk about a q to say an agent in state q.*

1.2.3 Boolean Combination

We will now prove the main result about population protocols: what they can compute. As it corresponds to the boolean combination of two kinds of predicates, we first prove that boolean combinations of computable predicates are computable. We then introduce the two kinds of computable predicates (Threshold and Modulo) and provide corresponding protocols.

Theorem 1.12 ([AAD+04]) *The set of computable predicates by Population Protocols is stable under Boolean Combination.*

Proof: To show this result, it suffices to prove the stability under complementary and union.

Negation: Let $\mathcal{P}$ a predicate over Σ and $(Q, \Sigma, \iota, \{True, False\}, \omega, \delta)$ a protocol computing it.

We define the protocol $(Q, \Sigma, \iota, \{True, False\}, \omega', \delta)$, where $\omega'(x) = True$ if and only if $\omega(x) = False$. Here follows a proof that it computes the complementary of $\mathcal{P}$:

Let $X \in \mathcal{P}$ and $(C_i)_{i\in\mathbb{N}}$ be a fair sequence with input X in the second protocol. The same sequence exists and is fair in the first protocol (as only ω differs). By fairness, there exists some j such that, for any $i \geq j$, C_i has the interpretation $True$. Each agent being in a state such that ω gives $True$ is in a state such that ω' gives $False$. From this, each C_i has the interpretation $False$ with the second protocol. From this, X is refused by the second protocol.

By symmetry, when $X \notin \mathcal{P}$, we can prove that X is accepted by the second protocol.

Hence, this protocol computes the complementary of $\mathcal{P}$.

Union: Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two predicates over Σ and let $(Q_1, \Sigma, \iota_1, \{True, False\}, \omega_1, \delta_1)$ and $(Q_2, \Sigma, \iota_2, \{True, False\}, \omega_2, \delta_2)$ be protocols computing respectively $\mathcal{P}_1$ and $\mathcal{P}_2$.

Let $(Q', \Sigma, \iota', \{True, False\}, \omega', \delta')$ be a protocol such that:

- $Q' = Q_1 \times Q_2$.
- $\iota'(s) = (\iota_1(s), \iota_2(s))$.
- $\omega'(q) = \omega_1(q) \vee \omega_2(q)$.
- $\delta'(q_1, q_2) = ((q_a, q_b), (q_c, q_d))$ with $\delta_1(q_1, q_2) = (q_a, q_c)$ and $\delta_2(q_1, q_2) = (q_b, q_d)$.

Let $X \in \mathcal{P}_1 \cup \mathcal{P}_2$ and $(C_i)_{i \in \mathbb{N}}$ be a fair sequence with input X in the new protocol. We suppose $X \in \mathcal{P}_1$ (the case $X \in \mathcal{P}_2$ is symmetric).

For each configuration C_i, let C_i^1 be the configuration where each agent only has its first state in Q_1. By fairness, as $X \in \mathcal{P}_1$, there exists some j such that, for any $i \geq j$, C_i^1 has the interpretation $True$. By definition of ω', for any $i \geq j$, each C_i has interpretation $True$. X is accepted by the new protocol.

Let $X \notin \mathcal{P}_1 \cup \mathcal{P}_2$ and $(C_i)_{i \in \mathbb{N}}$ be a fair sequence with input X in the new protocol.

Given a configuration C_i, let C_i^1 (respectively C_i^2) be the configuration where each agent only has its first state in Q_1 (respectively second state in Q_2). By fairness, as $X \notin \mathcal{P}_1$ (respectively $X \notin \mathcal{P}_2$), there exists some j_1 (resp. j_2) such that, for any $i \geq j_1$ (resp. j_2), C_i^1 (resp. C_i^2) has the interpretation $False$. By definition of ω', for any $i \geq \max(j_1, j_2)$, each C_i has interpretation $False$. X is refused by the new protocol.

From this, our protocol computes the union of $\mathcal{P}_1$ and $\mathcal{P}_2$. $\square$

1.2.4 Presburger's Predicates

Population protocols compute exactly semilinear sets, which are sets that can be described with Presburger's arithmetic. We first define this logic and will prove that it can be computed by population protocols.

Definition 1.13 ([Pre29]) Presburger's Arithmetic *is the first order theory, that corresponds to Peano's arithmetic without multiplication:*

It corresponds in other words to all first order formulas that can be written with the symbols 0, 1, +, =, $\forall$, $\exists$, $\vee$, $\wedge$ *and* $\neg$ *and variables.*

A set $X \in \mathbb{N}^d$ *is* **Definable** *in Presburger's arithmetic if there exists some formula* $\mathcal{F}$ *such that* $X = \{(x_1, \ldots, x_d) \in \mathbb{N}^d | \mathcal{F}(x_1, \ldots, x_d) \text{ is satisfied}\}$*, where each* x_i *is a free variable of* $\mathcal{F}$*.*

Example 1.14 *$\mathcal{F}(x,y) =$ "$\exists a : x+x+x = y+y+2+a$" is a Presburger's arithmetic formula that corresponds to $[3x - 2y \geq 2]$.*

$\mathcal{F}(x,y) =$ "$\exists a : (x = y+y+1+a+a+a) \vee (x+a+a+a = y+y+1)$" is a Presburger's arithmetic formula that corresponds to $[x - 2y \equiv 1[3]]$.

As a reminder, $a \equiv b[c]$ means there exists some integer $k \in \mathbb{Z}$ such that $a = b + kc$.

It can be proved that the set $\{(x,y,z) \in \mathbb{N}^3 | xy = z\}$ cannot be expressed by any Presburger's arithmetic formula.

Presburger's arithmetic can be actually simplified:

Theorem 1.15 ([Pre29]) *Any predicate definable in Presburger's arithmetic corresponds to a boolean combination of:*

- **Threshold Predicate***: $[\sum a_i x_i \geq b]$, with $a_1, \ldots, a_n, b \in \mathbb{Z}^{n+1}$.*
- **Modulo Predicate***: $[\sum a_i x_i \equiv b[c]]$, with $a_1, \ldots, a_n, b, c \in \mathbb{Z}^{n+2}$.*

Recall that predicates computable by population protocols are closed under boolean combination (Theorem 1.12). To prove that predicates definable in Presburger's arithmetic are computable by population protocols, we only need to prove that:

- Any Threshold predicate can be computed by a Population Protocol.
- Any Modulo predicate can be computed by a Population Protocol.

Theorem 1.16 ([AAD$^+$04]) *Any Threshold predicate can be computed by a Population Protocol.*

Proof: We want to compute the predicate $[\sum a_i x_i \geq b]$. We suppose $b \geq 0$. We will explain how to adapt the protocol otherwise.

The idea of the protocol is:

- We define m as the minimal element of $\{0, a_i\}$ and M as the maximal element of $\{b, a_i\}$.
- Each agent uses a triplet for its state:
  1. The first element, in $\{L, N\}$, is here to perform a Leader Election.
  2. The second one is an integer in $[m, M]$.
  3. The third one, in $\{True, False\}$, is the agent's output.
- At the beginning, the stored integer of each agent is its input.
- Agents perform a leader election in parallel to the main protocol.

- When a leader agent meets an agent, the leader stores the sum of the integers, the other one stores 0. If the sum is smaller than the minimal possible integer/greater than the maximal possible integer, the leader stores the minimal/maximal integer, the other stores the rest. If the leader's integer is greater or equal to b, it gives the output $True$ to the second agent, otherwise it gives the output $False$.

Here is the formal Protocol:

- $Q = \{L, N\} \times [m, M] \times \{True, False\}$.
- $\Sigma = \{s_1, \dots, s_n\}$.
- $\forall i, \iota(s_i) = (L, a_i, True)$.
- $Y = \{True, False\}$.
- $\forall x \in \{L, N\}$, $\forall y \in [m, M]$, $\omega(x, y, True) = True$, $\omega(x, y, False) = False$.
- δ is such that the non trivial rules are:

$$\begin{array}{llrll}
(L, i, \_) & (\_, j, \_) \rightarrow & (L, i+j, True) & (N, 0, True) & b \leq i+j \leq M \\
(L, i, \_) & (\_, j, \_) \rightarrow & (L, b, True) & (N, i+j-b, True) & M < i+j \\
(L, i, \_) & (\_, j, \_) \rightarrow & (L, i+j, False) & (N, 0, False) & m \leq i+j < b \\
(L, i, \_) & (\_, j, \_) \rightarrow & (L, m, False) & (N, i+j-m, False) & i+j < m
\end{array}$$

We give the symbol _ if any element works in the slot.

To prove this protocol, we first notice a few things:

1. The sum of the integers remains stable under the rules.
2. At some point, there is a unique leader.

From this, we can consider a time j from which there is a unique leader. Let consider three cases:

1. $\sum a_i x_i \geq b$**.** By fairness, it is possible to reach a configuration where each agent contains a non-negative integer: as the sum is positive, the leader can always alternate "positive" agents with "negative" ones until no negative integers remain in the population.

   After that, the leader can reach an integer greater or equal to b. By repeating the 2 first rules, the population will reach a configuration with interpretation $True$ that will never change.

2. $0 \leq \sum a_i x_i < b$**.** We can reach a point where each non-leader agent contains 0: The leader can contain the sum.

   Then, by repeating the third rule, each agent will have the output $False$, and the configuration will never change its interpretation.

3. $\sum a_i x_i < 0$. A configuration where each agent contains a non-positive integer can be reached: by symmetry to the previous case, each agent will finish with an integer null or negative.

   From this configuration, by repeating the 2 last rules described in δ, the population reaches a configuration with interpretation $False$. This interpretation will never change from this point.

If we have $b < 0$, then we can switch each a_i and b by $-a_i$ and $-b$. We need to compute the predicate $[\sum(-a_i)x_i \leq -b]$, which is equivalent to $[\neg(\sum(-a_i)x_i \geq (-b+1))]$. By stability under negation, this can be computed. □

Theorem 1.17 ([AAD$^+$04]) *Any Modulo predicate can be computed by a Population Protocol.*

Proof: We want to compute the predicate $[\sum a_i x_i \equiv b[c]]$. We assume that $0 \leq a_i, b < c$.

The idea of the protocol is:

- Each agent uses a triplet for its state:
  1. The first element, in $\{L, N\}$, is here to perform a Leader Election.
  2. The second one is an integer in $[0, c-1]$.
  3. The third one, in $\{True, False\}$, is the agent's output.
- At the beginning, the stored integer of each agent is its input.
- Agents perform a leader election in parallel to the main protocol.
- When a leader agent meets an agent, the leader stores the sum of the integers modulo c, the other one stores 0. If the leader stores b, it gives the output $True$ to the second agent, otherwise it gives the output $False$.

Here is the formal Protocol:

- $Q = \{L, N\} \times [0, c-1] \times \{True, False\}$.
- $\Sigma = \{s_1, \ldots, s_n\}$.
- $\forall i, \iota(s_i) = (L, a_i, True)$.
- $Y = \{True, False\}$.
- $\forall x \in \{L, N\}$, $\forall y \in [m, M]$, $\omega(x, y, True) = True$, $\omega(x, y, False) = False$.
- δ is such that the non trivial rules are:

$$\begin{array}{llrll}
(L,i,\_) & (\_,j,\_)\to & (L,b,True) & (N,0,True) & i+j=b \text{ or } i+j=b+c \\
(L,i,\_) & (\_,j,\_)\to & (L,i+j,False) & (N,0,False) & i+j\neq b \text{ and } i+j<c \\
(L,i,\_) & (\_,j,\_)\to & (L,i+j-c,False) & (N,0,False) & i+j\neq b \text{ and } i+j\geq c
\end{array}$$

We give the symbol _ if any element works in the slot.

To prove this protocol, we first notice a few things:

1. The sum of the integers modulo c remains stable under the rules.
2. At some point, there is a unique leader.
3. At some point, only the leader may contain a non 0 integer (it will happen when the leader will have met each agent).

If the sum is equal to b modulo c, the first rule will provide a stable configuration with the interpretation $True$. Otherwise, the second rule will provide a stable configuration with the interpretation $False$. □

The following theorem is the main result about population protocols: Only predicates definable in Presburger's arithmetic can be computed by a population protocol.

Theorem 1.18 ([AAER07, AAD$^+$04]) *The predicates computable by Population Protocols correspond exactly to the predicates definable in Presburger's Arithmetic.*

Actually, the sets definable in Presburger's arithmetic correspond exactly to the Semilinear sets. Here is the definition of such sets:

Definition 1.19 *A* **Semilinear Set** *over $\mathbb{N}^k$ for some $k > 0$ is a finite union of sets of the form*

$$\left\{ u + \sum_{i\leq I} a_i v_i | \textit{with } u, v_1, \ldots, v_I \in \mathbb{N}^k \textit{ and } a_1, \ldots, a_I \in \mathbb{N} \right\}$$

If we perform an analogy between $\mathbb{N}^k$ and $\mathbb{N}^\Sigma$, with $k = \Sigma$, Theorem 1.18 is equivalent to this version:

Theorem 1.20 ([AAER07, AAD$^+$04]) *The sets computable by Population Protocols correspond exactly to the Semilinear Sets.*

Remark 1.21 *Some of the complexity results of this thesis are about Semilinear sets. To simplify some of the proofs, we will often prove that boolean combinations, threshold predicates and modulo predicates can be computed.*

1.2.5 Non Deterministic Rules

In the seminal (and some others) version of population protocols, δ is not deterministic. In those versions, δ is a subset of Q^4 or a function $Q^2 \to \mathscr{P}(Q^2)$. In this case, for each (q_1, q_2, q_1', q_2') in δ (or (q_1', q_2') in $\delta(q_1, q_2)$), we have the rule $q_1 q_2 \to q_1' q_2'$. We add the rule $q_1 q_2 \to q_1 q_2$. When two agents meet, we apply one of the possible rules.

With transition function δ being non deterministic, there are two possibilities for acceptance of the input, leading to two different results:

1. One keeps the notion of fair sequences, and one just needs to find from any configuration a path to a configuration where an interpretation exists and cannot be changed.

   This version is not more powerful than the population protocols, as proven in the following Theorem. Hence, the choice of having δ deterministic does not induce a loss of power.

Theorem 1.22 ([AAC⁺05]) *For every protocol with non deterministic rules, there exists a set of deterministic rules such that if the first stably computes a predicate P, then the second also stably computes P.*

2. The second accepts an input if there exists a sequence of configurations leading to a configuration where an interpretation exists and cannot be changed. The model becomes more powerful than the population protocols. A better view of what can be computed is determined in [BBRR12].

1.3 Fast Computing

We want to consider now the speed of a Protocol. We were up to now speaking about fair computations. Here, we will follow the works in [AAE06]. We assume that interactions are randomly chosen, according to the most simple and intuitive law: Between two steps, each pair of agents has the same probability to meet. We will describe quickly the main tools used in the article to compute with speed $O(n \log^5 n)$ any predicate computed by any classic population protocol when the population starts with a unique Leader. These tools will be used later for our works on Fast Computing with Identifiers in Chapter 4.

1.3.1 Probabilistic Protocols

Definition 1.23 *Assume that the choice of the interacting pair is not chosen by an adversary, but is chosen according to the uniform probabilistic law: Each pair of agents has the same probability of being selected. We say* **Expected Number of Interactions** *for the expected number of interactions, with this law, needed to reach a stable configuration.*

Remark 1.24 *From any protocol computing a predicate, with the fairness of Definition 1.5, the probability that a probabilistic sequence reaches a stable configuration is 1.*

Proposition 1.25 *The expected number of interactions needed to reach a configuration with a unique Leader L with the Leader Election protocol of Example 1.3 is $\Theta(n^2)$.*

Proof: Let p_k be the probability to reach, from a configuration with $k+1$ agents in state L, a configuration with k agents in state L. This probability corresponds to the probability of selecting two of the agents in state L. From this, we have:
$p_k = \frac{\text{Number of possible pairs of } L}{\text{Number of possible pairs}} = \frac{k(k+1)/2}{n(n-1)/2} = \frac{k(k+1)}{n(n-1)}$.

Let $\mathbb{E}_k$ be the expected number of steps to reach the configuration $L^k N^{n-k}$ from a configuration $L^{k+1}N^{n-k-1}$. We have, as the success of this event does not depend of the number of previous failures, the following:
$\mathbb{E}_k = \frac{1}{p_k} = \frac{n(n-1)}{k(k+1)}$.

As the first configuration is composed of n leaders and we need to reach a configuration with a unique one, we get a bound for the expectation $\mathbb{E}$:
$\mathbb{E} = \sum_{k=n-1}^{1} \mathbb{E}_k = \sum_{k=n-1}^{1} \frac{n(n-1)}{k(k+1)} = n(n-1) \sum_{k=n-1}^{1} \frac{1}{k(k+1)}$

We have $\frac{1}{2} \leq \sum_{k=n-1}^{1} \frac{1}{k(k+1)} \leq \sum_{k=1}^{n-1} \frac{1}{k^2} \leq \sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}$.

From this, we get:

$$\frac{n(n-1)}{2} \leq \mathbb{E} \leq \frac{\pi^2}{6} n(n-1).$$

Hence, the expected number of interactions to reach a unique Leader is $\Theta(n^2)$. □

Proposition 1.26 *The expected number of interactions in Theorem 1.17 is $O(n^2 \log n)$.*

Proof: The proof is quite similar to the previous one:

- The protocol first work as a Leader Election. Hence, the uniqueness of a Leader will be reached in $O(n^2)$.
- Then a stable configuration will be reached when the leader will have met all the agents that do not have yet the right output. This corresponds to the coupon collector problem (see for example [ER61]), which has an expectation in $O(n^2 \log n)$.

□

Definition 1.27 ([AAE06]) *The* **Random Walk Broadcast** *protocol is a protocol working for a population with a single leader. It has an input, and tries to broadcast it as quick as possible:*

With $Q = \{L, N\} \times Y$, where Y is the set of outputs, the rules are:

$$\begin{array}{ll} (L,y)\ (N,y') \rightarrow (L,y)\ (N,y) & \forall y \in Y \\ (N,y)\ (N,y') \rightarrow (N,y)\ (N,y) & \forall y \in Y \end{array}$$

Theorem 1.28 ([AAE06]) *Starting from any initial configuration with a single leader, the random walk broadcast protocol converges to a configuration in which all agents have the same output value in $O(n^2)$ interactions in expectation.*

Proof: The proof can be found in [AAE06].

The improvement of the complexity here comes from the fact that not only the leader can switch agent's opinion. Angluin *et al.* prove that the combination of a broadcast from the leader and a random spread from the other agents (as agents gives their opinion, not knowing if they are right) is more effective. □

1.3.2 Epidemics

We will now introduce the important tools to compute quickly a semilinear set when the population starts with a unique leader. These tools are introduced and well described in [AAE06].

- Each agent will store the same number d of bits, for some $d \in \mathbb{N}$, called $R_1, \ldots, R_d$. With this, the whole population will encode d registers, each one containing a number between 0 and n. Register i corresponds to the sum of the digits R_i over all the agents.
- The leader will perform an algorithm consisting of operations (addition, subtraction, division, multiplication...). To do it, it will cut the operations into basic operations over the registers.
- To orchestrate the operations, the leader will spread (or broadcast) through epidemic protocols the operations to perform. To be almost sure that each step is over, it will use a clock system.

In Chapter 4, most of our protocols will work with epidemics. This section is here to prove the announced number of expected interactions. The next section will provide a tool that permits the population to know when an epidemic has happened with high probability.

Definition 1.29 ([AAE06]) *We call the* **Epidemic Protocol** *a protocol who spreads an epidemic. More formally, if 0 represents the not infected state and 1 the infected one, their is just a non trivial rule:*

$$1\ 0 \to 1\ 1$$

Most of the time, it will be a leader who will start a spreading of some information. The computation will start in the configuration 10^{n-1}, where 1 represents the leader.

Proposition 1.30 ([AAE06]) *Let T be the expected number of interactions before an epidemic protocol starting with a single infected agent infects all the other ones. For any fixed $c > 0$, there exists positive constants c_1 and c_2 such that, for sufficiently large n, with probability at most $1 - n^{-c}$:*

$$c_1 n \log n \leq T \leq c_2 n \log n$$

From this theorem, we know that any epidemic protocol will take $O(n \log n)$ expected interactions. It will even take $\Omega(n \log n)$ interactions. If we are (almost) sure that more than $c_2 n \log n$ interactions occurred, we will be (almost) sure that an epidemic have occurred.

1.3.3 Phase Clock Protocol

We introduce the phase clock protocol. It is a protocol that permits to be almost sure that more than $c_2 n \log n$ interactions occurred. With this protocol, we can be sure that an epidemic occurred with high probability (as it needs at most $c_2 n \log n$ to occur with high probability).

In Chapter 4, each time we run an epidemic, we run in parallel a round of the phase clock in parallel to ensure with high probability that the epidemic occurred.

Definition 1.31 ([AAE06]) *The* **Phase Clock Protocol** *is, for a fixed m, a protocol where each agent has an integer between 0 and $m-1$, corresponding to its "timer".*

At the beginning, everyone has the integer 0 and there is a unique Leader. The Leader always contains the current hour. When the leader meets an agent with the same hour, the leader increments its hour (modulo m). When two agents with respective hours a and b meet, they keep the most likely right hour (see Rule 3 of δ).

The non trivial rules are as follows:

$$\begin{array}{rlrll}
(L,a) & (N,a) & \rightarrow (L,a+1) & (N,a) & \text{with } a < m-1 \\
(L,m-1) & (N,m-1) \rightarrow & (L,0) & (N,m-1) & \\
(N,a) & (N,b) \quad \rightarrow & (N,a) & (N,a) & \text{with } a-b \leq m/2 \text{ or } a+m-b \leq m/2
\end{array}$$

A **Phase** *corresponds to an interaction where the leader increments its clock. A* **Round** *is when the leader's clock becomes* 0.

Proposition 1.32 ([AAE06]) *For any fixed $c, d_1 > 0$, there exist two constants m and d_2 such that, for all sufficiently large n, with probability at least $1 - n^{-c}$ the phase clock protocol with parameter m, starting from an initial configuration consisting of one leader L with timer 0 and $n-1$ agents N with timer $m-1$, completes n^c rounds of m phases each, where the minimum number of interactions in any of the n^c rounds is at least $d_1 n \log n$ and the maximum is at most $d_2 n \log n$.*

From this result, as long as our algorithm use, for some $c > 0$, at most n^c epidemics, we can use a clock to operate the computation with a success with a probability greater than $1 - n^{-c}$.

1.3.4 Basic Arithmetic Operations

We will not formally describe the algorithms developed in [AAE06]. We will just give the speed of basic operations over registers A, B and C (the speed corresponds to the number of epidemics needed):

Operation	Interpretation	Speed
Addition	$A \leftarrow A + B$	$2 \log n$
Subtraction	$C \leftarrow A - B$	$O(\log^3 n)$
Division	$C \leftarrow A/B$	$O(\log^4 n)$

The multiplication by constants can be performed by repeating additions and subtractions.

1.3.5 Semilinear Sets

Theorem 1.33 ([AAE06]) *For any semilinear predicate P, and for any $c > 0$, there is a probabilistic population protocol with a leader to compute P without error that converges in $O(n \log^5 n)$ interactions with probability at least $1 - n^{-c}$.*

Proof: The idea of the proof is that any semilinear predicate, using the previous arithmetics tools, can be computed with high probability with a speed at most $n \log^5 n$. We will combine this protocol P_1 with P_2, the one of Theorem 1.28.

The population computes the two protocols in parallel. It will also run a protocol P_3 which has high probability to take $\Omega(n^3)$ interactions. This protocol is constructed such that it finishes at some point by fairness.

As long as P_3 is not over, the output of each agent will correspond to P_1's. As soon as P_3 is over (or looks to be over for an agent), each agent switch its output to P_2's.

Most of the time, P_1 will finish in $O(n \log^5 n)$ interactions, then P_2 will finish in $O(n^2)$ interactions. Then, when P_3 will be over, each agent will already have the right output and it will not change. Hence, in this case, the output will be reached when P_1 has reached it.

In the corner cases, when P_2 and P_3 will finish, everyone will have the right output. This will take an expected number of $O(n^3)$ interactions. As this will happen with probability at most $1 - n^{-c}$ for any c, we can choose c large enough to get a global expectation of $O(n \log^5 n)$. □

1.4 Community Protocols

We present now an extension of Population Protocols introduced by Guerraoui and Ruppert in [GR09]: Agents have now unique identifiers, and can store a fixed number of them. Agents can compare 2 identifiers.

This model has been proved to correspond to $NSPACE(n \log n)$, even when we add a fixed number of malicious agents. We will not consider malicious agents in this thesis.

1.4.1 Definition

Definition 1.34 *A* **Community Protocol** *is given by seven elements $(B, d, \Sigma, Y, \iota, \omega, \delta)$ where:*

- *B is a finite set of basic states.*
- *$d \in \mathbb{N}$ is a positive integer corresponding to the number of identifiers that can be remembered by an agent.*
- *Σ, Y, ι and ω are the same as in the Population Protocols model.*
- *δ is a function $Q^2 \to Q^2$, with $Q = B \times U \times (U \cup \{\_\})^d$.*

Remark 1.35 *As with our definition of population protocols, we choose here that δ is deterministic. In the seminal model in [GR09], it is not. The power of the model remains unchanged with the determinism of δ, for the same reasons as the ones provided in the proof of Theorem 1.22 in [AAC+05].*

The set $Q = B \times U \times (U \cup \{\_\})^d$ of possible states for each agent is composed of three elements: the identifier, the state, and d slots for identifiers. The initial state of an agent assigned with the identifier id and the input s is $(\iota(s), id, \_^d)$, where $\_^d$ states for d repetitions of the empty slot $\_$.

An input will often be seen as a word of Σ^*, as it is possible to sort the input elements according to the identifiers. An input $u = u_1 \dots u_n$ is such that the agent with the smallest identifier has input u_1, the second has input $u_2 \dots$

The transition function δ has two restrictions: Agents cannot store identifiers that they never heard about, and the transitions must only depend on relative position of the identifiers in the slots and on the state in B. More formally, we have:

1. *if $\delta(q_1, q_2) = (q'_1, q'_2)$, and id appears in q'_1 or q'_2 then id must appear in q_1 or in q_2.*
2. *whenever $\delta(q_1, q_2) = (q'_1, q'_2)$, let $u_1 < u_2 < \cdots < u_k$ be the distinct identifiers that appear in any of the four states q_1, q_2, q'_1, q'_2. Let $v_1 < v_2 < \cdots < v_k$ be distinct identifiers. If $\rho(q)$ is the state obtained from q by replacing all occurrences of each identifier u_i by v_i, then we require that $\delta(\rho(q_1), \rho(q_2)) = (\rho(q'_1), \rho(q'_2))$.*

Remark 1.36 [Convention] *We will often call an agent of initial identifier $id \in U$ the agent id.*

We will sometimes write Id_k for the kth identifier present in the population.

An agent with identifier id, in state q and with a list of d identifiers $L = id_1, \dots, id_d$ will be written in what follows q if the identifiers do not matter, q_{id} or $q_{id,L}$ if the list does not matter, and $q_{id,id_1,\dots,id_d}$ in the other cases.

Example 1.37 (Leader Election) *It is possible to compute a Leader Election, where the leader will be the agent with the smallest identifier, with $O(n \log n)$ expected interactions. Agents will store the identifier of their leader. Here is the protocol, using above notations for rules:*

- $B = \{L, N\}$.
- $d = 1$.
- $\Sigma = L$, $Y = \{True\}$, $\iota(L) = L$ *and* $\omega(L) = \omega(N) = True$.
- δ *is such that the non-trivial rules are (using above conventions):*

$$\begin{array}{rcll} L_{id_a,\_}\ L_{id_b,\_} & \rightarrow & L_{id_a,\_}\ N_{id_b,id_a} & \text{with } id_a < id_b \\ L_{id_a,\_}\ N_{id_b,id_c} & \rightarrow & L_{id_a,\_}\ N_{id_b,id_a} & \text{with } id_a < id_c \\ L_{id_a,\_}\ N_{id_b,id_c} & \rightarrow & N_{id_a,id_c}\ N_{id_b,id_c} & \text{with } id_c < id_a \\ N_{id_a,id_b}\ N_{id_c,id_d} & \rightarrow & N_{id_a,id_b}\ N_{id_c,id_b} & \text{with } id_b < id_d \end{array}$$

We can see that by fairness, the population will reach a configuration with a unique leader, being the agent with the smallest identifier. All agents will have in their slot this identifier.

To determine the speed of this protocol, it suffices to realize that the final leader actually does an epidemic to spread its identifier. An epidemic takes $O(n \log n)$ expected interactions.

Definition 1.38 *The* **Sorted Language** *on a sorted alphabet $\Sigma = \{s_1, s_2, \ldots, s_k\}$ is the language $s_1^* s_2^* \ldots s_k^*$. It corresponds to the language where a s_i cannot appear after a s_j in a word if $i < j$.*

Example 1.39 *The following Community Protocol computes the Sorted Language:*

- $B = \Sigma \cup \{F\}$.
- $d = 0$.
- $Y = \{True, False\}$.
- $\forall i$, $\iota(s_i) = i$ *and*
- $\forall i$, $\omega(i) = True$, $\omega(F) = False$.
- δ *is such that the non-trivial rules are:*

$$\begin{array}{rcl} i_{id_a}\ j_{id_b} & \rightarrow & F_{id_a}\ \ F_{id_b} \quad \text{with } i < j \text{ and } id_a > id_b \\ i\ F & \rightarrow & F\ \ F \end{array}$$

This protocol uses the fact that an input is in the sorted language if and only if, for all $id_a < id_b$, the input of id_a is smaller than id_b's.

1.4.2 Characterisation

Theorem 1.40 ([GR09]) *Any decision problem in $NSPACE(n \log n)$ can be computed by a community protocol.*

Sketch of the proof : There are two important elements in the proof of [GR09] that permit to understand how this result is possible:

1. One can see the agents sorted by identifier as a tape. The first cell corresponds to a slot in the agent with the lowest identifier, the second cell to the same slot in the second lowest identifier...
2. The cell slots are not filled with bits, they are filled with identifier. To each identifier corresponds a sequence of $\log n$ bits. The lowest identifier corresponds to the chain $0^{\log n}$, the second to the chain $0^{\log n - 1}1$, the third to the chain $0^{\log n - 2}10$, ..., the ith to the number i written in binary on $\log n$ slots.

□

Remark 1.41

- *Notice that Guerraoui and Ruppert [GR09] established that this holds even with Byzantine agents, under some rather strong conditions.*
- *The converse of the theorem is also true. Here is an idea the proof that any community protocol can be simulated by a Machine M in $NSPACE(n \log n)$:*

  Deciding the output corresponds to solving two problems in the configuration graph:

  - *From initial configuration C_0, find a reachable configuration C_j that is output stable.*
  - *Checking that C_j is actually output stable. It means checking that there is no C' such that $C_j \to^* C'$ and C' has no output (or a different output that C_j's output).*

  If we give integer identifiers between 0 and $n-1$, an agent's state in Q can be written using $O(\log n)$ bits. Hence, we can write the whole configuration using a space of $O(n \log n)$.

  Deciding if C_j is output stable corresponds to check if, for all C' that can be reached, C' has the same interpretation. This can be done in $co-NSPACE(n \log n)$.

  As $n \log n \geq \log n$, we have, from Immerman-Szelepcsnyi's Theorem [Imm88, Sze88] $NSPACE(n \log n) = co-NSPACE(n \log n)$.

  Hence deciding if C_j is output stable is in $NSPACE(n \log n)$, and hence can be done by a non deterministic machine M using a space $n \log n$.

  Now, using this machine M as an oracle (sub-procedure) testing whether from initial configuration C_0, one can reach a configuration C_j that is output stable can be done easily in non deterministic space $n \log n$: guess non-deterministically one by one the sequence of configurations from C_0 to C_j, and use non-deterministic machine M as sub-procedure to test whether C_j is output stable.

1.5 Passively Mobile Protocols

We provide here formal definition of the model introduced by Chatzigiannakis *et al.* [CMN+11]: the Passively Mobile Protocols. Basically, the idea is to provide some Turing Machines to each agent. Each agent has the same space of computation. When the space is constant and does not depend of the size of the population, it corresponds to population protocols.

We will recall the results of [CMN+11]. They prove that with a space $o(\log \log n)$, the model is equivalently powerful to population protocols. With a space $f(n) = \Omega(\log n)$, the model is equivalent to a Turing Machine of space $n \times f(n)$, as agents can create unique identifiers.

Even if this model has been introduced before the Community Protocols model, we decided to introduce it after, has it is more powerful. With a space $\Omega(\log n)$, Passively Mobile Protocols can simulate Community Protocols.

1.5.1 Definitions

Definition 1.42 ([CMN$^+$11]) *A* **Passively Mobile Protocol (PM Protocol)** *is given by six elements* $(\Sigma, \Gamma, Q, \delta, \gamma, q_0)$ *where:*

- Σ *is the input alphabet, where* $\_ \notin \Sigma$*,*
- Γ *is the tape alphabet, where* $\_ \in \Gamma$ *and* $X \subset \Gamma$*,*
- Q *is the set of states,*
- δ*:* $Q^4 \to Q \times \Gamma^4 \times \{L, R\}^4 \times \{0, 1\}$ *is the internal transition function,*
- $\gamma : Q \times Q \to Q \times Q$ *is the external transition function (or interaction transition function), and*
- $q_0 \in Q$ *is the initial state.*

Each agent is equipped with the following:

- *A sensor in order to sense its environment and receive a piece of the input.*
- *Four read/write tapes: the working tape, the output tape, the incoming message tape and the outgoing message tape.*
- *A control unit that contains the state of the agent and applies the transition functions.*
- *Four heads (one for each tape) that read from and write to the cells of the corresponding tapes and can move one step at a time, either to the left or to the right.*
- *A binary working flag either set to 1 meaning that the agent is working internally or to 0 meaning that the agent is ready for interaction.*

The exact explanations of how the model work can be found in [CMN$^+$11]. It globally runs like population protocols, we will provide here a quick idea of how it works, giving a parallel with the population protocols' mechanisms.

To a given protocol is associated the function $f(n)$ that gives a bound of the size that the 4 tapes agents can have. All agents start with an input, in state $q_0 \in Q$, with flag 1, with its input symbol written on the first cell of the first tape and all the others cells full of blank symbol $\_$.

Agents are either working on their tape (flag equals to 1), either looking for an interaction (flag 0). As long as the flag is 1, they just keep applying the internal rules until δ's fifth element gets equal to 0 (it will mean that internal work is over and that the agent is now ready for an interaction).

When two agents interact (and they both have a flag 0), each writes in its incoming message tape what is written on the other's agent message tape. They also update their state in Q according to Γ. They switch their flag to compute taking into consideration their state update and what they wrote on their third tape.

A configuration has an output $\omega \in (\Gamma \setminus \_)^*$ if all agents have ω written on its output tape. If there are no a unique word on the output of each agents, the configuration has no output.

Like in population protocols, a configuration is output stable if it never evolves. The fairness definition is the same that in population protocols.

Definition 1.43 *Let* **PMSPACE(f(n))** *be the class of all predicates that are stably computable by some PM protocol that uses $O(f(n))$ space in every agent (and in all of its tapes).*

We also consider the particular case $f(n) = \log n$ by defining **PLM**= $PMSPACE(\log n)$.

To introduce the results from [CMN+11], we first need to introduce two computational classes:

Definition 1.44 *We define the two classes* **SSPACE(f(n))** *and* **SNSPACE(f(n))** *as the restrictions of $SPACE(f(n))$ and $NSPACE(f(n))$ to symmetric predicates (i.e. stable under input's permutation), respectively: that is to say, any element of these classes accepts a word $u = u_1 \dots u_n$ if and only if it accepts, for any permutation $\rho : \mathbb{N} \to \mathbb{N}$, $u' = u_{\rho(1)} \dots u_{\rho(n)}$.*

1.5.2 Computational Hierarchy

In [CMN+11], Chatzigiannakis *et al.* provide an exact characterization of the computational power of the model when we provide a space $o(\log \log n)$ to each agent and when we provide a space $\Omega(\log n)$. It induced two major results. We will give some intuition for both of them.

Theorem 1.45 ([CMN+11]) *We have $PLM = SNSPACE(n \log n)$.*

For any function $f(n) = \Omega(f(n))$, $PMSPACE(f(n)) = SNSPACE(nf(n))$.

Proof: The proof is performed in two inclusions. I will provide here some clues of how it is performed:

PLM⊂SNSPACE($n \log n$):

We need to simulate the protocol. To encode an agent, we just need to encode its state, that will use a space of $O(\log n)$ (the internal state and the 4 tapes). n agents can then be computed with space of $O(n \log n)$.

We just need to find a path of interactions from the input to a configuration with the right output and show that this output is stable. The proof is the same that in Remark 1.41: Finding the path is provided via the non-determinism. Proving the output stability is obtained via the fact that $SNSPACE(n \log n) = co-SNSPACE(n \log n)$ [Imm88, Sze88].

SNSPACE($n \log n$)⊂PLM:

The idea is that the population starts by creating identifiers with a simple algorithm. All agents start with identifier 1, and when two agents with the same identifier meet, one of them increments its identifier. By fairness, at some point, each agent will have a single identifier in $[1, n]$. This can be done as it requires $\log n$ bits. We can hence simulate any community protocol, as we have enough space to encode the d identifiers to store.

Then, by using Theorem 1.40, we get the inclusion.

The proof with $f(n) = \Omega(f(n))$ remains quite similar. We just need to notice that with the identifiers, we can see all the agents tapes as a single one (we put them in the order corresponding to the identifiers'). Then, the tape has a size $n \times O(f(n))$. $\square$

Theorem 1.46 ([CMN$^+$11]) *Any PM protocol that uses a space $f(n) = o(\log \log n)$ can only compute semilinear predicates.*

The authors in [CMN$^+$11] studied the case $f(n) = o(\log n)$. They did not find any characterization for this case, but proved that the second result of Theorem 1.45 does not work for $f(n) = o(\log n)$:

Corollary 1.47 ([CMN$^+$11]) *For any function $f(n) = o(\log n)$ it holds that:*

$$PMSPACE(f(n)) \subsetneq SNSPACE(nf(n)).$$

We will provide in Section 6.1 an exact characterization for the case $f(n) = O(\log \log n)$.

This concludes the presentation of the models used in this document. Population Protocols will be used in Part I. Community Protocols and Passively Mobile Protocols will be used in Part III.

Part II

Weakening over the Rules: Adding Rational Behavior to Population Protocols

Chapter 2

Pavlovian Population Protocols

This chapter focuses on ***Pavlovian Population Protocols***. Here, agents do not simply interact by applying rules. They are considered to play a symmetric game, using the pavlovian behavior from game theory to choose how to update internal states.

We first provide some results to motivate the chosen restrictions (by proving, in particular, that neither symmetric rules nor asymmetric games are restrictive enough). We prove that any symmetric 2 state protocol is pavlovian. We then give some interesting protocols (majority protocol, counting up to 3), and then a non trivial game computing the question "Are there at least 2^k agents in the population?" We finish with some conjectures about what we believe can and cannot be computed.

This chapter corresponds to the work published in [BCC+13] in the International Journal of Unconventional Computing. This also includes some results from [BCC+11] published in OPODIS 11. Those works was joint with Olivier Bournez, Jérémie Chalopin, Johanne Cohen and Xavier Koegler.

2.1 Model

2.1.1 Introduction

In this chapter, we turn two players games in the sense of game theory into dynamics over agents, by considering $PAVLOV$ behavior. The pairwise interactions between finite-state agents are sometimes motivated by the study of the dynamics of particular two-player games from game theory. For example, Dyer *et al* [DGG+02] considers the dynamics of the so-called $PAVLOV$ behavior in the iterated prisoner's dilemma. Several results about the time of convergence of this particular dynamics towards the stable state can be found in [DGG+02], and [FMP04], for rings, and complete graphs.

This is clearly not the only way to associate a dynamic to a game. They are several famous classical approaches: The first consists in repeating games: see for example [OR94, Bin91]. The second in using models from evolutionary game theory: refer to [HS03, Wei95] for a presentation of this latter approach. The approach considered here falls in method that consider dynamics obtained by selecting at each step some players and let them play a fixed game. Alternatives to $PAVLOV$ behavior could include $MYOPIC$ dynamics (at each step each player chooses the best response to previously played strategy by its adversary), or the well-known and studied $FICTIOUS-PLAYER$ dynamics (at each step each player chooses the best response to the statistics of the past history of strategies played by its adversary). We refer to [FL96, Bin91] for a presentation of results known about the properties of the obtained dynamics according to the properties of the underlying game. This is clearly non-exhaustive, and we refer to [Axe84] for an incredible zoology of possible behaviors for the particular iterated prisoner's dilemma game, with discussions of their comparative merits in experimental tournaments.

Jaggard *et al.* [JSW11] studied a distributed model similar to protocol populations where the interactions between pairs of agents correspond to a game. Unlike what happens in our model, there each agent has its own pay-off matrix and has some knowledge of the history. Jaggard *et al.*'s work gives several non-convergence results.

The organization of this chapter is the following:

1. A motivation about the analogy between repeated symmetric games and population protocols is presented.
2. A description of the model.
3. A motivation of our choices by 2 arguments:
    (a) Asymmetric Games is not restrictive enough to weaken the model.
    (b) Symmetric Rules do not change the power of classic Population Protocols. Hence the weakening will come from the game restrictions.
4. Some impossibility results.
5. Some complicate games computing predicates.
6. Some conjectures.

2.1.2 Game Theory on Symmetric Games

We now recall the simplest concepts from Game Theory. Here, we will restrict to symmetric games. Thus, we focus on symmetric non-cooperative games, with complete information, in normal form.

Definition 2.1 *A* **Symmetric Two-Player Game** *is a couple* (S, A) *where both players have the same finite set of actions* S*. For each player, the payoff function is denoted by* $A : S \times S \to \mathbb{R}$ *where* $A(i, j)$ *denotes the payoff to the player choosing the first argument when his opponent chooses the second argument.*

From now on, the payoff function is view as the payoff matrix. With some abuse of notation, $A_{i,j}$ denotes the payoff to the player choosing the strategy i when his opponent chooses strategy j.

Example 2.2 (Prisoner's dilemma) *The* **Prisoner's Dilemma** *game is a couple* (S, A) *where* $S = \{C, D\}$ *and the matrix* A *is as follow :*

$$\begin{array}{cc|cc|}
 & & \multicolumn{2}{c}{\textit{Opponent}} \\
 & & C & D \\
\cline{3-4}
\textit{Player} & C & 1 & -3 \\
 & D & 3 & -1 \\
\cline{3-4}
\end{array}$$

We will also introduce some game theory concept: the best response and the Nash equilibrium.

Definition 2.3 *Let* G *be a symmetric two-player game* (S, A)*.*

- *A strategy* $x \in S$ *is said to be a* **Best Response** *to strategy* $y \in S$*, denoted by* $x \in BR(y)$ *if for all strategies* $z \in S$*, we have* $A_{z,y} \leq A_{x,y}$
- *A strategy* $x \in S$ *is said to be a* **Best Response to Strategy** y **Among those Different from** x'*, denoted by* $x \in BR_{\neq x'}(y)$ *if for all strategies* $z \in S \backslash \{x'\}$*, we have* $A_{z,y} \leq A_{x,y}$

Remark 2.4 *In the Prisoner's dilemma, we have* $BR(C) = BR(D) = D$*,* $BR_{\neq D}(D) = C$*.*

Definition 2.5 *Let* G *be a symmetric two-player game* (S, A)*. A pair* (x, y) *is a* **(Pure) Nash Equilibrium** *if* $x \in BR(y)$ *and* $y \in BR(x)$*.*

In other words, two strategies (x, y) form a Nash equilibrium if in that state neither of the players has a unilateral interest to deviate from it. Note that a pure Nash equilibrium does not always exist.

Remark 2.6 *In the Prisoner's dilemma, since* $BR(C) = D$ *and* $BR(D) = D$*,* (D, D) *is the unique pure Nash equilibrium. In it, each player has score* -1*. The well-known paradox is that if they had played* (C, C) *(cooperation) they would have had score* 1*, that is more. The social optimum* (C, C)*, is different from the equilibrium that is reached by rational players* (D, D)*, since in any other state, each player fears that the adversary plays* D*.*

2.1.3 Repeated Games

Repeating k times the same game, is equivalent to extending the space of choices into S^k. The players chose their respective actions $x(t)$ and $y(t) \in S$ at time t for $t = 1, 2, \cdots, k$. Hence, this is equivalent to a two-player game with $|S|^k$ choices for players. To avoid confusion, we will call *actions* the choices $x(t)$, $y(t)$ of each player at time t, and *strategies* the sequences $X = x(1), \cdots, x(k)$ and $Y = y(1), \cdots, y(k)$, that is to say the strategies for the whole game.

Behaviors In practice, player I (respectively II) has to solve the following problem at each time t: given the history of the game up to now, that is to say $X_{t-1} = x(1), \cdots, x(t-1)$ and $Y_{t-1} = y(1), \cdots, y(t-1)$ what should player I (resp. II) play at time t? In other words, how to choose $x(t) \in S$? (resp. $y(t) \in S$?)

With these assumptions, two strategies come naturally. $TIT - FOR - TAT$ behavior consists in choosing the opponent's strategy if the player loses. ***PAVLOV behavior*** consists in taking the best response against the strategy met by the player if he lost, and keeping the same strategy if he wins. Here, our players are following the Pavlovian behavior.

Is is natural to suppose that this is given by some behavior rules:

$$x(t) = f(X_{t-1}, Y_{t-1}),$$

$$y(t) = g(X_{t-1}, Y_{t-1})$$

for some particular functions f and g.

The Specific Case of the Prisoner's Dilemma The question of the best behavior rule to use for the prisoner's dilemma gave birth to an important literature. In particular, after the book [Axe84], that describes the results of tournaments of behavior rules for the iterated prisoner's dilemma, and that argues that there exists a best behavior rule called $TIT - FOR - TAT$. This consists in cooperating at the first step, and then do the same thing as the adversary at subsequent times.

The $PAVLOV$ behavior for the Prisoner's Dilemma is the following: a player cooperates if and only if both players opted for the same alternative in the previous move. This name [Axe84, KK88, NS93] stems from the fact that this strategy embodies an almost reflex-like response to the payoff: it repeats its former move if it was rewarded by a positive score, but switches behavior if it was punished by receiving a negative one. Refer to [NS93] for some study of this strategy in the spirit of Axelrod's tournaments.

The $PAVLOV$ behavior can also be termed *WIN-STAY, LOSE-SHIFT* since if the play on the previous round results in a success, then the agent plays the same strategy on the next round. Alternatively, if the play resulted in a failure the agent switches to another action [Axe84, NS93].

Remark 2.7 *In the Prisoner's dilemma, if players i and j have a PAVLOV behavior, It is easy to see that this corresponds to executing the following rules:*

$$\begin{array}{l} C\ C \to C\ C \\ C\ D \to D\ D \\ D\ C \to D\ D \\ D\ D \to C\ C \end{array}$$

$PAVLOV$ behavior is Markovian: a behavior f is *Markovian*, if $f(X_{t-1}, Y_{t-1})$ depends only on $x(t-1)$ and $y(t-1)$. From such a behavior, it is easy to obtain a distributed dynamic. For example, let's follow [DGG$^+$02], for the prisoner's dilemma over a graph.

Suppose that we have a connected graph $G = (V, E)$, with N vertices. The vertices correspond to players. An instantaneous configuration of the system is given by an element of $\{C, D\}^N$, that is to say by the state C or D of each vertex. At each time t, one chooses randomly and uniformly one edge (i, j) of the graph. At this moment, players i and j play the prisoner dilemma with the $PAVLOV$ behavior.

What is the final state reached by the system? The underlying model is a very large Markov chain with 2^N states. The state $E^* = \{C\}^N$ is absorbing. If the graph G does not have any isolated vertex, this is the unique absorbing state, and there exists a sequence of transformations that transforms any state E into this state E^*. As a consequence, from well-known classical results in Markov chain theory, whatever the initial configuration is, with probability 1, the system will eventually be in state E^* [Bré01]. The system is *self-stabilizing*.

Several results about the time of convergence towards this stable state can be found in [DGG+02], and [FMP04], for rings, and complete graphs.

What is interesting in this example is that it shows how to go from a game, and a behavior to a distributed dynamic on a graph, and in particular to a population protocol when the graph is a complete graph.

2.1.4 From Games To Population Protocols

As explained, to any symmetric game, we can associate a population protocol. Here is how we perform it:

Definition 2.8 *Assume a symmetric two-player game is given by its set S of strategies and its matrix A. Let Δ be some threshold.*

The **Associated Protocol to the Game** *is a population protocol whose set of states is Q, where $Q = S$ is the set of strategies of the game, and whose transition rules through δ are given as follows:*
$\delta(q_1, q_2) = (q_1', q_2')$ *where*

- $q_1' = q_1$ *when* $A_{q_1,q_2} \geq \Delta$
- $q_1' \in BR_{\neq q_1}(q_2)$ *when* $A_{q_1,q_2} < \Delta$

and

- $q_2' = q_2$ *when* $A_{q_2,q_1} \geq \Delta$
- $q_2' \in BR_{\neq q_2}(q_1)$ *when* $A_{q_2,q_1} < \Delta$.

Remark 2.9

- *By subtracting Δ from each entry of the matrix M, we can assume without loss of generality that $\Delta = 0$. We will do so from now on.*
- *A Population Protocol obtained from a game as above is symmetric (a formal definition will be provided in Section 2.2.2).*

- *We assume here that the Best Response is unique, to keep the determinism of the protocol. It could be interesting to consider the cases where there can be more than one best response. This consideration will not be examined in this document.*

Definition 2.10 *A population protocol is* **Pavlovian** *if it can be obtained from a game as above.*

Here is a proof that some protocols cannot be seen as games:

Proposition 2.11 *Some symmetric population protocols are not Pavlovian.*

Proof: Consider for example a deterministic 3-states population protocol with set of states $Q = \{q_0, q_1, q_2\}$ and a joint transition function δ such that $\delta_1(q_0, q_0) = q_1$, $\delta_1(q_1, q_0) = q_2$, $\delta_1(q_2, q_0) = q_0$, with δ_1 being the projection over the first element of δ.

Assume by contradiction that there exists a 2-player game corresponding to this 3-states population protocol. Consider its payoff matrix A. Let $A_{q_0,q_0} = \beta_0$, $A_{q_1,q_0} = \beta_1$, $A_{q_2,q_0} = \beta_2$. We must have $\beta_0 < 0$, $\beta_1 < 0$ and $\beta_2 < 0$ since all agents that interact with an agent in state q_0 must change their state. Now, since q_0 changes to q_1, q_1 must be a strictly better response to q_0 than q_2: hence, we must have $\beta_1 > \beta_2$. In a similar way, since q_1 changes to q_2, we must have $\beta_2 > \beta_0$, and since q_2 changes to q_0, we must have $\beta_0 > \beta_1$. From $\beta_1 > \beta_2 > \beta_0 > \beta_1$ we reach a contradiction. □

This indeed motivates the following study, where we discuss which problems admit a Pavlovian solution.

2.2 Motivation of the Restrictions

2.2.1 Asymmetric Games

To motivate the use of symmetric games, here is a view of the works published with Olivier Bournez *et al.* in [BCC+11]. I provide here only a quick review of its results, as I want to focus on this document on the symmetric case.

We consider in [BCC+11] games that can be asymmetric. Instead of having a single payoff matrix, there is the ***initiator***'s one A and the ***responder***'s one B.

We consider games where both players have the same set S of strategies. We note $BR_A(q)$ (resp. $BR_B(q)$) the best response for the initiator (resp. responder) to a strategy q.

We will give the result that says that any basic predicate computable by a population protocol can be performed by an asymmetric game.

Definition 2.12 *Assume an asymmetric two-player game is given. Let A and B be the corresponding matrices.*

The **Associated Protocol to the Asymmetric Game** *is a population protocol whose set of states is Q, where $Q = S$ is the set of strategies of the game, and whose transition rules through δ are given as follows:*
$\delta(q_1, q_2) = (q'_1, q'_2) \in \delta$ *where*

- $q'_1 = q_1$ *when* $A_{q_1,q_2} \geq 0$,
- $q'_1 \in BR_A(q_2)$ *when* $A_{q_1,q_2} < 0$,

and

- $q'_2 = q_2$ *when* $B_{q_2,q_1} \geq 0$,
- $q'_2 \in BR_B(q_1)$ *when* $B_{q_2,q_1} < 0$.

Definition 2.13 *A population protocol is* **Asymmetric Pavlovian** *if it can be obtained from a game as above.*

With those definitions, we have the following results:

Proposition 2.14 *Any Modulo Predicate and any Threshold Predicate can be computed by an asymmetric pavlovian population protocol.*

Sketch of the proof : I will not provide the exact protocols. I will just explain the main tool we used: There exist two matrices A and B permit to compute an addition. I provide here a view of row k for some $k > 0$. The boundaries for k will depend on the predicate.

$$\begin{array}{ll|ccccc|}
\text{A} & & \multicolumn{5}{c}{\text{Opponent}} \\
 & & \dots & k\text{-}1 & k & k{+}1 & \dots \\
\cline{3-7}
 & \dots & -1 & -1 & -1 & -1 & -1 \\
 & k\text{-}1 & \mathbf{1} & -1 & -1 & -1 & -1 \\
\text{Player} & k & 0 & \mathbf{1} & -1 & -1 & -1 \\
 & k{+}1 & 0 & 0 & \mathbf{1} & -1 & -1 \\
 & \dots & 0 & 0 & 0 & \mathbf{1} & -1 \\
\cline{3-7}
\end{array}$$

$$\begin{array}{ll|ccccc|}
\text{B} & & \multicolumn{5}{c}{\text{Opponent}} \\
 & & \dots & k\text{-}1 & k & k{+}1 & \dots \\
\cline{3-7}
 & \dots & -1 & \mathbf{1} & 0 & 0 & 0 \\
 & k\text{-}1 & -1 & -1 & \mathbf{1} & 0 & 0 \\
\text{Player} & k & -1 & -1 & -1 & \mathbf{1} & 0 \\
 & k{+}1 & -1 & -1 & -1 & -1 & \mathbf{1} \\
 & \dots & -1 & -1 & -1 & -1 & -1 \\
\cline{3-7}
\end{array}$$

With those two matrices, we have the two following rules with δ:

$$\begin{array}{llll}
n\ m \rightarrow & n\ m & \text{with } n > m \\
n\ m \rightarrow & m{+}1\ n\text{-}1 & \text{with } n \leq m
\end{array}$$

□

To be able to have the boolean combination of predicates, we added the notion of multi-games as follows:

Definition 2.15 *Consider k (possibly asymmetric) two-player games. Given game i, let Q^i be the corresponding states, A^i and B^i the corresponding matrices.*

The associated **Multi-Game Population Protocol** *is the population protocol whose set of states is $Q = Q^1 \times Q^2 \times \ldots \times Q^k$, and whose transition rules are given as follows: $\delta((q_1^1, \ldots, q_1^k), (q_2^1, \ldots, q_2^k)) = ((q_1^{1\prime}, \ldots, q_1^{k\prime}), (q_2^{1\prime}, \ldots, q_1^{k\prime}))$ where, $\forall 1 \leq i \leq k$, $q_1^i\ q_2^i \rightarrow q_1^{i\prime}\ q_2^{i\prime}$ is a transition of the Asymmetric Pavlovian population protocol associated to the ith game.*

Proposition 2.16 *The class of predicates computable by multi-games are closed under boolean operations.*

Theorem 2.17 ([BCC+11]) *Sets computable by Multi-Game Population Protocols correspond exactly to semilinear sets.*

This result proves that the weakness of asymmetric games is not strong enough, as the computational power remains the same. This is why we focus now only on symmetric games.

2.2.2 Symmetric Rules

In this section, we show that restricting the rules to be symmetric does not actually change the power of computation of the model. Hence, any weakness of pavlovian population protocols cannot come from the restriction of symmetry on the rules.

Definition 2.18 *A Population protocol is* **Symmetric** *if,*

- *for all q_1, there exists some q_1' such that $\delta(q_1, q_1) = (q_1', q_1')$.*
- *for all q_1 and q_2, $\delta(q_1, q_2) = (q_1', q_2') \Rightarrow \delta(q_2, q_1) = (q_2', q_1')$.*

Proposition 2.19 ([BCC+13]) *Any Population Protocol can be simulated by a symmetric population protocol, as soon as the population is of size ≥ 3.*

Proof: To a population protocol $(Q, \Sigma, \iota, Y, \omega, \delta)$, we associate the symmetric population protocol $(Q \times \{1, 2\}, \Sigma, \iota', \omega', \delta')$.

- $\iota'(s) = (\iota(s), 1)$
- $\omega'(q, i) = \omega(q)$
- δ' is constructed by associating new rules adapted from the rules of δ. It is defined such to get the following rules (and their symmetric versions):

$$\begin{array}{llll}
(p,1)\ (q,1) \rightarrow (p,2)\ (q,2) & \forall p, q \in Q \\
(p,2)\ (q,2) \rightarrow (p,1)\ (q,1) & \forall p, q \in Q \\
(p,1)\ (q,2) \rightarrow (p',1)\ (q',2) & \forall p, q \in Q, \text{ with } \delta(p,q) = (p',q') \\
(p,2)\ (q,1) \rightarrow (p',2)\ (q',1) & \forall p, q \in Q, \text{ with } \delta(q,p) = (q',p')
\end{array}$$

The idea of this protocol is to duplicate each agent, using a second element in $\{1,2\}$. When two agents meet, if they have the same second element, they both switch it. If they have different second element, they interact as if the agent having a 1 was the left agent and the other was the right agent.

This protocol is symmetric, as the case $\delta'((q,i),(q,i))$ is solved in the first two rules and keep the symmetry. The comparison of $\delta'((p,i),(q,j))$ and $\delta'((q,j),(p,i))$ is also solved by the construction.

We need now to prove that this protocol computes exactly the same thing that the original. It is immediate, as an agent's first element changes in the second protocol only by applying δ to both interacting agents' first element.

Conversely, any rule from δ can be applied to a pair of agents a_1 and a_2 in the second protocol: If they have different second part, it suffices to have them interacted. If they have the same second part in $\{1,2\}$, the presence of a third agent a_3 permits to have the interaction happened: If a_3 has the same second part, a_1 and a_3 interacting together will switch their second part and we come back to the previous case. If a_3 has the reversed second part, we first have a_1 and a_2 interacted, then the 3 agents will have the same second part.

As the output function corresponds to the one of the first protocol on the first element, these protocols do compute the same thing. $\square$

With this result, we deduce the following corollary:

Corollary 2.20 *A predicate is computable by a symmetric population protocol if and only if it is semilinear.*

Proof: The problem of the previous transformation is that in a population of size 2, the protocol never evolves, as only the two first rules will apply in a loop. Let $\mathcal{P}$ be the predicate we want to compute.

To adapt, it suffices to run the protocol $((NOT\ P_{\geq 3}) \wedge P_a) \vee (P_{\geq 3} \wedge P_b)$, where:

- $P_{\geq 3}$ is a symmetric version of a protocol checking if the population has a size ≥ 3. It can be seen in Section 2.3.1, in Theorem 2.28. This protocol provides the output False if their are only 2 agents.
- P_a is a protocol that acts as if the population is of size 2. $Q_a = \Sigma \cup \{True, False\}$ and $\forall s \in \Sigma$, $\iota_a(s) = s$. δ_a is such as the rules are:

$$\begin{array}{lll} s_1\ s_2 & \rightarrow True\ True & \text{if } s_1 s_2 \in \mathcal{P} \\ s_1\ s_2 & \rightarrow False\ False & \text{if } s_1 s_2 \notin \mathcal{P} \\ q\ False & \rightarrow False\ False & \forall q \in Q \end{array}$$

  This protocol provides the right output for populations of size 2. It does not compute necessarily something for inputs of size greater.

- P_b is the transformed protocol according to the previous proposition.

$\square$

2.2.3 Unique Best Response

Before describing more complicate protocols, here is a motivation of why we use the best response different from the already used strategy:

Definition 2.21 *A Pavlovian Population Protocol is with* **Unique Best Responses** *if, for all strategy q, $A_{BR(q),q} \geq 0$.*

We shew in the previous section that this restriction is not an obstacle to compute semilinear sets with asymmetric games. With the symmetric games, we have the following negative result.

Theorem 2.22 *There is no Pavlovian protocol with unique best responses that computes the threshold predicate $[x_A \geq 3]$: That is to say the predicate which is true when there are at least 3 occurrences of input symbol A in the input x.*

Proof: We will prove this by contradiction. Assume there exists such a Pavlovian protocol. Without loss of generality we may assume that $\Sigma = \{0, A\}$ is a subset of the set of states Q.

As the protocol is Pavlovian, and hence symmetric, any rule $qq \to q'q''$, is such that $q' = q''$, that is to say of the form $qq \to q'q'$ for all $q \in Q$.

Let us consider the sequence of rules such that $AA \to q_1q_1 \to q_2q_2 \to \cdots \to q_kq_k \to \dots$ where $A, q_1q_2, q_3, \dots, q_k \in Q$.

Since Q is finite, there exist two distinct integers k and ℓ such that $q_k = q_\ell$ and $k < \ell$.

Let suppose $k+1 < \ell$, we have $q_kq_k \to q_{k+1}q_{k+1} \to \cdots \to q_\ell q_\ell \to q_kq_k$. Let T be the set of states $T = \{q_i : k \leq i \leq \ell\}$.

Since $q_iq_i \to q_{i+1}q_{i+1}$ is among the rules, since the protocol is Pavlovian with a matrix A, and by definition of Pavlov behavior, we have $q_{i+1} = BR_{\neq q_i}(q_i)$ (with the convention that $q_{\ell+1}$ is q_k). So, $BR(q_i) = q_{i+1}$ (because of the uniqueness of the best response).

Let us discuss the rules $q_iq_j \to q_i'q_j'$ for $q_i, q_j \in T$.
There are two possibilities for the value of q_i':

1. $q_i' = q_i$ if $A_{q_iq_j} \geq 0$;
2. $q_i' = BR_{\neq q_i}(q_j) = BR(q_j) = q_{j+1}$ if $A_{q_iq_j} < 0$.

In any case, we see that the value of q_i' is in T.

Symmetrically, we have two possibilities for q_j', both of them in T.

Hence, all rules of the form $q_iq_j \to q_i'q_j'$ preserve T: we have $q_i', q_j' \in T$, as soon as $q_i, q_j \in T$.

Consider the inputs x_1 and x_2 such that $x_1 = \{A, A\}$ and $x_2 = \{A, A, A, A\}$. From x_2 there is a derivation $x_2 \to \{q_1, q_1, A, A\} \to \{q_1, q_1, q_1q_1\} \to^* \{q_k, q_k, q_k, q_k\}$. From this last configuration, by the previous remark, the state of all agents will be in T. As x_2 must be accepted, ultimately all agents will be in states that belong to T whose image by ω is *True*. Consider now x_1. From x_1 there is a derivation $x_1 \to \{q_1, q_1\} \to^* \{q_k, q_k\}$ that will go through all configurations $\{q_iq_i\}$, for all $q_i \in T$ in turn. This cannot eventually stabilize to elements whose image by ω is $False$, as some of the elements of T have image 1 by ω, and hence x_1 is not accepted. This yields a contradiction, we cannot have $k+1<l$.

If $k+1=\ell$, we have $x_1 \to \{q_1,q_1\} \to^* \{q_k,q_k\}$ and $x_2 \to \{q_1,q_1,A,A\} \to \{q_1,q_1,q_1q_1\} \to^* \{q_k,q_k,q_k,q_k\}$. The two configurations $\{q_k,q_k\}$ and $\{q_k,q_k,q_k,q_k\}$ are stable, i.e. any step will not change the configuration. They have the same output, which brings a contradiction with the fact that x_1 cannot be accepted and x_2 must be accepted.

Hence such a Pavlovian protocol cannot exist. □

2.3 Protocols and Properties

2.3.1 Some Simple Pavlovian Protocols

We start here with two easy protocols. First, a statement about any 2-state protocol, then a game permitting to compute $[x_A \geq 3]$. We finish by the Majority Problem seen in Example 1.7.

Theorem 2.23 *Any symmetric 2-states population protocol is Pavlovian.*

Proof: Consider a deterministic symmetric 2-states population protocol. Note $Q=\{+,-\}$ its set of states. There exists $\alpha_{++},\alpha_{+-},\alpha_{-+},\alpha_{--} \in Q^4$ such that the rules can be written as follows:

$$\begin{array}{l}
+\ + \to \alpha_{++}\ \alpha_{++}\\
+\ - \to \alpha_{+-}\ \alpha_{-+}\\
-\ + \to \alpha_{-+}\ \alpha_{+-}\\
-\ - \to \alpha_{--}\ \alpha_{--}
\end{array}$$

This corresponds to the symmetric game given by the following pay-off matrix A

$$\begin{array}{ccc}
 & & \text{Opponent} \\
 & & \begin{array}{cc} + & - \end{array} \\
\text{Player} & \begin{array}{c} + \\ - \end{array} & \begin{array}{|cc|} \beta_{++} & \beta_{+-} \\ \beta_{-+} & \beta_{--} \end{array}
\end{array}$$

where for all $q_1,q_2 \in \{+,-\}$,

- $\beta_{q_1q_2}=1$ if $\alpha_{q_1q_2}=q_1$,
- $\beta_{q_1q_2}=-1$ otherwise.

To check this, we just need to notice that :

- if $\alpha_{q_1q_2}=q_1$, the first agent keeps its state. With $\beta_{q_1q_2}=1$, $A_{q_1,q_2}\geq 0$, so the first agent will keep its state.

- if $\alpha_{q_1 q_2} \neq q_1$, the first agent switches its state. With $\beta_{q_1 q_2} = -1$, $A_{q_1,q_2} < 0$, so the first agent will chose $BR_{\neq q_1}(q_2)$. Because there are only 2 states, the agent's new state with our pavlovian protocol will be the same that the one with the original symmetric protocol.

The above proof is sufficient for the second agent because the protocol is symmetric. □

Proposition 2.24 *There is a Pavlovian protocol that computes the predicate* $[x_A \geq 2]$*, which is true when there are at least* 2 *occurrences of input symbol* A *in the input* x*.*

Proof: The following protocol is a solution considering

- $\Sigma = \{0, A\}$, $Q = \{0, A, 2\}$,
- $\omega(0) = \omega(A) = False$,
- $\omega(2) = True$.

$$\begin{array}{rl}
0\ 0 \rightarrow & 0\ 0 \\
0\ A \rightarrow & 0\ A \\
A\ 0 \rightarrow & A\ 0 \\
0\ 2 \rightarrow & 2\ 2 \\
2\ 0 \rightarrow & 2\ 2 \\
A\ A \rightarrow & 2\ 2 \\
A\ 2 \rightarrow & 2\ 2 \\
2\ A \rightarrow & 2\ 2 \\
2\ 2 \rightarrow & 2\ 2
\end{array}$$

Indeed, if there are at least two A, then by fairness and by the rule on line 6, they will ultimately be changed into two 2s. Then 2s will turn all other agents into 2s. Now, this is the only way to create a 2.

This is a Pavlovian protocol since it corresponds to the following payoff matrix:

$$\begin{array}{cc|ccc|}
 & & & \text{Opponent} & \\
 & & 0 & A & 2 \\
\hline
 & 0 & 0 & 0 & -1 \\
\text{Player} & A & 0 & -1 & -1 \\
 & 2 & 1 & 1 & 1 \\
\hline
\end{array}$$

□

Proposition 2.25 *The Majority Problem $[x_A > x_B]$ (see Example 1.7: given some population of input symbols A and B, determine whether there are more A than B) can be solved by a Pavlovian population protocol.*

Proof:

We claim that the protocol provided in Example 1.7 with a small change (the first rule has been modified) can be extracted from a game. The change does not alters what the protocol computes. To be convinced, here arel the rules (we assume that δ also provides their symmetric version). We then provide a Matrix of a game corresponding to the rules:

$$\begin{array}{l} A\ B \rightarrow b\ a \\ A\ b \rightarrow A\ a \\ B\ a \rightarrow B\ b \\ a\ b \rightarrow b\ b \end{array}$$

$$\begin{array}{cc|cccc|} & & \multicolumn{4}{c}{\text{Opponent}} \\ & & \text{b} & \text{a} & A & B \\ \cline{3-6} & \text{b} & 1 & 0 & -1 & 1 \\ \text{Player} & \text{a} & -1 & 0 & 1 & -1 \\ & A & 0 & 0 & 0 & -1 \\ & B & 0 & 0 & -1 & 0 \\ \cline{3-6} \end{array}$$

□

2.3.2 Some Structural Properties on Pavlovian Rules

We are now going to describe some not so simple Pavlovian protocols. Before doing so, and in order to help to prove that a given set of rules is Pavlovian, without building explicitly possibly intricate matrices, we start with some structural properties on Pavlovian protocols.

Proposition 2.26 *Consider a set of rules. For all rule $p\ q \rightarrow p'\ q'$, denote $\delta_q(p) = p'$. Since we consider symmetric rules, we then have symmetrically $\delta_p(q) = q'$.*

Let

$$Stable(q) = \{p \in Q | \delta_q(p) = p\}.$$

Then the set of rules is Pavlovian iff there exists a function $max : Q \rightarrow Q$ such that $\forall a \in Q$:

- *$Stable(q) \neq \emptyset$ implies that $max(q) \in Stable(q)$*
- *$\forall p \notin Stable(q), p \neq max(q)$ implies $\delta_q(p) = max(q)$.*

Proof: First, we consider a Pavlovian population protocol P obtained from corresponding matrix A. For each state q in Q, let a be the best response to strategy q for matrix A (i.e $a = BR(q)$). We set $max(q) = a$, i.e. max corresponds to BR.

If $Stable(q)$ is not empty, there exists some p such that $p\ q \to p\ q'$. By Definition 2.8, we have $A_{p,q} \geq 0$. $A_{BR(q),q} \geq A_{p,q} \geq 0$, so $BR(q) = max(q) \in Stable(q)$.

Let $p \notin Stable(q), p \neq max(q)$. We have $A_{p,q} < 0$ (otherwise, $p \in Stable(q)$). We focus on the rule $p\ q \to p'\ q'$. $A_{p,q} < 0$ implies $p' \neq p$. We have $p' = BR_{\neq p}(q)$. Because $p \neq max(q)$, $p \neq BR(q)$, so $BR_{\neq p}(q) = BR(q)$. We deduce that $\delta_q(p) = max(q)$.

The function max with the restrictions exists.

Conversely, consider a population protocol P satisfying the properties of the proposition. All rules $p\ q \to p'\ q'$ are such that $\delta_q(p) = p'$ and $\delta_p(q) = q'$. We focus on the construction on a two-player game having the corresponding matrix A.

- If $Stable(q) = Q$, then let $\forall p \in Q$, $A_{p,q} = 0$.
- If $Stable(q) = \emptyset$, then let $A_{max(q),q} = -1$ and $A_{\delta_q(max(q)),q} = -2$. Moreover, $A_{p,a} = -3$ for each state p different to $max(q)$ and $\delta_q(max(q))$.
- If $\emptyset \subset Stable(q) \subset Q$ then the value $A_{p,q}$ of each element p depends on the set $Stable(q)$ and $max(q)$.
  - Take $A_{max(q),q} = 1$.
  - If $p \in Stable(q)$ and if $p \neq max(q)$, then let $A_{p,q} = 0$.
  - If $p \notin Stable(q)$, then let $A_{p,q} = -1$.

It is easy to see that this game describes all rules of P. So, P is a Pavlovian population Protocol. $\square$

Remark 2.27 *As a consequence, a protocol can be given by describing for any state q:*

1. *the set $Stable(q)$,*
2. *the value of $max(q)$,*
3. *and whenever $Stable(q) = \emptyset$, then the value of $\delta_q(max(q))$.*

Note that if $Stable(q)$ is not $\emptyset$, then $\delta_q(max(q)) = max(q)$ otherwise $\delta_q(max(q)) \neq max(q)$.

We will then provide non-trivial pavlovian protocols, described using this framework.

2.3.3 Counting up to 3

With two states, it is possible to compute $[x_A = 0]$ and $[x_A \geq 1]$. We also gave $[x_A \geq 2]$. I conjecture that we can actually compute with a pavlovian protocol $[x_A \geq b]$ for any $b \in \mathbb{N}$. We give here the case $b = 3$. The general case seems too difficult and we did not succeed to provide protocols, but the next section gives a hard protocol for the case $b = 2^k$ for any $k \geq 2$.

By providing a protocol computing $[x_A \geq 3]$, we prove in this section that the version with a unique best response is strictly weaker than the version without.

Proposition 2.28 *There is a Pavlovian protocol that computes the predicate $[x_A \geq 3]$, which is true when there are at least 3 occurrences of input symbol A in the input x.*

Proof:

Let us consider the following protocol:

- $Q = \{0, 1, 2_+, 2_-, X, Y, \top\}$.
- $A \in \Sigma$.
- $\iota(A) = 1$, $\forall s \neq A$, $\iota(s) = 0$.
- $\omega(\top) = True$, $\forall q \neq \top$, $\omega(q) = False$.
- δ is such that the rules are as follows (by symmetry of the protocol, symmetric rules are not indicated but can be deduced):

$$\begin{array}{l} 0\ 0 \to 0\ 0 \\ 0\ 1 \to 0\ 1 \\ 0\ 2_+ \to 2_-\ 0 \\ 0\ 2_- \to 2_+\ 0 \\ 0\ X \to 0\ X \\ 0\ Y \to 0\ Y \\ 0\ \top \to \top\ \top \end{array} \quad \begin{array}{l} 1\ 1 \to 2_+\ 2_+ \\ 1\ 2_+ \to 2_-\ 2_- \\ 1\ 2_- \to 2_+\ 2_- \\ 1\ X \to \top\ X \\ 1\ Y \to \top\ Y \\ 1\ \top \to \top\ \top \end{array} \quad \begin{array}{l} 2_+\ 2_+ \to 2_-\ 2_- \\ 2_+\ 2_- \to X\ Y \\ 2_+\ X \to \top\ 2_- \\ 2_+\ Y \to \top\ 2_- \\ 2_+\ \top \to 2_+\ 2_- \end{array} \quad \begin{array}{l} 2_-\ 2_- \to 2_+\ 2_+ \\ 2_-\ X \to \top\ 2_+ \\ 2_-\ Y \to \top\ 2_+ \\ 2_-\ \top \to 2_-\ 2_+ \end{array}$$

$$\begin{array}{l} X\ X \to \top\ \top \\ X\ Y \to X\ Y \\ X\ \top \to \top\ \top \end{array} \quad \begin{array}{l} Y\ Y \to \top\ \top \\ Y\ \top \to \top\ \top \end{array} \quad \begin{array}{l} \top\ \top \to \top\ \top \end{array}$$

It is Pavlovian as it corresponds to the following payoff matrix :

					Opponent			
		0	1	2_+	2_-	X	Y	$\top$
	0	1	0	-3	-3	0	0	-1
	1	0	-1	-3	-3	-1	-1	-1
	2_+	-1	1	-3	-1	-1	-1	0
Player	2_-	-1	0	-1	-3	-1	-1	0
	X	0	0	-2	-3	-1	0	-1
	Y	0	0	-3	-2	0	-1	-1
	$\top$	0	0	-3	-3	1	1	1

As said in the previous remark, it can be described by the parameters $Stable(a)$, $max(a)$, $\delta_a(max(a))$ for each state a:

q	$Stable(q)$	$max(q)$	$\delta_q(max(q))$
0	$\{0, 1, X, Y, \top\}$	0	0
1	$\{0, 2_+, 2-, X, Y, \top\}$	2_+	2_+
2_+	$\emptyset$	2_-	X
2_-	$\emptyset$	2_+	Y
X	$\{0, Y, \top\}$	$\top$	$\top$
Y	$\{0, X, \top\}$	$\top$	$\top$
$\top$	$\{2_+, 2_-, \top\}$	$\top$	$\top$

Now, we prove that this Pavlovian protocol computes the threshold predicate $[x_A \geq 3]$ using the number of occurrences of input symbol A. In this proof, to point two interacting agents, we underline them before the transition (for example, $\underline{XX}Y \rightarrow \top\top Y$).

[x_A =**0**]**:** If there is no occurrence of input symbol A in the input x, then the starting configuration is (0. . . 0). In this case, no interaction allows agents to change their state.

[x_A =**1**]**:** This argument can be also applied for the case where there is one occurrence of input symbol A in the input x: the starting configuration is (1)(0. . . 0).

[x_A =**2**]**:** Now we consider the case where there are two occurrences of input symbol A in the input x. If the number of agents is 2, then after the first interaction, the system switches between two configurations (2_+2_+) and (2_-2_-). Otherwise, the first interaction that changes a state is the interaction between two agents with state 1. From the initial configuration 110, there is a derivation $\underline{11}0 \rightarrow \underline{2_+2_+}0 \rightarrow \underline{2_-2_-}0 \rightarrow 2_+\underline{2_+0} \rightarrow \underline{2_+}0\underline{2_-} \rightarrow Y0X$ where the states underlined represent the next interaction. From this situation, no interaction allows agents to change their state. Furthermore, these configurations are the only possibly reachable : if in configuration 2_+02_- the 0 were to interact with 2_+, the system would only go back to the previously reached configuration 2_-2_-0.

[x_A ≥**3**]**:** Now, we consider the case where there are at least three occurrences of input symbol A in the input x. First of all, the number of the agents being in state 0 never increases and so there are at least three elements in $\{1, 2_+, 2_-, X, Y, \top\}$. Additionally no agent can become in state 1 from any other given state.

So after a first $11 \to 2_+2_+$ we can be sure that there are always at least 2 agents with states in $\{2_+, 2_-, X, Y, \top\}$.

Third remark, we can notice that X and Y, as 2_+ and 2_-, have symmetric roles.

We will prove now that any triplet of $\{1, 2_+, 2_-, X, Y, \top\}$ with at most one 1 can reach three $\top$. With this proved, the fairness will permit to conclude that a configuration with only $\top$ will always be reached, as $0\top \to \top\top$. By using the symmetry of roles, some triplets will not be treated.

- $\underline{12_i}2_i \to \underline{2_{\bar{i}}2_i}2_i \to X\underline{Y2_i} \to X\underline{2_{\bar{i}}\top} \to X\underline{2_{\bar{i}}2_i} \to \underline{XX}Y \to \top\underline{\top Y} \to \top\top\top$ with $i \in \{+,-\}$
- $1\underline{2_i\top} \to 1\underline{2_i2_{\bar{i}}} \to \underline{1X}Y \to \top\underline{\top Y} \to \top\top\top$ with $i \in \{+,-\}$
- $1\underline{2_iX} \to 1\underline{2_+2_-} \to \underline{1X}Y \to \underline{\top X}Y \to \top\underline{\top Y} \to \top\top\top$
- $1\underline{XX} \to \underline{1\top}\top \to \top\top\top$
- $1\underline{X\top} \nearrow \qquad \swarrow 2_{\bar{i}}\underline{2_i\top} \leftarrow \underline{2_i\top}\top \leftarrow 2_i\underline{XX}$
- $\underline{2_i2_i}2_i \to 2_{\bar{i}}\underline{2_{\bar{i}}2_i} \to \underline{2_{\bar{i}}X}Y \to \underline{\top 2_i}Y \to \underline{2_{\bar{i}}2_i}Y \to X\underline{YY} \to \underline{X\top}\top \to \top\top\top$
- $2_{\bar{i}}\underline{2_{\bar{i}}X} \nearrow \qquad \nwarrow 2_{\bar{i}}\underline{2_{\bar{i}}\top} \qquad\qquad X\underline{XX} \nearrow \qquad \nwarrow X\underline{Y\top}$
- $\underline{XX}\top \to \top\top\top$

All the cases being proved to reach $\top\top\top$, we can conclude by fairness that a configuration with only $\top$ will be reached, providing the right output. $\square$

2.3.4 Counting up to 2^k

We now provide the most complicated Pavlovian Protocol we have been able to create. The classical protocol for a threshold predicate is quite easy, but cannot be directly turned into pavlovian. We had to create a more tricky protocol to compute $[x_A \geq 2^k]$ that the one in Example 1.10 to make it pavlovian.

Theorem 2.29 *For any $k \in \mathbb{N}$, there is a Pavlovian protocol that computes the predicate $[x_A \geq 2^k]$, which is true when there are at least 2^k occurrences of input symbol A in the input x.*

The remaining of this section is devoted to prove this Theorem.

The case $k = 0$ is a 2-state protocol, and is handled by Theorem 2.23, the case $k = 1$ has been treated in Proposition 2.24. We now prove that the following protocol P is a solution for $k \geq 2$.

- $Q = \{0, \top\} \bigcup_{i=1}^{2^{k-1}-1} \{i_+, i_-\}$.

- $\Sigma = \{0, A\}$.
- $\iota(A) = 1_+$, $\iota(0) = 0$.
- $\omega(\top) = True$, $\forall q \neq \top$, $\omega(q) = False$.
- Its transition function can be written as follows:

$$\begin{array}{lll} 0\ 0 \rightarrow 0\ 0 & n_-\ m_+ \rightarrow m_-\ n_+ & \\ 0\ n_+ \rightarrow n_-\ 0 & n_+\ m_- \rightarrow m_+\ n_- & \text{whenever } m \neq n \\ 0\ n_- \rightarrow n_+\ 0 & n_-\ m_- \rightarrow m_+\ n_+ & \\ 0\ \top \rightarrow \top\ \top & n_+\ m_+ \rightarrow m_-\ n_- & \end{array}$$

$$\begin{array}{lll} n_+\ n_+ \rightarrow n_-\ n_- & n_-\ n_+ \rightarrow (2n)_+\ (2n+1)_+ & \text{whenever } n < 2^k \\ n_-\ n_- \rightarrow n_+\ n_+ & n_-\ n_+ \rightarrow \quad \top\ \top & \text{whenever } 2^{k-1} \leq n < 2^{k-1} \end{array}$$

$$\begin{array}{ll} n_-\ \top \rightarrow n_-\ n_+ & \\ n_+\ \top \rightarrow n_+\ n_- & \top\ \top \rightarrow \top\ \top \end{array}$$

This system is Pavlovian because we can define $Stable(q), m(q)$ and $\delta_q(m(q))$ for any state q as follows:

q	$Stable(q)$	$max(q)$	$\delta_q(max(q))$
0	$\{0, \top\}$	0	0
$\top$	$Q \setminus \{0\}$	$\top$	$\top$
n_+ (with $n < 2^{k-1}$)	$\emptyset$	n_-	$(2n+1)_+$
n_- (with $n < 2^{k-1}$)	$\emptyset$	n_+	$(2n)_+$
n_+ (with $n \geq 2^{k-1}$)	$\emptyset$	n_-	$\top$
n_- (with $n \geq 2^{k-1}$)	$\emptyset$	n_+	$\top$

Let us consider another protocol P' which differs from P only in its transition function given by:

$$\begin{array}{lll} 0\ 0 \rightarrow 0\ 0 & n_-\ m_+ \rightarrow n_+\ m_- & \\ 0\ n_+ \rightarrow 0\ n_- & n_+\ m_- \rightarrow n_-\ m_+ & \text{whenever } m \neq n \\ 0\ n_- \rightarrow 0\ n_+ & n_-\ m_- \rightarrow n_+\ m_+ & \\ 0\ \top \rightarrow \top\ \top & n_+\ m_+ \rightarrow n_-\ m_- & \end{array}$$

$$\begin{array}{lll} n_+\ n_+ \rightarrow n_-\ n_- & n_-\ n_+ \rightarrow (2n)_+\ (2n+1)_+ & \text{whenever } n < 2^{k-1} \\ n_-\ n_- \rightarrow n_+\ n_+ & n_-\ n_+ \rightarrow \quad \top\ \top & \text{whenever } 2^{k-1} \leq n < 2^k \end{array}$$

$$\begin{array}{ll} n_-\ \top \rightarrow n_-\ n_+ & \\ n_+\ \top \rightarrow n_+\ n_- & \top\ \top \rightarrow \top\ \top \end{array}$$

The transition function of P' is the transition function of P in which transition rules of form $ab \to a'b'$ have sometimes been replaced by corresponding (and computationally equivalent) rule $ab \to b'a'$. The anonymity of agents in a population protocol implies that from a population protocol point of view, protocols P and P' are equivalent and compute the exact same predicate (if any). The difference is that P' is maybe not Pavlovian. However, proving that P' computes the desired predicate will yield that the same holds for P. We will now study P' instead of P in order to simplify the proof.

Let a be an arbitrary agent of the population. Let q and C be respectively a state in Q and a configuration.

Let us define $C(a)$ the state of agent a in configuration C and $C^{\#}(q)$ the number of agents in state q in configuration C.

Finally, we define $v(q)$ the level of a state q such that $v(0) = 0$, $v(n_+) = v(n_-) = n$ and $v(\top) = 2^k$.

Lemma 2.30 *For any two configurations C and C' such that $C \to C'$ in protocol P', $v(C'(a)) \geq v(C(a))$ or $C(a) = 2^k$.*

Proof: Checking every rules one by one permits to prove this lemma.

This lemma means that an agent that is not in state 2^k cannot be sent back in the execution of states which is easily verified by looking at the transition rules for P'. Note that this key property does not hold for P and will allow us to simplify the following proofs. $\square$

Lemma 2.31 *Let $C_0, ..., C_{i_1}$ be an execution of configurations such that:*
$\forall i \in [0, i_1], C_i \to C_{i+1}$ and $C_0^{\#}(\top) = ... = C_{i_1}^{\#}(\top) = 0$ and $\forall q \notin \{0, 1_+\}, C_0^{\#}(q) = 0$.
Then $\forall n, 1 \leq n \leq 2^{k-1}, \forall i \leq i_1, C_i^{\#}(n_+) + C_i^{\#}(n_-) \leq C_0^{\#}(1_+)2^{-\lfloor log(n) \rfloor}$.

Proof: We will actually prove a slightly stronger statement by induction over n :

In the execution of configurations $C_0, \ldots, C_{i_1}$ no more than $C_0^{\#}(1_+)2^{-\lfloor log(n) \rfloor}$ agents may ever reach states n_+ or n_-. Calling $S(n), n \geq 1$ the set of agents that ever reach states n_+ or n_- in this computation, we will prove that $|S(n)| \leq C_0^{\#}(1_+)2^{-\lfloor log(n) \rfloor}$.

First, we note that this is an execution of configurations in which no agent ever reaches state $\top$. Because of this and the previous lemma, an agent can only have growing value in the execution $C_0, \ldots, C_{i_1}$. In addition, agents initially in state 0 can never change their state (Looking at the rules, we see that this would only be possible if they encountered an agent in state $\top$). Thus, the set of agents with state different from 0 is the same in every configuration in the execution. It follows naturally that no more than $C_0^{\#}(1_+)$ agents may ever reach states 1_+ or 1_-.

To prove the statement for a given $n \geq 2$ assume by induction that it is true for all $k < n$. It follows from the transition rules that $S(k) \subset S\left(\left\lfloor \frac{k}{2} \right\rfloor\right)$ by looking at the state an agent was before it ever reached level k. This also holds for level n. In fact any given agent in $S(n)$ first reaches level n through an interaction of type $\frac{n}{2}_+\frac{n}{2}_- \to n_+(n+1)_+$ if n is even (and $\frac{n-1}{2}_+\frac{n-1}{2}_- \to (n-1)_+n_+$ if n is odd). Thus, it appears that at most half the agents in

$S\left(\left\lfloor\frac{n}{2}\right\rfloor\right)$ ever reach level n (the other half either being sent to level $n-1$ or $n+1$ depending on the parity of n or never going beyond level $\left\lfloor\frac{n}{2}\right\rfloor$).

Therefore $|S(n)| \leq \frac{|S(\frac{n}{2})|}{2} \leq C_0^{\#}(1_+)2^{-\lfloor log(n)\rfloor}$.

□

Lemma 2.32 *Let $C_0, C_1, \ldots, C_i, \ldots$ be a correct execution of protocol P'.*
If $C_0^{\#}(1_+) < 2^k$ then $\forall i \geq 0, C_i^{\#}(\top) = 0$.

Proof: By contradiction, let us assume that there exists at least one configuration C_i with at least one agent in state $\top$. Let us consider C_{i_0} the earliest such configuration. This means that the transition $C_{i_0-1} \to C_{i_0}$ happens through an encounter of two agents n_+n_- with $n \in [2^{k-1}...2^k-1]$.

Then $C_0, \ldots, C_{i_0-1}$ is a non-empty execution fitting the conditions in the previous lemma. Which tells us that $C_{i_0-1}^{\#}(n_+) + C_{i_0-1}^{\#}(n_-) \leq 2^{-\lfloor log(n)\rfloor}C_0^{\#}(1_+)$. Since $C_{i_0-1}^{\#}(n_+) + C_{i_0-1} \geq 2$ for this interaction to be possible, we have:
$C_0^{\#}(1_+) \geq 2^{\lfloor log(n)\rfloor} \geq 2^k$. □

We have now proved that if strictly less than 2^k agents are in state 1_+ initially, no agent will ever reach state $\top$ and thus all agents will always agree on output $False$ and the computation will be correct. We will now prove that the computation will be correct if at least $\top$ agents are initially in state 1_+.

Lemma 2.33 *For any configuration C in which at least 2^k agents are in non-zero states, there exists a configuration C' such that $C \to {}^*C'$ and $C'^{\#}(\top) \geq 1$.*

Proof: If $C^{\#}(\top) \geq 1$, just take $C' = C$. Now assume $C^{\#}(\top) = 0$, then the pigeonhole principle insures that there are at least 2 agents at the same level. Let n be the smallest level with at least 2 agents. We will now prove that there is a finite sequence of configurations forming valid computation steps that increases the value of n. We shall now differentiate the cases where $C^{\#}(n_+) + C^{\#}(n_-) > 2$ and $C^{\#}(n_+) + C^{\#}(n_-) = 2$.

If $C^{\#}(n_+) + C^{\#}(n_-) > 2$, if at least two agents in level n are in different states, we can have those two interact according to rule $n_+n_- \to (2n)_+(2n+1)_-$ to decrease the number of agents in level n by two (and not create agents in lower levels) or, if they are all in the same state, uniformity can be broken by having two of them interact via rules $n_+n_+ \to n_-n_-$ or $n_-n_- \to n_+n_+$ to create two agents in the other level-n state and be brought back to the previous configuration. By iterating on those two actions, we can bring the number of agents at level n to two or less. If only one remains, we have achieved our goal, otherwise, we handle the two remaining agents as follows: If $C^{\#}(n_+) + C^{\#}(n_-) = 2$, we again, differentiate: if $C^{\#}(n_+) = C^{\#}(n_-) = 1$ then selecting the two agents in states n_+ and n_- for an interaction $n_+n_- \to (2n)_+(2n+1)_-$ will yield the desired result. Otherwise, if both agents at level n are in the same state, then we can break this symmetry by having one interact with any other non-zero agent to come back to the previous case. Such a non-zero and non-level-n agent exists since there are at least $2^k \geq 4$ non zero agents.

Thus, from configuration C we have an execution $C \rightarrow^* C_1$ where $C_1^{\#}(n_-) < 2$. If $C_1^{\#}(\top) > 0$ we have achieved our desired result, otherwise, we can reiterate on C_1, knowing that the minimal level with at least two agents in C_1 is $n_1 > n$ by construction. Iterating this process gives us an execution $C \rightarrow^* C_1 \rightarrow^* ... \rightarrow^* C_j$ such that either an agent in state $\top$ appears or we have a corresponding execution $n < n_1 < ... < n_j$ of minimal level with at least two agents. Since this strictly growing execution is upper bounded by 2^k it is finite which guarantees that after a certain amount of iterations, at least one agent will reach state $\top$. □

For any configuration C with at least one agent in a non-zero state, we define $m(C)$ the smallest level with non-zero count in C.

Remark 2.34 *Note that $m(C)$ can never decrease: no interaction rule creates agents in a level lower than those already existing in the system.*

Lemma 2.35 *For any configuration C such that $C^{\#}(\top) \geq 1$, and $m(C) < 2^k$, there exists a configuration C' such that $C \rightarrow {}^*C'$ and $m(C') > m(C)$.*

Proof: Similarly to what was done before, if there are at least two agents at level $m(C)$ then we can reduce the number of agents at level $m(C)$ by two and, iterating the process bring it to at most 1. Note that this process can be done by preserving the existence of agents in state $\top$. If we are left with no agents in state $m(C)$, we have achieved our goal. If not, we are left with a single agent a in stat $m(C)$ and all other agents in states greater than $m(C)$, including at least one agent a' in state $\top$. Assume that a is in state n_+ (with, $n = m(C)$, the case n_- being symmetric). Then we can eliminate our final agent a through two interactions with a' :

$$n_+\top \rightarrow n_+n_- \rightarrow (2n)_+(2n+1)_+.$$

This brings us to a configuration C' such that $C'^{\#}(m(C)) = 0$ and thus, $m(C') > m(C)$. □

Lemma 2.36 *From any configuration C in which at least 2^k agents are in non-zero states, there exists a computation execution leading to a configuration in which all agents are in state $\top$.*

Proof: This is achieved mainly by iterations of the previous two lemmas: from configuration C, following Lemma 2.33, reach a configuration C' where there is at least one agent in state $\top$. If some non-zero agents are not in state $\top$, increase the minimal non-zero level in the system per Lemma 2.35. Iterate until the minimal non-zero level is 2^k, i.e. all agents are either in state $\top$ or in state 0. Then user the transition rule $0\top \rightarrow \top\top$ to convert all remaining agents from state 0 to state $\top$. Note that such a configuration is stable. □

Theorem 2.37 *Protocol P computes the predicate $[x_A \geq 2^k]$.*

Proof: From Lemma 2.36, the fairness property ensures that any fair computation of protocol P' starting in a configuration with at least 2^k agents in state 1_+ stably converges to a configuration in which all agents are in state $\top$ and thus agree on output $True$. By contradiction, if the initial configuration holds strictly less than 2^k agents in state 1_1 then Lemma 2.32 guaranties that all agents will always agree on output $False$.

Thus protocol P' computes the predicate $[x_A \geq 2^k]$ and, since they are equivalent, so does P. □

2.4 Conclusion

2.4.1 Resume

We provided here some interesting games that computes predicates:

- $[x_A \geq 3]$.
- The Majority Protocol $[x_A > x_B]$.
- $[x_A \geq 2^k]$ for any $k \in \mathbb{N}$.

We also provided proofs of the necessity to use the second best response if we want not to be too restrictive. Though, we did not provide any provably uncomputable predicate.

2.4.2 Conjectures

I have the two following conjectures about what could or could not be computed:

Conjecture 2.38 *There is no pavlovian population protocol that computes a modulo predicate of the form $[x_A \equiv b[c]]$ for any $c > b \geq 0$.*

I do not have any intuition behind, this is just the conclusion of hours spent trying to create a game computing $[x_A \equiv 1[2]]$.

Conjecture 2.39 *There is a pavlovian population protocol that computes the threshold predicate $[x_A \geq b]$ for any $b \in \mathbb{N}$.*

The intuition this time comes from the 2^k predicates. An idea of solution is to modify the protocol of $2^{\log b+1}$. We can notice that, when two agents n_+ and $(2n)_+$ meet, we are sure that there are at least $2^{\log n} + 2^{\log n+1}$ A in the input (We can prove that the $2^{\log n}$ agents that lead to the apparition of $(2n)_+$ will never be relevant in the apparition of the state n_+).

Hence, if we run the $2^{\log b+1}$, if at some point in the configuration we have agents in state 2^i and the sum of these integers is greater than b, we can be sure that there were at least b agents with input A.

The remaining problem is to find a way to "sum" these agents in a pavlovian way. Directly from the 2^k protocol, it looks not possible. My belief is that it is possible to create such a game for this protocol.

Chapter 3

Trustful Population Protocols

This chapter is about ***Trustful Population Protocols***. This time, each agent has an opinion, encoded in the internal state. When two agents meet, if they agree on a same opinion, they cannot change it. We consider two variants: strongly and weakly trustful population protocols.

We provide an exact characterization of what can be computed for both variants, helped by a more restrictive version where opinion and output are the same: We compute exactly boolean combination of 0-threshold predicates. We then give some considerations when we provide Turing Machine tapes to agents, proving for example that proportions can then be computed.

This chapter corresponds to the work published in [BLR13] in DISC 13. These works was joint with Olivier Bournez and Jonas Lefèvre.

3.1 Model

In this chapter, agents carry opinions and some idea of trust, and hence we introduce the Trustful Population Protocols model: We add two restrictions on how the interactions can happen. When two agents have the same opinion we basically assume they cannot change their opinion. We call this *trustful interactions.* Actually, we consider two variants. In the *strong* version, two agents do not change their state if they meet an agent with a similar opinion. The *weak* version just gives the constraint that the new states remain in the same subset of opinion and we add a subset for agents not having yet any opinion.

3.1.1 Introduction

As far as we know, in all already considered restrictions of population protocols, agents are not restricted to be *trustful*: agents with similar opinion that meet can still change their opinion. In many contexts, and in particular in models coming from social networks [EK10] or natural algorithms [Cha09], it makes sense to assume that people agreeing still agree after meeting. Notice that this is often a very basic assumption of models in all these contexts:

see for example all the models in [CFL09]. This current paper is born from an attempt to understand the impact of such a restriction on the population protocol model.

We basically prove that computable predicates correspond to frequency based computable predicates. Considering that the interactions are frequency based and that the macroscopic dynamics of populations are given by laws of evolution on frequencies of agents (in possible microscopic states) in the system, it is something very natural. This is also a basic assumption in all models of nature, physics, and biology (see, e.g., [Mur02]). That is also true for all classical models from evolutionary game theory [Wei95].

Frequency based results have been obtained in another model in [HOT11]. The model [HOT11] permits even to compute the exact frequencies of each inputs. This cannot be stored by finite agents in the model based on the trustful population protocols introduced in this chapter. We will however provide a quick parallel with the [CMN$^+$11] model in the case where agents carry Turing Machines. This extra space of computation could permit to Trustful Population Protocols to compute the exact proportions of each input.

In this chapter, we characterize predicates computable by trustful population protocols in both variants. We basically prove that both variants compute exactly the same predicates that is to say boolean combination of threshold predicates with null constant.

To obtain this result, we first consider the case where the possible outputs and the opinions are the same. We did not provide an exact characterization of what this precise restriction computes. The upper bound of the computational power of this model is however enough to get our main result for the generic case.

We then consider the case of agents that can be non-finite state: agents can be arbitrary Turing machines. We provide a conjecture of characterization: The computable functions should be those frequency based that can be computed by the Passively Mobile Protocols [CMN$^+$11] on the smallest possible input with those frequencies (i.e. if the input symbol count can be divided, the protocol should use the same space of computation that in the divided case).

3.1.2 Two Restrictions

Trustful Population Protocols are obtained as a restriction over the rules of the known model of Population Protocols.

The difference here with the classic model is the addition of opinion set I. The set of states Q becomes partitioned into $|I|$ parts, one for each opinion. δ will have two different restrictions, depending on if we consider the weak of strong version of our model.

More formally:

Definition 3.1 *A* **Trustful Population Protocol** *is given by seven elements* $(I, Q, \Sigma, \iota, Y, \omega, \delta)$ *where:*

- I *is the finite set of opinions.*

- *Q is a finite set of states partitioned in $|I|$ subsets: that is to say, $Q = \bigcup_{i \in I} Q_i$ with $Q_i \cap Q_j = \emptyset$ if $i \neq j$.*
- *Σ is the finite set of entry symbols.*
- *ι is a function $\Sigma \to Q$.*
- *Y is the finite set of possible outputs.*
- *ω is a function $I \to Y$.*
- *δ is a function $Q^2 \to Q^2$.*

Remark 3.2 *Notice that unlike most of our protocols where $Y = \{True, False\}$, here, we give an importance to have $|Y| \geq 2$, as different opinions may lead to different outputs.*

This will be important with the averaging problem: see Proposition 3.10.

We add now two restrictions on how the interactions can happen when two agents have the same opinion to get the notion of trustful interactions. In the *strong* version, two agents do not change their state if they meet an agent with a similar opinion. The *weak* version just gives the constraint that the new states remain in the same subset of opinion and we add a subset for agents not having yet any opinion. We will prove later that these two versions compute exactly the same functions.

Definition 3.3 *A* **Strongly Trustful Population Protocol** *is such that δ does not modify two agents with the same opinion:*

$\forall i \in I$, $\forall q_1, q_2 \in Q_i$, $\delta(q_1, q_2) = (q_1, q_2)$.

A **Weakly Trustful Population Protocol** *is such that δ remains stable under two agents with the same opinion, if they have one (i.e. we add a no opinion set $Q_?$, corresponding to states from which agents can change their state regardless of the state met):*

$\forall i \in I \setminus \{?\}$, $\forall q_1, q_2 \in Q_i$, $\delta(q_1, q_2) = (q_1', q_2') \Rightarrow q_1', q_2' \in Q_i$.

As explained, this time Y might have more than two elements. We introduce here the notion of a computed set by trustful population protocols, in particular the computation of partitions.

Definition 3.4 *Let f be the function computed by a trustful protocol with output set Y. This protocol* **Computes** *a subset X of $\mathbb{N}^{|\Sigma|}$ if there exists some $y \in Y$ such that $X = f^{-1}(y)$.*

A subset X of $\mathbb{N}^{|\Sigma|}$ can be **Computed** *if there exists a protocol computing it.*

A **Partition** *of $\mathbb{N}^{|\Sigma|}$* **Can be Computed** *if there exists a trustful population protocol that computes simultaneously all of its sets: More precisely, to a given strict repartition on the space into subsets having no intersections is associated a protocol such that all subsets correspond to some output by f. Most of the time, to each subset will correspond an opinion.*

Example 3.5 *We provide a Trustful Population Protocol that computes the partition over* $\{A, B, C\}^+$ *according to the four following predicates:*

- $\mathcal{P}_1 = [x_A = 0]$
- $\mathcal{P}_2 = [x_A > 0 \wedge x_B > 0 \wedge x_C = 0]$
- $\mathcal{P}_3 = [x_A > 0 \wedge x_B = 0 \wedge x_C > 0]$
- $\mathcal{P}_4 = [x_A > 0 \wedge ((x_B > 0 \wedge x_C > 0) \vee (x_B = 0 \wedge x_C = 0))]$.

It is quite instinctive: it means that each partitioned subset corresponds to an output symbol. Here, to $\mathcal{P}_i$ *we associate the output* i.

A protocol computing this partition is:

- $I = \{1, 2, 3, 4\}$
- $Q = Q_1 \bigcup Q_2 \bigcup Q_3 \bigcup Q_4$, *with* $Q_1 = \{B, C, B+C\}$, $Q_2 = \{A+B\}$, $Q_3 = \{A+C\}$ *and* $Q_4 = \{A, A+B+C\}$.
- $\Sigma = \{A, B, C\}$.
- $\forall s \in \Sigma,\ \iota(s) = s$
- $Y = I$
- $\forall i \in I,\ \omega(i) = i$.
- δ *is such as, when two agents meet, both remember what types of input both have heard about. Here are some examples of rules:*

$$\begin{array}{rlcll}
A & B & \to & A+B & A+B \\
A+B & B & \to & A+B & A+B \\
A & A+B & \to & A+B & A+B \\
A & C & \to & A+C & A+C \\
B & C & \to & B+C & B+C \\
A & B+C & \to & A+B+C & A+B+C \\
A+B & B+C & \to & A+B+C & A+B+C
\end{array}$$

An agent in state q *is sure that all input symbols appearing in* q *are in the configuration. At the beginning, each agent knows only the presence of their own input. When two agents meet, they merge their knowledge.*

By fairness, at some point, each agents will know exactly what were the input symbols present at the beginning.

This protocol is both weakly and strongly trustful. We can see that this protocol is in the case where $Y = I$. *There is no way to describe* $\mathcal{P}_4$ *by using only disjunctions of inequalities.*

3.1.3 Two Trustful Protocols

We describe now two protocols that are both strongly and weakly trustful. They will be used later on in our proofs.

The first example is the 0-Threshold problem. It is important as the main result will be that the model computes exactly boolean combination of 0-threshold predicates.

Definition 3.6 *A* **0-Threshold** *predicate corresponds to the case of a threshold predicate where the second element of the inequality is equal to 0. That is to say:*
$[\sum a_i x_i \geq 0]$*, with* $a_1, \ldots, a_n \in \mathbb{Z}^n$

Proposition 3.7 *Any 0-threshold predicate can be computed by both versions of trustful population protocols.*

Proof: The principle of the protocol is slightly different to the threshold protocol in Theorem 1.16. The population tries to reach a configuration where all agents are either storing 0, either storing a non nul integer, with all of these integers having the same sign. This time, we do not use a leader, which simplifies a lot the rules.

More formally, the protocol is:

- $I = \{+, -\}$
- $Q = Q_+ \bigcup Q_-$, with $Q_- = [a, 0[\cup \{0^-\}$ and $Q_+ = [0, b]$.
- Σ, $\iota(s_i) = a_i$, $Y = \{True, False\}$, $\omega(+) = True$ and $\omega(-) = False$.
- δ is given by the following rules:

$$\begin{array}{rcll} i\ j & \to & 0\ \ (i+j) & \text{if } i \cdot j < 0 \text{ and } i+j \geq 0 \\ i\ j & \to & 0^-\ \ (i+j) & \text{if } i \cdot j < 0 \text{ and } i+j < 0 \\ 0^-\ i & \to & 0\ \ i & i \geq 0 \\ 0\ i & \to & 0^-\ \ i & i < 0 \end{array}$$

First, it is clear that this protocol is both strongly and weakly trustful. Second, if we introduce the function $e : Q \to \mathbb{N}$ such that $e(i) = i$, $e(0^-) = 0$, we can notice that the sum of the values of e applied to the state of the agents never changes.

To prove that this protocol computes the 0-threshold predicate, it suffices to establish two facts : 1- In each fair sequence, there exists a time t such that, after it, either each agent have a state in $[a, 0] \cup \{0^-\}$ and are considered to have a negative value, either they all are in a state in $[0, b] \cup \{0^-\}$ and are considered to have a negative values (we consider 0 and 0^- to be both a positive and a negative integer). 2- After that point, the sequence reaches a configuration with the right interpretation which is also stable.

Fact 1-: Here, the decreasing function is the sum of the absolute values of the integers kept in each agent. If there exists 2 agents with a different sign, it still can decrease. When every agent has the same sign, it cannot decrease anymore and it will not be possible to see any agent with a strictly different sign. At that point, the time t is reached.

Fact 2-: If now there exists an agent with a value strictly positive, all the 0^- will be turned into 0 thanks to the fairness. A configuration with only 0 and strictly positive integers is stable and has the right interpretation.

If there exists an agent with a value strictly negative, let consider a configuration that appears infinitely often in the sequence. Having the negative agent interacting with all the 0 will turn the 0 into 0^-. So, by fairness the sequence will reach a configuration with only agents in state strictly negative or 0^-. This configuration is stable and has the right interpretation.

If there are only agents in state 0 or 0^-, there is at least two agents in state 0 because the last rule that has removed the last non zero agents is the first one. From this configuration, the number of 0 can only increase, and can reach a configuration with only 0. By fairness, we can be sure that the sequence will reach this configuration which is stable and has the right interpretation. □

Remark 3.8 *The inequalities of the 0-threshold predicates can actually be large or strict. To obtain the strictness, it suffices to replace each a_i by $-a_i$.*

For the first time in this document, we will introduce a population protocol that computes a function (see Definition 1.6), the average of the inputs:

Definition 3.9 *The* **Averaging Problem** *is the following problem (see e.g. [HOT11]): We have a function $g : \Sigma \to [a, b]$, with $a, b \in \mathbb{N}^2$, $Y = [a, b] \bigcup_{i\in[a,b[} \{]i, i + 1[\}$ and we want to compute the function f such that:*

$$f(I) = \begin{cases} m & \text{if } m \in \mathbb{N} \\ ]\lfloor m\rfloor, \lfloor m\rfloor + 1[& \text{otherwise.} \end{cases}, \text{ with } m = \frac{\sum_{s\in\Sigma} g(s)\times n_s}{\sum_{s\in\Sigma} n_s}.$$

Proposition 3.10 *The Averaging problem can be computed by both versions of trustful population protocols.*

Proof: The idea of the protocol is that at some point, each agent reaches the average of the population if it is an integer. If it is not, then the sum of the inputs is maintained, and each agent are either in state $\lfloor s\rfloor^+$, either in state $(\lfloor s\rfloor + 1)^-$.

More formally, the protocol is:

- $Q = \bigcup_{i\in[a,b]} \{i, i^+, i^-\}$, with $I = [a, b] \bigcup_{i\in[a,b[} \{]i, i + 1[\}$ and $\forall i \in I$, $Q_i = \omega^{-1}(i)$.
- Σ, $\iota = g$, $Y = [a, b] \bigcup_{i\in[a,b[} \{]i, i + 1[\}$, and $\omega(i^+) = \omega((i + 1)^-) =]i, i + 1[$, $\omega(i) = i$.
- δ is given by the following rules:

$$\begin{array}{llll} i^* & j^* \to \frac{i+j}{2} & \frac{i+j}{2} & \text{if } i + j \equiv 0[2] \text{ and } i \neq j \\ i & i^* \to i & i & \\ i^* & j^* \to k^+ & (k+1)^- & \text{if } i + j \equiv 1[2] \text{ and } i < j, \text{ with } k = \frac{i+j-1}{2} \end{array}$$

With $i^* \in \{i, i^+, i^-\}$ and $j^* \in \{j, j^+, j^-\}$.

First, it is clear that this protocol is both strongly and weakly trustful. Second, if we introduce the function $e : Q \to \mathbb{N}$ such that $e(i^+) = e(i^-) = e(i) = i$, we can notice that the sum of the values of e applied to the state of the agents never changes.

To prove that this protocol computes the averaging problem, it suffices to establish two facts: 1- In each fair sequence, there exists a time t such that, after it, every agent keeps the same integer in its state. 2- After that point, the sequence reaches a configuration with the good interpretation which is also stable.

Fact 1-: First, let's prove that we can reach a configuration where all agents have an integer with a difference of at most one with each other agents. For this, we introduce a notion of energy: $E(i) = \sum\limits_{a_1, a_2 \in C_i} \left\lfloor \frac{|e(a_1) - e(a_2)|}{2} \right\rfloor$. We can prove that this function is decreasing, positive and can reach 0.

We first notice, that E changes only in the first and third rule. In these two scenarios, with a_1 and a_2 being the interacting agents, a'_1 and a'_2 their new states and a_3 an exterior one, we can prove that $|e(a_1) - e(a_2)| > 0 = |e(a'_1) - e(a'_2)|$ and $|e(a_1) - e(a_3)| + |e(a_2) - e(a_3)| \geq |e(a'_1) - e(a_3)| + |e(a'_2) - e(a_3)|$ (it is true whatever the order between a_1, a_2 and a_3).

As long as E is strictly positive, there exists two agents with a difference greater than 2. By fairness, we then can assure that they will be an interaction that will decrease E in the sequence. With this, we can ensure that there exists a time t from which, for every $t' > t$, $E(t) = 0$.

When $E(t) = 0$, the only non trivial interactions are the second and the third one. And in these interactions, agents keep their value with e. With that, we know that after the time t, each agent keeps the same integer in their state.

Fact 2-: We have two possible scenarios here: all agents have the same integer i, or some have the integer i and the others have $i + 1$.

- If all agents store the same integer i, the possible states present in the configuration are i, i^+ and i^-. There is at least two agents in the state i, because the number of agents in this state cannot decrease after the time t and the last rule applied that modified the integers in agent's memory was the first one. It has put two agents in state i in the configuration. The remaining non trivial rules can only increase the number of i, so, by fairness, the sequence will reach a configuration with only i. This configuration is stable and has the right interpretation i.

- If there exist two agents, one storing i and the other storing $i+1$ in the configuration, let consider a configuration C appearing infinitely often in the sequence. In configuration C, we can choose an agent a such that $e(a) = i$. We have this agent interact with each agent $c \in \{(i+1), (i+1)^+\}$. Symmetrically, we make an agent b such that $e(b) = i+1$ interact with all agents $c \in \{i, i^-\}$. We finally get a configuration with only i^+ and $(i+1)^-$. This configuration is stable and has the right interpretation $]i, i+1[$.

This permits to conclude that our protocol computes the averaging problem. $\square$

3.2 Computability Result

3.2.1 Partition of the Space

Here come now two propositions that are similar to the boolean combination of Theorem 1.12.

Proposition 3.11 *The two versions of trustful population protocols is stable under boolean combination: if A and B are subset of $\mathbb{N}^{|\Sigma|}$ that can be computed by trustful population protocol, then $A \cup B$ and $A \cap B$ can be computed.*

Proof: Let $(I_A, Q_A, \Sigma, \iota_A, Y_A, \omega_A, \delta_A)$ (resp. $(I_B, Q_B, \Sigma, \iota_B, Y_B, \omega_B, \delta_B)$) a protocol computing A (resp. B), and let f_a (resp. f_b) the corresponding function of the protocol. Let consider a population protocol given by:

- $Q = Q_A \times Q_B = \bigcup\limits_{(i_a,i_b)\in I_A\times I_B} Q_{i_a} \times Q_{i_b}$.
- Σ, $\iota(s) = (\iota_A(s), \iota_B(s))$, $Y = \{True, False\}$.
- $\delta((q_{A,1}, q_{B,1}), (q_{A,2}, q_{B,2})) = (q'_{A,1}, q'_{B,1}), (q'_{A,2}, q'_{B,2})$ with $\delta_A(q_{A,1}, q_{A,2}) = (q'_{A,1}, q'_{A,2})$ and $\delta_B(q_{B,1}, q_{B,2}) = (q'_{B,1}, q'_{B,2})$.

Whatever ω is, this protocol is trustful (strongly if the initial ones are both Strongly Trustful, Weakly otherwise). Let y_A and y_B such that $f_A^{-1}(y_A) = A$ and $f_B^{-1}(y_B) = B$.

Case where $A \cup B$**:** Let ω such that, for any $i_a, i_b \in I_A \times I_B$, for any $a, b \in Q_{i_a} \times Q_{i_b}$, $\omega(a, b) = True$ if and only if $\omega_A(i_a) = y_A$ or $\omega_B(i_b) = y_B$. Our protocol computes $A \cup B$.

Case where $A \cap B$**:** Let ω such that, for any $i_a, i_b \in I_A \times I_B$, for any $a, b \in Q_{i_a} \times Q_{i_b}$, $\omega(a, b) = True$ if and only if $\omega_A(i_a) = y_A$ and $\omega_B(i_b) = y_B$. Our protocol computes $A \cap B$.

$\square$

Proposition 3.12 *Let $(A_j)_{j\in J}$ be a partition of $\mathbb{N}^{|\Sigma|}$ with J finite such that for each j, A_j can be computed by some trustful population protocol. There exists a trustful population protocol that computes this partition.*

Proof: We construct the same *Product Protocol* that in Proposition 3.11, unifying the $|J|$ protocols (for each $j \in J$, $(I_j, Q_j, \Sigma, \iota_j, Y_j, \omega_j, \delta_j)$ computes A_j, with $v_j \in Y$ the corresponding value in Y to the set A_j). We also have $Y = J \cup \{\perp\}$. Here is the construction of ω:

For each $j \in J$, let $X_j = \omega_j^{-1}(v_j)$.

$\omega^{-1}(j) = X_j \times \prod\limits_{j'\neq j} (I_{j'} \setminus X_{j'})$. The remaining elements of $I = \prod_{j\in J} I_j$ have the value $\perp$ with ω. It is well partitioned this way because the $|J|$ first sets have no intersection. We have now to prove that our protocol computes each A_j.

Let $E \in \mathbb{N}^{\Sigma}$ be an input. Because the $(A_j)_{j\in J}$ is a partition of the set of inputs, there exists some $k \in J$ such that $E \in A_k$.

By fairness, at some point, for each j, there exists some $i \in I_j$ such that the projection of the configuration on Q_j remains in Q_{j_i}.

Because $E \in A_k$, any state in this configuration has its projection on Q_k in a Q_{k_i} with $i \in X_k$. As we have a partition, for each $j \neq k$, the projection on Q_j is on a Q_{j_i} with $i \notin X_j$. With that, we can conclude that the interpretation with our protocol is k.

For all $E \in \Sigma^{\mathbb{N}}$, $E \in Q_k \Rightarrow f(E) = k$. Our Protocol is Trustful and it computes each of the A_j. Hence, it computes the partition. □

We get as a consequence, the first inclusion of the characterization of our model:

Proposition 3.13 *Any finite partition of $\mathbb{N}^{|\Sigma|}$ where every class of the partition can be defined as a boolean combination of 0-threshold predicates can be computed by a Trustful Population Protocol.*

To prove the second inclusion, we will work on a more restricted version. Even if we do not have an exact characterization of what this restricted version computes, it will help to conclude the main result of this section.

3.2.2 When Agents Trust on the Output

Now we consider the special case where $Y = I$ and ω is the identity function over I.

Definition 3.14 *We say that a trustful population protocol* **Trusts the Output** *if Y and I are the same set and ω is the identity function over I.*

Our goal is to prove, in that case, that for every $y \in Y$, $f^{-1}(y)$ is a set of elements satisfying a boolean combination of inequalities of the form $\sum\limits_{s \in \Sigma} a_i \times s_i \geq 0$ and $\sum\limits_{s \in \Sigma} a_i \times s_i > 0$. In other words:

Theorem 3.15 *The partitions that can be computed by trustful population protocols that trust the output is such that each class of the partition can be defined as a boolean combination of 0-threshold predicates.*

To prove this theorem, we first need some lemmas.
In this section, we will have the following notations:

- $\mathbb{N}_{>0}$ for $\mathbb{N} \setminus \{0\}$.
- $\mathbb{Q}_{>0}$ for $\left\{\frac{p}{q} : p, q \in \mathbb{N}_{>0}\right\}$.
- $\mathbb{Q}_{\geq 0}$ for $\left\{\frac{p}{q} : p \in \mathbb{N}, q \in \mathbb{N}_{>0}\right\}$.

Lemma 3.16 *Let $(A_i)_{i \in I}$ be the partition of $\mathbb{N}^{|\Sigma|}$ corresponding to a trustful population protocol that trusts the output. Each A_i satisfies the two following properties:*

- C^+: *For each* $a, b \in A_i$, $a + b \in A_i$
- C^*: *For each* $a \in \mathbb{N}^{|\Sigma|}$, *each* $\lambda \in \mathbb{N}_{>0}$, $\lambda a \in A_i \Leftrightarrow a \in A_i$

Proof:

- By hypothesis, there exists a sequence that brings all agents from any $a \in A_i$ into states in Q_i. Symmetrically, there exists a also sequence that brings all agents from $b \in A_i$ into states in Q_i. From input $a + b$, we just need to use the same sequences to the two subsets, and then our configuration will be stable. Hence, $a + b$ must also belong to A_i.
- From the first point, we can deduce that $a \in A_i \Rightarrow \lambda a \in A_i$. For the converse implication, let j such that $a \in A_j$. As $\lambda a \in A_j$ and the $(A_i)_{i \in I}$ form a partition of the space, we deduce $i = j$.

$\square$

From now on, we assume $S \in \mathbb{N}^d$ is a given semi-linear set, i.e. it is a union of sets of the form

$$\left\{ u_i + \sum_{j \in J_i} a_j v_j : a_j \in \mathbb{N} \right\}.$$

We also suppose that S satisfies the conditions (C^+) and (C^*) introduced in the previous lemma (as we want to give a characterization of what can be computed by protocols that trust the output).

Lemma 3.17

$$S = \mathbb{N}^d \cap \bigcup_{i \in I} \left\{ a u_i + \sum_{j \in J_i} a_j v_j : a \in \mathbb{Q}_{>0}, a_j \in \mathbb{Q}_{\geq 0} \right\}$$

Proof: Let $x \in \mathbb{N}^d \cap \bigcup_{i \in I} \left\{ a u_i + \sum_{j \in J_i} a_j v_j : a \in \mathbb{Q}_{>0}, a_j \in \mathbb{Q}_{\geq 0} \right\}$.

Then there is an $i_0 \in I$, an $a \in \mathbb{Q}_{>0}$ and an $a_j \in \mathbb{Q}_{\geq 0}$ for each $j \in J_{i_0}$ such that $x = a u_{i_0} + \sum_{j \in J_{i_0}} a_j v_j$. There exist p and q in $\mathbb{N}_{>0}$ such that $a = \frac{p}{q}$ and for each a_j there exist $p_j \in \mathbb{N}$ and $q_j \in \mathbb{N}_{>0}$ such that $a_j = \frac{p_j}{q_j}$. We get:

$$x = \frac{p}{q} u_{i_0} + \Sigma_{j \in J_{i_0}} \frac{p_j}{q_j} v_j.$$

Let $y = (q.\Pi_{j \in J_{i_0}} q_j).x = (q.\Pi_{j \in J_{i_0}} q_j) a.u_{i_0} + \Sigma_{j \in J_{i_0}} \left(p_j.(q.\Pi_{j' \neq j} q'_j)\right) v_j$.

We have:

$y - \left[(q.\Pi_{j \in J_{i_0}} q_j)a - 1\right].u_{i_0} = u_{i_0} + \Sigma_{j \in J_{i_0}} \left(p_j.(q.\Pi_{j' \neq j} q'_j)\right) v_j \in \left\{ u_{i_0} + \sum_{j \in J_i} a_j v_j : a_j \in \mathbb{N} \right\}.$

If $(q.\Pi_{j\in J_{i_0}} q_j)a = 1$, then we have $y \in S$.

Otherwise, $u_{i_0} \in S \Rightarrow \left[(q.\Pi_{j\in J_{i_0}} q_j)a - 1\right].u_{i_0} \in S$ (using C^*).
Hence, as y is a sum of two elements of S, by C^+, $y \in S$.
Finally, we have from C^* that $x \in S$.

The other inclusion is easy since $\mathbb{N} \subset \mathbb{Q}_{\geq 0}$. The lemma follows. $\square$

For the rest of the proof, we will consider that $v_0 = u_i$ (i.e. if u_i was not in the initial set of vectors, we add it directly).

To pursue the proof, we need a stronger version of Caratheodory's Theorem. Its original version is the following:

Lemma 3.18 (Caratheodory's Theorem[CHKB35]) *If $x \in \mathbb{Q}^d$ is a positive linear combinations of $\{v_j\}_{j\in J_i}$ then there exists a linearly independent subset $\{v_{l_1}, \ldots, v_{l_k}\} \subset \{v_j\}_{j\in J_i}$ such that x is a positive linear combination of $v_{l_1}, \ldots v_{l_k}$.*

In our proof, we need to be sure that the vector $v_0 = u_i$ stays after the simplification of the vectors. This is why we prove a the following version of Lemma 3.18.

Lemma 3.19 *If $x \in \mathbb{Q}^d$ is a positive linear combinations of $\{v_j\}_{j\in J_i}$, with $0 \in J_i$, $v_0 \neq 0$ and $\forall j \in J_i$ $v_j \in \mathbb{N}^d$, then there exists a linearly independent subset $\{v_{l_1}, \ldots, v_{l_k}\} \subset \{v_j\}_{j\in J_i}$ such that x is a positive linear combination of $v_0, v_{l_1}, \ldots v_{l_k}$.*

Proof: Let x be a positive linear combination of k linearly independent vectors $\{v_{l_1}, \ldots, v_{l_k}\}$. We have $x = \sum a_m v_{l_m}$.

We need to separate three cases here: (i) v_0 is already in the subset, (ii) the new subset is still independent if we add v_0 and (iii) v_0 can be generated by the subset:

- $v_0 = v_{l_m}$ for some m: This case is simple, as applying a permutation on the numeration of the vectors to put v_{l_m} in first position is enough.

- v_0 **is independent** from $\{v_{l_1}, \ldots, v_{l_k}\}$: This case is easy. Adding v_0 to the set is allowed, as the linear independence is kept.

- v_0 **is not independent**: We have $v_0 = \sum b_m v_{l_m}$. Note that the b_m are not obligatory positives. Let M be an integer such that b_M is strictly positive and such that $\frac{a_M}{b_M}$ is minimal (among the m such that $b_m > 0$ and $a_m > 0$). M exists because if all b_m are negative or null, v_0's coordinates would all be negative or null. From $v_0 = \sum b_m v_{l_m}$ we get $v_M = \frac{1}{b_M} v_0 - \sum\limits_{m\neq M} \frac{b_m}{b_M} v_{l_m}$. We will now insert this v_M in x:

$$\begin{aligned} x = \sum a_m v_{l_m} &= \sum_{m\neq M} a_m v_{l_m} + a_M \left(\frac{1}{b_M} v_0 - \sum_{m\neq M} \frac{b_m}{b_M} v_{l_m} \right) \\ &= \frac{a_M}{b_M} v_0 + \sum_{m\neq M} \left(a_m - a_M \frac{b_m}{b_M} \right) v_{l_m} \end{aligned}$$

  We need now to prove that all the coefficients are positive. As M is chosen such that b_M is positive, $\frac{a_M}{b_M}$ is positive. Let prove that for each $m \neq M$, $a_m - a_M \frac{b_m}{b_M} \geq 0$.

– If $b_m \leq 0$, then $-a_M \frac{b_m}{b_M} \geq 0$, we deduce $a_m - a_M \frac{b_m}{b_M} \geq 0$.
– If $b_m > 0$, then, as M was chosen to minimize $\frac{a_M}{b_M}$,
$\frac{a_m}{b_m} \geq \frac{a_M}{b_M} \Rightarrow a_m - a_M \frac{b_m}{b_M} \geq 0$.

This concludes the proof that v_0 can be put in the set of the linearly independent vectors. We now suppose that we always chose $v_{l_1} = v_0$ if $v_0 \neq 0$. This lemma is proved. $\square$

With this lemma, we have :

$$\left\{ \sum_{j \in J} a_j v_j : a_j \in \mathbb{Q}_{\geq 0} \right\} = \bigcup_{\{v_{l_1}=v_0, v_{l_2}, \ldots, v_{l_k}\} \text{ l.i.}} \left\{ \sum_{j \in \{l_1, \ldots, l_k\}} a'_j v_j : a'_j \in \mathbb{Q}_{\geq 0} \right\}$$

(where *l.i.* means linearly independent).

Hence we can rewrite

$$S = \mathbb{N}^d \cap \bigcup_{i \in I |} \bigcup_{\{v_{l_1}=v_0, v_{l_2}, \ldots, v_{l_k}\} \subset J_i \text{ l.i.}} \left\{ a u_i + \sum_{1 \leq m \leq k} a_m v_{l_m} : a \in \mathbb{Q}_{>0}, a_m \in \mathbb{Q}_{\geq 0} \right\}$$

Let consider now the set $V = \{v_{l_1}, \ldots, v_{l_k}\} \subset J_i$ of linearly independent vectors and the set $H_V = \left\{ a u_i + \sum\limits_{1 \leq m \leq k} a_m v_{l_m} : a \in \mathbb{Q}_{>0}, a_m \in \mathbb{Q}_{\geq 0} \right\}$.

We then have that S can be characterized by a disjunction (for each possible J_i) of disjunction (for each linearly independent set of vectors V) of sets of the form $\mathbb{N}^d \cap H_V$.

We will prove now that the belonging in a H_V can be characterized as a conjunction of inequalities and equalities with the second term being 0 (i.e. 0-threshold predicates). It corresponds to have the following lemma to characterize H_V:

Lemma 3.20 *For every $m \leq d$, there exist some $y_m \in \mathbb{N}^d$ and some $\propto_m \in \{\geq, >, =\}$ such that:*

$$x \in H_V \Leftrightarrow \forall m \leq d,\ x \cdot y_m \propto_m 0.$$

Proof: We have $k \leq d$. If $k < d$, then we can complete V with $k - d$ linearly independent vectors $w_{k+1}, \ldots w_d$ such that $B = V \cup \{w_{k+1}, \ldots w_d\}$ is a base of $\mathbb{Q}^d$, if $k = d$ then set $B = V$.

Let then swap the base to B. Let $(x_1, \ldots, x_d)$ the coordinates of an element x in the new system. We have:

- If $u_i = 0$:

$$x \in H_V \Leftrightarrow \begin{cases} x_m \geq 0 & \text{if } m \leq k \\ x_m = 0 & \text{otherwise.} \end{cases}$$

- If $u_i \neq 0$:

$$x \in H_V \Leftrightarrow \begin{cases} x_1 > 0 & \\ x_m \geq 0 & \text{if } 1 < m \leq k \\ x_m = 0 & \text{otherwise.} \end{cases}$$

Coordinates in a new base correspond to apply scalar products to a set of vectors. Hence, this change of base implies the existence of vectors $y_1, \ldots, y_d$ such that, for all m, $x_m = y_m \cdot x$. From this, we get the expected result:

- If $u_i = 0$:

$$x \in H_V \Leftrightarrow \begin{cases} y_m \cdot x \geq 0 & \text{if } m \leq k \\ y_m \cdot x = 0 & \text{otherwise.} \end{cases}$$

- If $u_i \neq 0$:

$$x \in H_V \Leftrightarrow \begin{cases} y_1 \cdot x > 0 & \\ y_m \cdot x \geq 0 & \text{if } 1 < m \leq k \\ y_m \cdot x = 0 & \text{otherwise.} \end{cases}$$

$\square$

From this study, we deduce the proposition :

Proposition 3.21 *If S is a semi-linear set satisfying (C^+) and (C^*), then it can be described by a boolean combination of formulas of the form $y \cdot x \propto 0$ with $\propto \in \{\geq, >, =\}$.*

This permits to get Theorem 3.15, as we now know that the sets of the computed partition are semilinear sets satisfying the properties (C^+) and (C^*).

Remark 3.22 *Actually, we did not provide an exact characterization of what is computed by trustful population protocols that trust the output.*

Our first thoughts was that what can be computed where exactly disjunction of 0-threshold predicates, but a counter-example was given in Example 3.5:
the set $\{(x, y, z) | x > 0 \wedge ((y > 0 \wedge z > 0) \vee (y = 0 \wedge z = 0))\}$ is computable.

We have a conjecture of what exactly can be computed, but no proof yet of its veracity.

Conjecture 3.23 *Trustful population protocols that trust the output compute exactly sets that can be described by disjunctions of the form:*

$$\bigcup_{0 \leq i < k} \left(\bigcap_{h(i) < j \leq h(i+1)} (y_j \cdot x > 0) \cap \bigcap_{j \leq h(i)} (y_j \cdot x = 0) \right)$$

Where $k \leq l \leq d$, each $y_i \in \mathbb{N}^d$, and $h : [0, k] \to [0, l]$ is a strictly increasing function with $h(0) = 0$.

The geometric interpretation is that the computed sets are convex strict cones united recursively in their frontiers by computable sets.

3.2.3 Computational Power of Trustful Population Protocols

As we know to what corresponds each $i \in I$ (in term of computability), we can now have the characterization of what can be computed by our model:

Theorem 3.24 *The partitions that can be computed by (strong and weak) trustful population protocols correspond exactly to partitions where each set can be defined as a boolean combination of 0-threshold predicates.*

Proof: The first inclusion was already proved in Proposition 4.22.

We have, for $y \in Y$, $f^{-1}(y) = \bigcup_{i \in \omega^{-1}(y)} A_i.$, where A_i is the set of inputs leading to opinion i.

With the previous result, we have $a \in f^{-1}(y) \Leftrightarrow a \in \bigcup_{i \in \omega^{-1}(y)} A_i \Leftrightarrow a$ satisfies the formula of an A_i. With this, $f^{-1}(y)$ is defined by the disjunction of boolean combinations of threshold problems (with a null constant), where the inequality can be strict. □

3.2.4 Interpretation in Terms of Frequencies

Here is another interpretation:

Proposition 3.25 *Both versions of trustful population protocols are stable under multiplication and division: For any function f computed by a trustful population protocol, for any input $E \in \mathbb{N}^{|\Sigma|}$, for any $k \in \mathbb{N}^+$, $f(E) = f(k \times E)$, and if the number of each input symbol in E is a multiple of k, $f(E) = f(E/k)$.*

Proof: As every computation needs to have ultimately the good interpretation, we can first separate the population of $k \times E$ in k populations E. There is a sequence from E that reaches a configuration where all agents have the same interpretation. We can have all of the k populations E perform the same sequence. From that configuration, as every agent is in the same set Q_i for some $i \in I$, the interpretation cannot change anymore because of the trustful constraint. Because of that, $f(E) = f(k \times E)$.

The case E/k is exactly the same: we use the previous result with E/k and $k \times E/k = E$. □

Remark 3.26 *With this result, there are some functions computable by population protocols that can be shown very easily not to be computable by our models:*

- $[x_A \geq 2]$*: with $E = (A, A, B, B)$, we have $f(E) = True \neq False = f(E/2)$.*
- $[x_A \equiv 0[2]]$*: With the same input E as above, $f(E) = True \neq False = f(E/2)$.*

A way to interpret previous theorem, is then to observe that computable predicates are always **Frequency Based**: i.e. a predicate on frequencies of letters in the input.

Indeed, observe that a Threshold $\sum\limits_{s \in \Sigma} g(s) n_s \geq c$ with $c = 0$ can always be written $\sum\limits_{s \in \Sigma} g(s) f_s \geq 0$, where $f_s = \frac{n_s}{n}$, with $n = \sum_s n_s$. Here f_s is the frequency of agents in state s in the population.

3.3 Non Finitely States Agents

In this section, we consider that agents can carry Turing Machines, as in the Passively Mobile Protocols [CMN+11]. We give a brief definition and a result that says that frequencies can be computed. This result uses a proof quite similar to the one used in [HOT11] to find the exact proportions of each input symbol.

Definition 3.27 *A* **Trustful Population Protocol with Space** $f(n)$ *is a PM protocol such that we add a fifth tape to write the opinion:*

- *The opinion set I has a size depending of the size n of the population:* $I \subset \{?\} \cup (\{0,1\}^{f(n)} \times Y)$.
- *When two agents meet, if they have the flag 1, they copy the state of the other agent only if they do not share the same opinion (i.e. they have two different things written on their fifth tape).*

Here is the result that motivates a possible study:

Theorem 3.28 *If all agents have a space of at least $\Omega(n \log n)$, there exists a trustful population protocol that computes the exact frequency of each possible input. That means it computes for each $s \in \Sigma$ some $m_s \in \mathbb{N}$ such that input symbol s has the following proportion in the population:*

$$\frac{m_s}{\sum\limits_{s' \in \Sigma} m_{s'}}$$

Proof: The idea of this protocol is to run the averaging problem with different parameters at the same time. $I(n)$ will be all the possible proportions (frequencies) for populations of size smaller or equal to n (it can be represented on $|\Sigma| \log(n)$ bits, as it suffices to have n and $n_s \leq n$ for each s).

For each $s \in \Sigma$, each agent will run the averaging problem on $k \times n_s$ for an increasing number of k until the protocol provides an integer result. When the protocol stops on an average a_k for a given k, the population knows that the proportion n_s/n is equal to a_k/k, as $k \times n_s = a_k \times n$. As long as the agent does not contain any integer average, it will have the opinion $?_s$ for input s. As soon as it gets an integer value for some k, it will keep computing until all the average protocols for $k' \leq k$ get a coherent proportion (i.e. if p_k is the integer and $p_{k'}$ is also an integer, $k'p_k = kp_{k'}$, else the k' protocol has to give the interval $]p_{k'}, p_{k'}+1[$ such that p_k/k is in it).

For $k = n$, we can be sure that the protocols will finish. If we separate the space in $f(n)/|\Sigma|$ bits (one portion for each $s \in \Sigma$), each portion will need, to compute the corresponding proportion, $\log 1 + \log 2 + \ldots \log n \leq n \log n$ bits. $\square$

I encountered during my thesis issues to get a formal definition working for this model to compute functions. The trustful part forces the protocols to compute exactly the same output for two inputs with the same frequencies.

As the population needs to compute the same thing that does the minimal input, it has to be computable by the smallest input (i.e. if k is the maximal integer that divides the input E without providing fractions for some input, the protocol needs to use at most $f(n/k)$ space on each agents).

Proposition 3.29 *Let $F : \mathbb{N}^{|\Sigma|} \to Y$ be computable by a Trustful Population Protocol on space $f(n)$ for some f. F has to be frequency based:*

If $E(n_{s_1}, \ldots, n_{s_k}, \ldots) \in \mathbb{N}^{|\Sigma|}$ is an input and p is the larger common divisor of the n_{s_k}, then $F(E) = F(E/p)$. The protocol needs at most $f(n/p)$ space on each agent to compute its output.

Proof: The proof is really similar to the proof of Proposition 3.25. The protocol has to be stable under multiplication of the input, and by immediate result, stable under division of the input. □

3.4 Conclusion

3.4.1 Resume

We introduced two versions of trustful population protocols, a *strong* and a *weak*, that finally proved to compute same predicates. Both versions compute exactly the partitions that can be defined as a boolean combination of 0-threshold predicates.

To obtain this result, we studied the restriction where opinion and output are the same. This more restrictive version has not been yet characterized.

We also proved that the model is frequency based, which means that a protocol accepts an input if and only it accepts this input multiplied by any positive integer. In particular, if we combine the trustful restriction with the Passively Mobile Protocols [CMN$^+$11], it is possible to compute the exact proportions of each input.

3.4.2 Conjecture

The introduction of the restriction where agents trust the output leaves the question about what is the computational power of this version. Conjecture 3.23 raises what we think the computable partitions are. No proof has been found yet to confirm it.

Part III

Weakening of the Community Protocols and Some Other Extensions

Chapter 4

Fast Computing with Community Protocols

This chapter introduces the class ***CPPOLYLOG***, corresponding to what can be computed with Community Protocols in a polylogarithmic number of expected parallel interactions, or a polylogarithmic number of broadcasts.

We provide two easy protocols: one computing the sorted language and the other computing semilinear sets. We provide then a protocol computing the size of the population (by encoding it in binary on agents). We prove that the model is quite weak with two impossibility results including the fact that the rational language $(ab)^*$ cannot be computed. We finish with some comparisons with other computational classes. We introduce a class of Turing Machine to try to match what can be computed, summarizing the power of computation we found.

4.1 Model

4.1.1 Introduction

Angluin *et al.* [AAE06] proved that any computable predicate by a Population Protocol can be computed in $O(n \log^5 n)$ expected interactions, as long as there is a unique leader at the beginning. Angluin *et al.*, in [AAE06], also provide arguments leading to the idea that their exists protocols computing a leader election in $O(n \log n)$ expected interactions.

The exact characterization of what can be computed by populations having unique leaders gave the motivation to look to what can be computed in $O(n \log^k n)$ expected interactions (for any $k > 0$), with the Community Protocols model [GR09]. We can notice that $O(n \log^k n)$ expected interactions corresponds to a polylogarithmic expected number of *parallel* interactions. We will now consider up to the end of this chapter that each pair of agents (or identifiers) have the same probability to be chosen at each step of a computation.

For sake of simplicity, we will often talk about epidemics and expected number of interactions using the results provided in Section 1.3.

Let introduce the class we will work with in this chapter:

Definition 4.1 *We define the class* **CPPOLYLOG** *as the set of languages that can be recognized by a Community Protocol with $O(n \log^k n)$ expected interactions for some $k \in \mathbb{N}$.*

We start by introducing some computational results. In particular, we provide a fast protocol to compute with high probability the size of the input (writing it in binary).

We will then provide two impossibility results:

- There is no time for all agents to know what is the Next and the Previous identifier to their own.
- The language $(ab)^*$ cannot be computed.

We finish by considering comparisons of the class $CPPOLYLOG$ with a few other classes. We did not find any exact characterization. We provide in this chapter the tightest characterization we got.

Definition 4.2 *In this chapter, we will often talk about* **Next** *and* **Previous**. *Those are two functions $U \to U$ that provides, to a given identifier, the next one/previous one present in the population. More formally:*

- $Next(id_a) = \min\{id_b : id_b > id_a\}$.
- $Previous(id_a) = \max\{id_b : id_b < id_a\}$.

By convention, $Next$ of the highest identifier is the smallest, and $Previous$ of the smallest identifier is the highest one. Thus, these two functions are bijective.

Sometimes, $Next$ and $Previous$ will be slots in protocols, with the purpose to find the right identifier corresponding to the function. "Finding its $Next$" means that the agent needs to put the right identifier in its slot $Next$.

4.1.2 Two Easy Protocols

We start by describing some protocols. After two easy ones, we fill focus on a way to globally compute the size of the input and to mark any polylogarithmic number of agents.

Proposition 4.3 *The Sorted Language (see Definition 1.38) can be computed by a Community Protocol with $O(n \log n)$ expected interactions.*

This language is in $CPPOLYLOG$.

Proof: The idea of the protocol is quite similar to the Election Leader described in Example 1.37:

Each agent stores $2k$ identifiers, two for each input $s_i \in \Sigma$. The first corresponds to the smallest agent known (by agents already met) as starting with input s_i, the second to the highest agent starting with this input.

When two agents meet, they update their identifiers. If at some point, an agent sees a contradiction (i.e. some i and j such as $i < j$ and highest agent starting with s_i higher than the smallest starting with s_j), it broadcasts a $False$ output.

For each s_i, the smallest (and highest) identifier starting with it will be spread all over the population in a single epidemic starting from the agent itself. Hence, in a single epidemic, all agents will know the two needed identifiers for each s_i. Agents will then be able to detect directly a problem.

Thus, this protocol will use an epidemic, which means $O(n \log n)$ expected interactions. □

Proposition 4.4 *Any Semilinear set is in $CPPOLYLOG$.*

Proof: As the Leader Election can be performed in $O(n \log n)$ (see Example 1.37), and as any semilinear set can be computed in $O(n \log^5 n)$ expected interactions (see Theorem 1.33), the result is a direct corollary. □

Remark 4.5 *To ensure the expected number of interactions in this chapter, we will use the same technique that in [AAE06]: We will run in parallel a fair protocol that computes the same function. This will handle the case where the probabilistic protocol fails to provide the right output.*

In the following sections, we will never argue about the speed of the fair protocol that we will run in parallel to ensure that any computation will leed to the right output. The protocol we will run in parallel will be as follows:

- *The protocol elects the agent with the smallest identifier as a leader.*

  It takes the time of an epidemic taking $O(n \log n)$ interactions in expectation.

- *Each agent will look for the smallest identifier bigger than its own (called $Next$) and the biggest identifier smallest than its own (called $Previous$).*

- *Each time an agent updates its $Next$ of $Previous$, it looks for the leader to start back the computation.*

  It takes at most $O(n^2 \log n)$ expected interactions, as in $O(n^2 \log n)$ interactions, each pair of agents has an high probability to have interacted.

- *The leader performs the epidemics by itself. It looks its own $Next$, then the $Next$ of its $Next$... until it found an agent without any $Next$. The leader will be sure to have spread the information to all agents at this point if every agents have found there $Next$.*

  Each of these "new" epidemics takes $O(n^3 \log n)$ expected interactions, since it needs n specifics interactions.

This protocol multiplies by n^2 the expected number of interactions needed. Hence, since it will be needed with a probability $1 - n^{-c}$ for any c, the expected number of interactions needed by the probabilistic protocol will correspond to the one announced in each of our results.

4.2 The Size of the Population

The purpose of the two following sections is to find a way to compute the size of the population. As each agent can only contain a finite state, each agent will store one bit, and the $\log n$ first agents (according to their identifiers) will ultimately have the size written in binary when you put their bits according to their order.

This way to compute the size will be explored again in the Homonym Population Protocols model in Chapter 5.

4.2.1 The Median Identifier

To perform such an algorithm, we first need a protocol that finds quickly the median identifier. Then, we will repeat it on the first half of the population, and go on until we have exactly one agent.

Theorem 4.6 *Finding the median identifier is in $CPPOLYLOG$. The median identifier is the identifier Med such that $|\{id : id \leq Med\}| - |\{id : id > Med\}| \in \{0, 1\}$.*

We will described a protocol based on dichotomy. For this, we construct the following recursive protocol that has as input two identifiers Min and Max and tries to find med knowing that it is in $]Min, Max[$.

We do not provide a full description of the protocol but all the ideas to build it. We first describe the d slots of identifiers. Each agent will also store some bits also described bellow. We then describe the algorithm of the protocol. We finally provide a lemma that proves that the protocol is in $CPPOLYLOG$.

Each agent stores four slots of identifiers and three bits:

- Slot $Leader$ containing the leader's identifier.
- Slots Min and Max (that will be initialized in steps 0.1 and 0.2, before launching the first time the dichotomy protocol).
- Slot $Cand$ that contains the current candidate to be the median identifier.
- Bit C equals to 1 if and only if the agent's identifier is in $]Min, Max[$.
- Bit G that equals 1 if and only if the agent's identifier is strictly greater than $Cand$.
- Bit S that equals 1 if and only if the agent's identifier is smaller or equal to $Cand$.

To an agent with identifier id, its slot S will be written $Slot_{id}$ and its bit B will be written B_{id}.

We will not describe formally all the rules of the protocol but the essential steps. Here are of the steps of the protocol (0.1, 0.2 and 0.3 are the initializing steps):

Step 0.1 We start with a Leader Election using the smallest identifier as a Leader. Each agent stores the leader's identifier in its slot $Leader$ and in its slot Min.

Step 0.2 At the same time, the Leader performs an epidemic, each agent updating its slot Max each time they see a bigger candidate.

We know that when the epidemic stops, each agent, with high probability, knows the smallest and the biggest identifier present in the population.

Step 0.3 Every agents that are not Min and Max set their slot $Cand$ to their own identifier and put their bit C to 1.

Min and Max put _ in slot $Cand$ and setbit C to 0.

Step 1. The Leader looks for a candidate to be the medium identifier Med. For this, it propagates an epidemic where each agent such that $Cand = \_$ fills it as soon as it meets an agent having an identifier with this slot filled (putting it in its own slot).

Step 2. When the leader finds a candidate, it starts a new epidemic to spread the candidate's identifier to each agent. When an agent gets the candidate, it updates its slots $Cand$, G and S (according to its relative position to the identifier).

Step 3. The leader launches two protocols described in [AAE06]. The first one decides if $\sum_{id}(S_{id} - G_{id}) \in \{0, 1\}$ and the second one decides if $\sum_{id}(S_{id} - G_{id}) \geq 2$.

Section 1.3.4 recalled that a subtraction can be performed in $O(\log^3 n)$ expected epidemics.

Note that $\sum_{id} S_{id}$ corresponds to the number of agents having an identifier smaller or equal to slot $Cand$, and $\sum_{id} G_{id}$ corresponds to the number of agents greater.

Step 4. According to the result, the protocol acts as follows:

- If $\sum_{id}(S_{id} - G_{id}) \in \{0, 1\}$, slot $Cand$ corresponds to identifier Med, the protocols ends.
- If $\sum_{id}(S_{id} - G_{id}) \geq 2$, we have $Min < Cand < Med < Max$. The Leader does an epidemic asking each agent to put identifier $Cand$ in their slot Min, and if the agent's identifier is smaller than $Cand$, the agent updates $Cand$ to _ and switches C to 0 .

  Then the Leader goes back to step 1.
- Else, we have $Min < Med < Cand < Max$. The Leader does an epidemic asking each agent to put $Cand$ in their Max, and if the agent's identifier is higher than $Cand$, the agent updates $Cand$ to _ and switches C to 0.

  Then the Leader goes back to step 1.

Lemma 4.7 *The protocol described above finishes in $O(n \log^3 n)$ expected interactions.*

Proof: This protocol finishes, as in each step, there is at least one less candidate (formally, $\sum_{id} C_{id}$ is strictly decreasing after each passage in the loop). Each loop uses a polylogarithmic number of epidemics (The decision problem defined in [AAE06] uses $O(\log^2 n)$ expected interactions).

Hence, we only need to prove that we have a polylogarithmic number of loops in expectation.

Let c_k be the number of agents such as $Cand_{id} = 1$ after the kth loop. We have $c_0 = n$ and $\forall k$, $c_k > c_{k+1}$. We will show now that if $c_k > 1$, $\Pr\left(c_{k+1} \leq \frac{3}{4}c_k\right) \geq \frac{1}{4}$.

We notice first that each agent having its bit C set to 1 has the same probability to be chosen by the Leader, as each of these agents act the same way. We will call $Cand$ the identifier of the selected agent.

We assume that $|\{id : id \leq Med \;\&\; C_{id} = 1\}| \geq |\{id : id \geq Med \;\&\; C_{id} = 1\}|$ (the case $\leq$ is symmetric). We have $\Pr(Cand \leq Med) \geq 1/2$, as we choosed to focus on the case where the majority of candidates have a smaller identifier than Med.

We now suppose that $Cand \leq Med$. Let M be the median identifier of the subset $\{id : id \leq Med \;\&\; C_a = 1\}$. We have $\Pr(Cand \geq M | Cand \leq Med) \geq 1/2$, as M is the median identifier among those smaller than Med.

In the case $Cand \geq M$, we have the following inequality on the number candidates that will no longer be one:
$|\{id : id \leq M \;\&\; C_{id} = 1\}| \geq \frac{1}{2}|\{id : id \leq Med \& C_{id} = 1\}| \geq \frac{1}{2} \times \frac{1}{2}|\{id : C_{id} = 1\}| = \frac{1}{4}c_k$.

Hence, if we have $Cand \geq M$, we have $c_{k+1} = c_k - |\{id : id \leq Cand \;\&\; C_{id} = 1\}| \leq \frac{3}{4}c_k$.

The expectation of the number of trials before we are in this case is less than 4 (as we have more than half a chance to be inferior to Med and then have again more than half a chance to be greater than M).

$(3/4)^i c_0 \leq 1 \Leftrightarrow i \ln(3/4) + \ln n \leq 0 \Leftrightarrow i \geq \frac{\ln n}{\ln 4/3}$.

This protocol uses, in expectation, at most $\frac{4}{\ln 4/3} \ln n$ loops before finding the Med identifier.

This protocol uses $O(n \log^3 n)$ expected interactions to compute the median identifier.

□

4.2.2 The Size of the Population

The protocol of the previous section will be used as a tool to write the size of the population on the $\log n$ first agents.

Theorem 4.8 *There exists a protocol that writes in binary on the first* $\log n$ *agents the size of the population, taking an expected number of* $O(n \log^4 n)$ *interactions.*

Proof: To build this protocol, we first adapt the previous one as follows:

- The Median protocol can be used on a segment:
  Instead on working on the whole population, we accept to launch it with two identifiers A and B. We will look on the median identifiers among those who are in $[A, B]$. To adapt the original protocol, it suffices to definitively fix the bits S_{id} and G_{id} of Theorem 4.6 to 0 for the agents with $id \notin [A, B]$, as they no longer are in the interval considered.

- The protocol needs to check if the number of agents in the segment $[A, B]$ is even or odd. This corresponds to check if $|\{A \leq id \leq Med\}| - |\{B \geq id > Med\}|$ is equal to 0 or 1.

Each agent stores a bit $Size$ set to 0. The bits of the size are computed from the right to the left as follows:

1. Let min (resp. max) be the smallest (resp. higher) identifier present in the population. We initialize A and B with, respectively, min and max.

   We also initialize an identifier C to min, it will represent the cursor to know on which agent we write the bit of the size of the population when it is computed.

2. We compute Med, the median agent in $[A, B]$, and write the parity on the bit $Size$ of agent C.

3. We update the identifiers as follows: $B \leftarrow Med$, $C \leftarrow Next(C)$.

4. If $A \neq B$, come back to step 2, else end.

When this protocol is over, we have

$$n = \sum_{i=0}^{\log n} 2^i Size_{Next^i(Min)}.$$

where $Size_{Next^i(Min)}$ is the bit $Size$ of the $(i + 1)$th agent.

The Median protocol will be iterated exactly $\log n$ times, hence we get the expected number of interactions being $O(n \log^4 n)$. □

Corollary 4.9 *The number of each input symbol can be written in binary on the* $\log n$ *first agents with an expected number of* $O(n \log^4 n)$ *interactions.*

Proof: It suffices to iterate the protocol of the size of the population on each subset of agents based on their input. □

4.2.3 Marking $log^k n$ agents

Theorem 4.10 *In $O(n(\log^4 n + \log^{k+1} n))$ expected interactions, it is possible to mark exactly $\log^k n$ agents (if $n \geq \log^k n$).*

More formally, if each agent has a bit B, there is a protocol such as at the end, exactly $\log^k n$ agents have their bit B equal to 1.

Proof: Each agent has its bit B initially set to 0.

To mark exactly $\log n$ agents, it suffices to run the counting protocol, and mark each agent whose identifier appeared in slot C (slot C of the proof of Theorem 4.8).

To mark exactly $\log^k n$ agents, the leader will use a k-tuple of identifiers. It is initialized by k times its own identifiers. Let L be the $\log n$th identifier (corresponding to the identifier in slot C at the end of the counting protocol).

The leader also stores an identifier M corresponding to the identifier of the next to mark. It initializes it with its own identifier I_L.

The tuple will be seen as a counter. Each possible element in the slots will be the identifiers smaller or equal to L. The protocol works as follows:

1. The leader marks M.
2. The leader computes $Next(M)$, and put the identifier in the slot M.
3. The leader increments its k-tuple seen a number written in base $\{I_L, Next(I_L), \dots, L\}$. If the last element of the tuple is not L, its switches its to its $Next$. If it is L, then it switches it to I_L, then it increments the next identifier until an identifier is switched to its $Next$.
4. If an identifier has been, in the step 3, switched by its $Next$, the protocol goes back to Step 1, else the algorithm is finished.

Each loop takes $O(n \log n)$ expected interactions, since it needs two epidemics to get the two $Next$. As the counter is equivalent to a counter in base $\log n$, there will be exactly $\log^k n$ loops.

This protocol switches the $\log^k n$ first agents' B bit to 1 in $O(n(\log^4 n + \log^{k+1} n))$ expected interactions with high probability. □

4.3 Two Impossibility Results

We bring now two results that motivate my opinion on this model: the population cannot take into consideration precisely the subwords in the population. More precisely, only a polylogarithmic number of agents may know what their is exactly on its "neighbors". It is supported by the fact that only a polylogarithmic number of agents will know the identifier next of their own (Section 4.3.1).

The proof that $(ab)^* \notin CPPOLYLOG$ (Section 4.3.2) is based on the fact that there is a pair of consecutive identifiers such that, with high probability, the population will not be able to differentiate them, as these identifiers will not appear in a common interaction during the computation.

4.3.1 Arrangement Protocol

Definition 4.11 *An* **Arrangement Protocol** *is such as, at some point, each agent knows its next identifier present in the population. More formally, each agent has an identifier slot $Next$. For any fair execution, there exists some $I \in \mathbb{N}$ such as, for any $i \geq I$ and any identifier id_a, in the ith configuration, we have $Next(id_a) = \min\{id_b : id_b > id_a\}$.*

The $Next$ of the agent of maximal identifier will correspond by convention to the smallest identifier in the population.

Theorem 4.12 *An arrangement protocol takes at least $O(n\sqrt{n})$ expected interactions.*

Recall that U denotes the set of possible identifiers.

Let T be the number of interactions needed in a sequence of configurations $(C_i)_{i\in\mathbb{N}}$ to reach the point where all agents have the right $Next$. We define, for each identifier id and each configuration C_i, the following elements:

- $E_{id}(i) = \{id_b \in U : id_b \neq Previous(id)$ & id has been in a slot of id_b in a configuration C_j with $j \leq i\}$. It corresponds to the set of identifiers where id appeared in a slot, excluding $Previous(id)$.
- $M_{id}(i) = |E_{id}(i)|$.
- $E_{id} = E_{id}(T)$ and $M_{id} = M_{id}(T)$.

We first prove the following lemma:

Lemma 4.13
$$\sum_{id\in U} M_{id} \leq d(n+2T).$$

Proof: From a configuration C_i, we define, for $id \in U$, $p_{id}(i) \subset U$ the set of identifiers appearing in the d slots of id.

Let $L_{id}(i)$ be the set of all of the $p_{id}(j)$, with $j \leq i$ (we have $L_{id}(i) \in \mathscr{P}(U)$, $\mathscr{P}(U)$ being the set of subsets of U).

Let $N_{id}(i)$ be the number of times id appears in each $L_{id_b}(i)$ and $N_{id} = N_{id}(T)$. More formally:
$$N_{id}(i) = \sum_{id_b\in U} |\{p : id \in p \ \& \ p \in L_{id_b}(i)\}|$$

Finally, let $S(i) = \sum_{id\in U} \sum_{p\in L_{id}(i)} |p|$ and $S = \sum_{id\in U} \sum_{p\in L_{id}} |p|$.

We have $S(i) = \sum_{id \in U} N_{id}(i)$, as each element of each set p is counted exactly once in one of the $N_{id}(i)$.

We also have $N_{id}(i) \geq M_{id}(i)$, as, if $id_b \in E_{id}$, there is some $p \in L_{id_b}$ such as $id_b \in p$. We then also get $N_{id} \geq M_{id}$.

Let $Z(i) = \sum_{id \in U} |L_{id}(i)|$ and $Z = \sum_{id \in U} |L_{id}|$.

We know that for any p, $|p| \leq d$ (each agent stores at most d identifiers at a same time). From this, we get

$$S(i) \leq \sum_{id \in U} \sum_{p \in L_{id}(i)} d = d \sum_{id \in U} |L_{id}(i)| = dZ(i).$$

We can notice that $Z(0) = n$ and that for any i, $Z(i+1) \leq Z(i) + 2$, as an interaction can only add at most one element in the $L(i)$ of the interacting identifiers.

From this, we have $Z(i) \leq n + 2i$ et $Z \leq n + 2T$.

We finally obtain $\sum_{id \in U} M_{id} \leq \sum_{id \in U} N_{id} = S \leq dZ \leq d(n + 2T)$. $\square$

We consider now T as the random variable corresponding to the end of the arrangement protocol (i.e. the first time where all agents know their $Next$ identifier), and let $\mathbb{E}(T)$ be its expectation.

Lemma 4.14

$$\mathbb{E}(T) \geq \frac{1}{\sqrt{20d}} n\sqrt{n}.$$

Proof: We work in the case where $T \leq 2\,\mathbb{E}(T)$.

Let p_k be the probability that $k+1$ agents found their $Next$, supposing that k agents have already found their $Next$.

As $Previous$ is bijective and $\{id \text{ without } Next\} \subset \{id \in U\}$, we have

$$p_k = \frac{1}{n(n-1)} \sum_{id \text{ without } Next} M_{Previous(id)} \leq \frac{1}{n(n-1)} \sum_{id} M_{id}.$$

By applying Lemma 4.13 and the hypothesis $T \leq 2\,\mathbb{E}(T)$, we obtain:

$$p_k \leq \frac{d}{n(n-1)}(n + 4\,\mathbb{E}(T)).$$

We have $\mathbb{E}(T) \geq n$, since the quickest way to end the protocol is by having each agent meeting its $Next$ directly. Hence, $p_k \leq \frac{d}{n(n+1)} 5\,\mathbb{E}(T)$.

Hence, $\mathbb{E}(T | T \leq 2\,\mathbb{E}(T)) = \sum_{k=0}^{n-1} \frac{1}{p_k} \geq \sum_{k=0}^{n-1} \frac{n(n-1)}{5d\,\mathbb{E}(T)} = \frac{n^2(n-1)}{5d\,\mathbb{E}(T)}$.

In the general case, we have:
$\mathbb{E}(T) = \Pr(T \leq 2\,\mathbb{E}(T)) \cdot \mathbb{E}\,(T | T \leq 2\,\mathbb{E}(T)) + \Pr(T > 2\,\mathbb{E}(T)) \cdot \mathbb{E}(T | T > 2\,\mathbb{E}(T))$.

Markov's inequality provides the result that: $\Pr(T \leq 2\,\mathbb{E}(T)) \geq \frac{1}{2}$. Hence, we deduce: $\mathbb{E}(T) \geq \Pr(T \leq 2\,\mathbb{E}(T)) \cdot \mathbb{E}\left(T | T \leq 2\,\mathbb{E}(T)\right) \geq \frac{n^2(n-1)}{10d\,\mathbb{E}(T)}$.

From this, we have $\mathbb{E}(T)^2 \geq \frac{n^2(n-1)}{10d} \geq \frac{n^3}{20d}$. This provides the final result:

$$\mathbb{E}(T) \geq \frac{1}{\sqrt{20d}} n\sqrt{n}$$

□

Lemma 4.14 permits to conclude the proof of Theorem 4.12: the arrangement protocol must take at least $O(n\sqrt{n})$ expected interactions.

Corollary 4.15 *The number of agents who have found their Next, in CPPOLYLOG, has an expectation at most polylogarithmic.*

Proof: Let m be the number of agents who have found their $Next$ after exactly T steps. If we follow the same reasoning as above, we get:

$$\mathbb{E}(T | T \leq 2\,\mathbb{E}(T)) = \sum_{k=0}^{m-1} \frac{1}{p_k} \geq \sum_{k=0}^{m-1} \frac{n(n-1)}{5d\,\mathbb{E}(T)} = \frac{n(n-1)(m-1)}{5d\,\mathbb{E}(T)} \geq \frac{n^2 m}{4\,\mathbb{E}(T)\,d}$$

$$\mathbb{E}(T) \geq \Pr(T \leq 2\,\mathbb{E}(T))\,\mathbb{E}\left(T | T \leq 2\,\mathbb{E}(T)\right) \geq \frac{n^2 m}{10d\,\mathbb{E}(T)}$$

We obtain, $\mathbb{E}(T) \geq \frac{1}{2\sqrt{2d}} n\sqrt{m}$. As we want to have $\mathbb{E}(T) \leq n \log^k n$ for some k, m has to be polylogarithmic. □

4.3.2 The Language (ab)*

We bring now another impossibility result. There is no protocol in CPPOLYLOG computing the rational language $(ab)^*$. Recall that we see inputs as words in Σ^* by sorting the input symbols according to the order between the identifiers.

Theorem 4.16 *The language $(ab)^*$ is not in CPPOLYLOG.*

To prove this result, the idea is to prove that for any protocol, there exists some $k \leq n/2$ such that there is a high enough probability that the protocol acts the same way on the inputs $(ab)^{n/2}$ and $(ab)^k ba (ab)^{n/2-k-1}$.

We work on the pairs (Id_{2k+1}, Id_{2k+2}), and want to prove that there is some k such as, with high probability, these two identifiers never appeared in the same interaction.

To prove that, we first prove the following lemma:

Lemma 4.17 *Let f a function bounded above, for some m, by $n \log^m n$ (for some n large enough).*

To each identifier id, we define the set E_{id} and value M_{id} as
$E_{id} = \{$Agents having had id in one of its register after $f(n)$ steps$\}$ and $M_{id} = |E_{id}|$.

There exists some polylogarithmic function g such as, for n large enough, after $f(n)$ steps:

$$\mathbb{E}(|\{id : M_{id} \leq g(n)\}|) \geq \frac{3}{4}n$$

Proof: Let h be a function such as $\mathbb{E}(|\{id : M_{id} \geq h(n)\}|) \geq \frac{1}{4}n$. In this proof, we will reuse some notations from the proof of Proposition 4.12.

For each identifier id, we associate the set $L_{id} \subset \mathscr{P}(U)$ of the list of identifiers the agent id have had during the $f(n)$ first steps.

Let $S = \sum\limits_{id \in U} \sum\limits_{p \in L_{id}} |p|$. We want to prove that $S \geq \frac{n}{4}h(n)$.

Let N_{id} be the number of occurrences of id in all the L_{id_b}.

We have $N_{id} = \sum\limits_{id_b \in U} |\{p : id \in p \ \&\ p \in L_{id_b}\}|$ and $S = \sum\limits_{id \in U} N_{id}$.

We have $N_{id} \geq M_{id}$ (an agent id appearing at least once in id_b appears at least in one of its lists). As we supposed $M_{id} \geq h(n)$ for at least $n/4$ agents, we get $S \geq \frac{n}{4}h(n)$.

Our purpose is to get a lower bound of $Z = \sum\limits_{id \in U} |L_{id}|$.

We know that for any $id \in U$ and $p \in \mathrm{L}_{id}$, $|p| \leq d$. Then,

$$S \leq \sum_{id \in U} \sum_{p \in L_{id}} d = d \sum_{id \in U} |L_{id}| = dZ.$$

Hence, $Z \geq \frac{n}{4d}h(n)$.

Let Z_i be the value of Z after i steps.

We have $Z_0 = n$ and $Z_{i+1} \leq Z_i + 2$. Hence, $n + 2f(n) \geq Z \geq \frac{n}{4d}h(n)$.

From this, we get $h(n) \leq 4d(1 + \frac{2}{n}f(n))$.

As f is n-polylog, h must be polylogarithmic.

If we chose h maximal matching our initial postulates, we get the expected result:

$$\mathbb{E}(|\{id : M_{id} \leq h(n) + 1\}|) \geq \frac{3}{4}n$$

□

With this first result, we will prove that at most a polylogarithmic number of pairs (Id_{2k+1}, Id_{2k+2}) could have met after n-polylog number of steps.

Lemma 4.18 *Let f be a n-polylog function. For n big enough, after $f(n)$ steps:*

$$\mathbb{E}(|\{k : Id_{2k+1} \text{ and } Id_{2k+2} \text{ were in a same interaction}\}|) \leq \frac{3}{8}n$$

Proof:

For any i and j, let $L_{i,j}$ be the random variable that is equal to 1 if identifiers i and j appeared in a same interaction, 0 otherwise.

We will work on the number of pairs that interacted $N = \sum_{k<n/2} L_{2k+1,2k+2}$. We want to prove that the expectation of this variable is majored by $\frac{3}{8}n$.

$\mathbb{E}(N) = \sum_{k<n/2} \mathbb{E}(L_{2k+1,2k+2}) = \sum_{k<n/2} \Pr(Id_{2k+1} \text{ met } Id_{2k+2} \text{ after } f(n) \text{ steps})$

From Lemma 4.17, we deduce that from at least half of the pairs, each identifier is present on at most $g(n)$ agents. Hence, for these pairs, $\Pr(Id_{2k+1} \text{ met } Id_{2k+2} \text{ after one step}) \leq \frac{g(n)^2}{n(n-1)}$.

$\Pr(Id_{2k+1} \text{ met } Id_{2k+2} \text{ after } f(n) \text{ steps}) \leq 1 - \left(1 - \frac{g(n)^2}{n(n-1)}\right)^{f(n)}$.

Let $\alpha > 1$. The function $l(x) = x^\alpha$ is convex (as $l''(x) = \alpha(\alpha-1)x^{\alpha-2} > 0$). The convexity implies that $l(x) \geq l(y) + l'(y)(x-y)$. With $x = (1-X)$ and $y = 1$, we have:

$(1-X)^\alpha \geq 1 + \alpha(1 - X - 1) = 1 - \alpha X$.

With $X = \frac{g(n)^2}{n(n-1)}$ and $\alpha = f(n)$ (as for n large enough, $f(n)$ is always greater than 1), we have the following inequality:

$\left(1 - \frac{g(n)^2}{n(n-1)}\right)^{f(n)} \geq 1 - \frac{f(n)g(n)^2}{n(n-1)}$

$\Pr(Id_{2k+1} \text{ met } Id_{2k+2} \text{ after } f(n) \text{ steps}) \leq 1 - \left(1 - \frac{g(n)^2}{n(n-1)}\right)^{f(n)} \leq \frac{f(n)g(n)^2}{n(n-1)}$.

As f is n-polylog and g is polylogarithmic, for n large enough, we have $\frac{f(n)g(n)^2}{n(n-1)} \leq \frac{1}{4}$.

If we sum these probabilities for the half number of pairs we considered, we get the upper bound of $\frac{1}{4}\frac{n}{2} = \frac{1}{8}n$ expected pairs that appeared in a same interaction.

Even if all the pairs of identifiers of the not considered half met, the upper bound for the second half provides the result:

$$\mathbb{E}(N) \leq \frac{1}{4}n + \frac{1}{8}n \leq \frac{3}{8}n$$

□

Lemma 4.19 *Let f be a n-polylog function. For any n large enough, there exists $k < n/2$ such as:*

$$\Pr(Id_{2k+1} \textit{ met } Id_{2k+2}) \leq \frac{3}{4}$$

Proof: Let suppose that for any k, $\Pr(Id_{2k+1} \text{ met } Id_{2k+2}) > \frac{3}{4}$.

That implies, $\mathbb{E}(N) = \sum_{k<n/2} \Pr(Id_{2k+1} \text{ met } Id_{2k+2}) > \frac{3}{8}n$.

This is a direct contradiction of the previous lemma. □

To prove our theorem, we need to prove the following proposition:

Proposition 4.20 *For any protocol, for any n-polylog function f, for any input of size n large enough, there exists some k such that the probability that the identifiers Id_{2k+1} and Id_{2k+2} never appeared on a same interaction after $f(n)$ steps is greater than $\frac{1}{4}$.*

This proposition is a direct corollary of previous lemma. With this proposition, the proof of Theorem 4.16 can be done as follows:

Proof: Let P be a protocol computing $(ab)^*$ in less than $n\log^m n$ expected interactions for some $m \in \mathbb{N}$.

We have, from Markov's Inequality that :

Pr(number of steps to compute$\leq 9n\log^m n) \geq \frac{8}{9}$.

It implies that at least $\frac{8}{9}$ of the sequences of configurations of length $9n\log^m n$ provides the right output.

By applying the previous proposition with $f(n) = 9n\log^m n$, we obtain the existence of some k such as the probability that the identifiers Id_{2k+1} and Id_{2k+2} never appeared on a same interaction after $9n\log^m n$ steps is greater than $\frac{1}{4}$.

This implies that in at least $\frac{1}{4}$ of the sequences of configurations of length $9n\log^m n$, Id_{2k+1} and Id_{2k+2} were never in a common interaction.

Hence, if Id_{2k+1} and Id_{2k+2} never appear on a same interaction, then P will not see any difference between the two inputs $(ab)^{n/2}$ and $(ab)^k ba(ab)^{n/2-k-1}$.

Between the $\frac{8}{9}$ of the sequences that provides the right output on these two inputs, at least $\frac{7}{9}$ are common (two sequences are here said to be common if the sequence of the interacting identifiers are equals).

As $\frac{7}{9} \geq 1 - \frac{1}{4}$, amongst those common sequences, some of them does not involve Id_{2k+1} and Id_{2k+2} in a same interaction. As the protocol cannot differentiate those two inputs during those sequences, it cannot bring the right output.

This provides a contradiction. There is no protocol in $CPPOLYLOG$ that computes $(ab)^*$. □

4.4 Set Considerations

4.4.1 SPACE($n\log^k n$)

We provide here a large upper bound:

Proposition 4.21 *For any protocol in $CPPOLYLOG$, there exists some $K \in \mathbb{N}$ such as this protocol is in $SPACE(n\log^K n)$.*

Proof: Any configuration can be described on a space $O(n\log n)$ by writting, for each agent, its state and the list of identifiers (using identifier 1 for the first,... n written in binary for the last).

Let $f(n) = n\log^k n$ for some k being a function bounding by above the number of expected interactions to find the output. We will prove that the result holds considering $K = k + 1$.

The idea is to simulate all the sequences of configurations of length $3f(n)$ (i.e. sequence of $3f(n)$ interactions).

With more details: We can write a sequence by providing the order of the pairs that interacted. Hence, we need a space of $3f(n)\log n$ to encode a sequence on a Turing Machine.

Going from one sequence to the next does not require more space. We then just simulate the sequence of interactions on the population. Finding, from a configuration, if it has an output or not does not require more space.

For each possible output, we use a counter initialized at 0. Each time we computed a sequence, if the configuration does have an output, we increase the corresponding counter.

When all the sequences have been simulated, we provide the same output that the one that got the highest counter.

From Markov's inequality, we get that the probability to use more that $3f(n)$ interactions is less than one third. Hence, more than the half of the $3f$ sequences must provide the right output. The output of our Machine is the same that the protocol's.

This machine is deterministic and uses a space $O(f(n)\log n) = O(n\log^{k+1} n)$. □

Theorem 4.22

$$CPPOLYLOG \subset \bigcup_{k\in\mathbb{N}} SPACE(n\log^k n)$$

$$CPPOLYLOG \subset NSPACE(n\log n) \cap \bigcup_{k\in\mathbb{N}} SPACE(n\log^k n)$$

Proof: The second intersection comes from the fact that functions computed by Community Protocols correspond to $NSPACE(n\log n)$ (cf Remark 1.41). □

4.4.2 Computing $POLYLOGTIME$

We define here a new class of Turing Machine where

- The number of each input symbol in the input is known.
- The machine can only read the beginning of the input.

We prove then that this class can be simulated by protocols of $CPPOLYLOG$.

Definition 4.23 *A language L on input alphabet $\Sigma = \{s_1, \dots, s_\sigma\}$ of σ letters is in* **POLYLOGTIME** *if there exists a deterministic Turing Machine M and some $k \in \mathbb{N}$ such that:*

- *It runs on $\sigma + 1$ tapes.*
- *If we write x on the first tape and n_1, ..., n_s on the σ others, where n_i is the number of symbol s_i in x, with n_i written in binary:*
  - *The machine runs in $O(\log^k n)$ steps, with $n = |x|$.*
  - *The machine accepts x if and only if $x \in L$.*

Theorem 4.24

$$POLYLOGTIME \subsetneq CPPOLYLOG$$

Proof: We saw that it is possible, using a protocol, to write in $O(n \log^4 n)$ interactions the sizes n_i in binary on the $\log n$ first agents of the population (cf Corollary 4.9).

We can see the population sorted by identifiers as a tape. The leader keeps $\sigma + 1$ slots, to track on which cell each reading head is (first, these slots are initialised with its own identifier).

Each agent stores a $\sigma + 1$-tuple. At the initialization of the simulation, the first element contains the input. the ith element of the jth agent correspond to the jth bit of the writing in binary of n_{i-1}. The leader stores the initial state of the simulated machine.

To perform a step, the leader does an epidemic to get the current symbol on each tape read by the head, corresponding for each of the $i \leq \sigma + 1$ identifier to its ith element of the $\sigma + 1$-tuple. It then computes the transition of the simulated machine on its state and the $\sigma + 1$ symbols read.

The leader spreads an epidemic to have each of the agents with a reading head updated its corresponding state. At the same time, for each tape where the leader needs to move the reading head to the right, it looks for the next identifier (and look for the previous identifier if the head goes left). It also updates its own state according to the transition function of the simulated machine.

Simulating a step of the Turing Machine needs two epidemics, so it needs $O(n \log n)$ expected interactions.

The simulation of $O(\log^k n)$ steps of a Machine of $POLYLOGTIME$ needs, at most, $O(n(\log^4 n + \log^{k+1} n))$ expected interactions.

The inclusion is strict, as the sorted language is not in $POLYLOGTIME$. □

4.4.3 A Tighter Bound

The previous Machines permit to compute semilinear predicates, but do not permit to compute the sorted language.

We provide here a more powerful Machine that can be simulated by $CPPOLYLOG$ Protocols, and that computes any language of $POLYLOGTIME$.

Theorem 4.25 *Let a Turing Machine on alphabet Γ recognizing the language L having the following restrictions: There exists some $k \in \mathbb{N}$ such that*

- *The machine has 4 tapes. The first one is for the input x.*
- *The space of work is restricted as follows:*
  - *The first tape uses only the input space of $|x|$ cells.*
  - *The 2nd and the 3rd use at most a space of $\log |x|$ cells.*
  - *The 4th uses at most a space of $\log^k |x|$ cells.*
- *The Machine can only do at most $\log^k |x|$ unitary operations among the following one:*

1. A regular Turing Machine step.

2. Mark/Unmark the cells that have the symbol $\gamma \in \Gamma$.

3. Write in binary on the 2nd tape the number of marked cells.

4. Go to the cell of the number written on the 3rd tape if this number is smaller than $|x|$.

5. Mark/Unmark all the cells left to the pointing head on the first tape.

6. Turn into state γ' all the marked cells in state $\gamma \in \Gamma$.

Then we have $L \in CPPOLYLOG$.

Proof: The protocol will write the 2nd and 3rd tapes on the first $\log n$ agents, the 4th one on the first $\log^k n$ agents.

Let prove that the 6 items can be simulated:

1. It is easy, as the Leader only needs to know on which agent each lecture head is.
2. This one can be performed in a single epidemic.
3. It corresponds to the process described in Section 4.2.2 performed on the agents marked.
4. It is a similar process to finding the median identifier:
   (a) We keep two identifiers Min and Max.
   (b) We take a random agent in $]Min, Max[$.
   (c) We compute the number of agents of smaller identifier.
   (d) Either it is the right one and it is over, either we have an update of Min or Max. In the second case, we go back to step (b) with the new interval.

   We can see that this process will use a logarithmic number of loops in expectation.
5. We perform an epidemic to mark all agents of identifier smaller of the pointing head's.
6. Again, an epidemic is enough.

□

Remark 4.26

- *The sorted language can be computed by such Machines. To do that, for each $s_i \in \Sigma$, it marks all symbol s_j with $j \leq i$. It then counts the number of marked cells. It goes to that number, unmarks all the cell left to the head, and checks that all cells are unmarked.*
- *Any protocol in $CPPOLYLOG$ provided in this document appears to be computable by the Machines describe in Theorem 4.25. We did not find any proof that these Machines are enough, and even any clew of how to perform one.*
- *The class $POLYLOGTIME$ is included in this class of Machines.*

4.5 Conclusion

We worked with a restriction over the number of expected interactions for the community protocols. We gave some protocols and proved that the population is aware of global informations such that the number of agents. More precisely, we know that we can compute:

- The sorted language.
- Any semilinear predicate.
- Any function in $POLYLOGTIME$.

Theorem 4.25 provides the tighter Turing Machine we found that can be simulated by this model.

We provided two impossibility results that raise a problem: a polylogarithmic number of parallel interactions is not enough for all agents to know their direct neighbors. The rational language $(ab)^*$ cannot be computed because any local permutation of letters change the desired output.

No exact characterization has been found for this model. I conjecture that the tighter bound provided in Theorem 4.25 is really close to what can be computed. As for today, I do not see computable sets in $CPPOLYLOG$ that is not computable by the Machine provided in Theorem 4.25.

Chapter 5

Homonym Population Protocols

This chapter is about ***Homonym Population Protocols***. This variant of Community Protocols accepts that several agents may share the same identifier.

We prove that $\log n$ identifiers is enough to simulate a Turing Machine, and that with tuples, we can increase the number of identifiers. We introduce a hierarchy depending of the ratio of identifiers $f(n)$ where n is the size of the population. We provide an exact characterization for the cases $O(1)$ (equivalent to population protocols), $\Theta(\log^r n)$ with $r > 0$ (equivalent to non deterministic Turing Machines on space $\log^k n$ for any k) and $\Theta(n^\epsilon)$ (equivalent to community protocols). We then discuss about the case $o(\log n)$ with non trivial protocols.

Most of the results in this chapter have been published in a work performed with Olivier Bournez and Johanne Cohen in [BCR15] in NETYS 15.

5.1 Model

We studied in the previous chapter what happens when we add time restrictions to the Community Protocols. This chapter follows another restriction: what happens when a same identifier may be shared by several agents.

We will provide a hierarchy depending on the number of different identifiers present on a population of size n (the case $O(1)$ leads to Population Protocols, the case n is exactly Community Protocols).

We will provide results when we have $\Theta(\log^r n)$ identifiers, where r is a positive real number, and when we have $\Theta(n^\epsilon)$ identifiers, with $\epsilon \leq 1$. These results will help to complete, in Chapter 6, a gap in the hierarchy of Passively Mobile Protocols [CMN+11].

Basically, Table 5.1 summarizes the results on the hierarchy according to the number of identifiers, where $MNSPACE(S(n))$ is the set of f-symmetric[1] languages recognized by Non Deterministic Turing Machines on space $O(S(n))$.

[1]These classes are defined in Definition 5.7.

$f(n)$ identifiers	Computational power
$O(1)$	Semilinear Sets Theorem 5.25
$\Theta(\log^r n)$ with $r \in \mathbb{R}_{>0}$	$\bigcup_{k\in\mathbb{N}} MNSPACE\left(\log^k n\right)$ Theorem 5.22
$\Theta(n^\epsilon)$ with $\epsilon > 0$	$MNSPACE(n \log n)$ Theorem 5.23
n	$NSPACE(n \log n)$ [GR09]

Table 5.1: Homonym population protocols with n agents and $f(n)$ identifiers.

5.1.1 Motivation

There is a link between the Community Protocols model and the Passively Mobile Protocols. When passively mobile agents can use a space of $O(\log n)$, article [CMN+11] proves their results by creating identifiers for each agent. Each agent then has a single identity. If we see the agents sorted by their new identifiers, it gets then possible to simulate Community Protocols.

The idea of having less identifiers than agents, that is to say of having "homonyms", has been considered in other contexts or with not closely related problematics [DFG+11, DFT12, AAI+12, DLB+13]. We use this idea to provide the homonymy restriction over Guerraoui and Ruppert's model.

The link described above works again with less identifiers. $O(\log n)$ different identifiers can be described on tapes of size $O(\log \log n)$. Hence, the result of homonym population protocols provide a answer to the question of the case $S(n) = O(\log \log n)$. The result is described in Section 6.1 of the next chapter.

This chapter is organized as follows. Section 5.1 introduces the formal definitions and hypothesis of the model. Section 5.2 is devoted to the case where the number of identifiers $f(n)$ is polylogarithmic. Section 5.3 focuses on the rest of the hierarchy: what happens when $f(n) = \Theta(n^\epsilon)$ and when $f(n) = o(\log n)$.

5.1.2 Definitions

As previously said, the model we consider is a variation of the community protocol model described in Section 1.4. The three big differences are:

- Two agents may have the same identifier. Hence, we provide a function f giving the number $f(n)$ of identifiers present in the population depending of its size n.
- We no longer work on an arbitrary set U of possible identifiers. This time, $U = \mathbb{N}$, and we add another restriction. All the identifiers $u < f(n)$ are present in the population.

- The rules over δ permit to identify when an identifier is equal to 0 and when one identifier is the direct successor of another.

Definition 5.1 *A* **Homonym Population Protocol** *is given by eight elements* $(f, B, d, \Sigma, Y, \iota, \omega, \delta)$ *where:*

- f *is a function* $\mathbb{N} \to \mathbb{N}$ *associating to the size of the population the number of identifiers present. We suppose* $f(n) \leq n$ *for all* n.

  For a population of size n*, the set of identifiers present corresponds exactly to* $[0, f(n)-1]$.

- δ *is a function* $Q^2 \to Q^2$*, with* $Q = B \times U \times (U \cup \{\_\})^d$.

- B*,* $d \in \mathbb{N}$*,* Σ*,* Y*,* ι *and* ω *are the same as in the Community Protocols model.*

The transition function δ *has the two restrictions of Community Protocols. In comparison of the formal explanation of these restrictions we put in Chapter 1, we add two new restrictions. Agents need to know* ***1-*** *when an identifier is equal to* 0 *and* ***2-*** *when two identifiers are consecutive (i.e.* $id_1 = id_2 + 1$*). More formally, we have:*

1. *if* $\delta(q_1, q_2) = (q'_1, q'_2)$*, and* id *appears in* q'_1, q'_2 *then* id *must appear in* q_1 *or in* q_2.
2. *whenever* $\delta(q_1, q_2) = (q'_1, q'_2)$*, let* $u_1 < u_2 < \cdots < u_k$ *be the distinct identifiers that appear in any of the four states* q_1, q_2, q'_1, q'_2*. Let* $v_1 < v_2 < \cdots < v_k$ *be identifiers such that* $u_1 = 0 \Leftrightarrow v_1 = 0$ *and* $v_i + 1 = v_{i+1} \Leftrightarrow u_i + 1 = u_{i+1}$*. If* $\rho(q)$ *is the state obtained from* q *by replacing all occurrences of each identifier* u_i *by* v_i*, then we require that* $\delta(\rho(q_1), \rho(q_2)) = (\rho(q'_1), \rho(q'_2))$.

To avoid the multiplication of names, we will write community protocols for the model of [GR09], and homonym population protocols for our version, with sometimes the precision "with $f(n)$ distinct identifiers".

Remark 5.2

- *This weakening of the original model does not change the computational power in the case where all agents have distinct identifiers (i.e. with* $f(n) = n$*, this model is equivalent to Community Protocols).*

  On the other hand, if we choose $f(n) = 1$*, we come back to the Population Protocols model.*

- *Our purpose is to establish results with minimal hypotheses. Our results work when identifiers are consecutive integers, say* $\{0, 1, 2, \ldots, f(n) - 1\}$.

  I conjecture that without the possibility to know if an identifier is the successor of another one, the model is far too weak. Without this asumption, our first protocol (in Proposition 5.10) does not work.

- *I think that knowing whether an identifier is equal to 0 is not essential. It is used in the protocol of Proposition 5.10, but we provide an inch on how to work without this hypothesis.*

- *Most of the definitions still hold. Just the notion of input needs to be redefined with this model.*

As a reminder, we will use the notations explained in Remark 1.36.

Example 5.3 (The Chain Protocol) *Here is a little adaptation of the usual Leader Election. This time, instead of having a unique agent different from all the others, we get one for each identifier.*

The protocol is quite simple. We use the Leader Election rules, adding the restriction that two agents truly interact only if they have the same identifier. Here is the protocol:

- $B = \{L, N\}$.

- $d = 0$.

- $\Sigma = L$, $Y = \{True\}$, $\iota(L) = L$ *and* $\omega(L) = \omega(N) = True$.

- δ *has only one difference from the Leader Election of Example 1.37: the first rule is split according to the identifiers.*

$$\begin{array}{ll} L_{id_a}\ L_{id_a} \to L_{id_a}\ N_{id_a} & \\ L_{id_a}\ L_{id_b} \to L_{id_a}\ L_{id_b} & \text{with } id_a \neq id_b \end{array}$$

At some point, there will be exactly one L_{id} agent for each $id \leq f(n)$. We will note L_{id} a/the leader with identifier id.

For the rest of this section, **The Chain** *will refer to these particular agents. We will often see these agents as a tape of $f(n)$ symbols in B sorted according to the identifiers.*

When we talk about the **Leader**, *we always talk about L_0, that is to say the leader with identifier 0.*

Definition 5.4 *An* **Input** *of size $n \geq 2$ is $f(n)$ non empty multisets X_i over alphabet Σ, one for each of the $f(n)$ identifiers. The* **Initial Configuration** *for n agents is a vector in Q^n of the form $((\iota(x_j), i-1, \_, \ldots, \_))_{1 \leq i \leq f(n), 1 \leq j \leq |X_i|}$ where x_j is the jth element of X_i: in other words, every agent starts in a basic state encoding $\iota(x_j)$, its associated identifier and no other identifier stored in its d slots.*

Definition 5.5 *Let I be a positive integer and let $(X_i)_{i \leq I}$ be a finite sequence of multisets of Σ.*

$(X_i)_{i \leq I}$ is in the **Included Language** *if and only if, for all $i < I$, $\emptyset \neq X_{i+1} \subset X_i$.*

This language corresponds to finite sequences of non-empty multisets where each multiset is included in the previous one.

Proposition 5.6 *The included language can be computed by a homonym population protocol.*

Proof: We will provide a protocol that works for any f. The idea is that each agent with an identifier $i > 0$ looks for an agent of identifier $i - 1$ with the same input to "delete" it.

We have that $X_{i+1} \subset X_i$ if and only if each agent in X_{i+1} manages to "delete" an agent with the same input in X_i.

- $B = \Sigma \times \{I, D\}^2 \times \{T, t, F, f\}$.
- $d = 0$.
- $Y = \{True, False\}$.
- $\forall s \in \Sigma$, $\iota(s) = (s, I, I, F)$.
- $\forall s \in \Sigma, \forall a, b \in \{I, D\}^2, \omega(s, a, b, T) = \omega(s, a, b, t) = True, \omega(s, a, b, F) = \omega(s, a, b, f) = False$.
- δ is such that the non trivial rules are:

$$\begin{array}{llll}
(s,I,b,F)_0 \;\; q & \rightarrow (s,D,b,T)_0 \;\; q & & \forall b, q \\
(s,a,I,c)_i \;\; (s,I,b,c')_{i+1} & \rightarrow (s,a,D,c)_i \;\; (s,D,b,T)_{i+1} & & \forall a,b,c,c' \\
(s,a,b,F)_i \;\; (s',a',b',c')_j & \rightarrow (s,a,b,F)_i \;\; (s',a',b',f)_j & & \forall a,b,a',b',c' \\
(s,a,b,T)_i \;\; (s',a',b',f)_j & \rightarrow (s,a,b,T)_i \;\; (s',a',b',t)_j & & \forall a,b,a',b',c'
\end{array}$$

Each agent has 4 elements:

1. The first is its input symbol.
2. The second is I if the agent has not yet "deleted" an agent with the same input and with the previous identifier. It is D if the deletion has been performed.
3. The third element is I if it has not been deleted yet by an agent with the successor identifier. It is D as soon as the deletion has been performed.
4. The fourth element corresponds to the output. State F means that the agent needs to perform a deletion. The agent knows that the input has to be $False$ as long as it has not deleted an agent with the previous identifier. State f means the agent believes that at least one deletion needs to be performed. State T means that the agent made its deletion and since did not meet agent needing to perform a deletion. State t means that the agent believes that no deletions need to be done.

Agents with identifier 0 does not need to do a deletion, Rule 1 handles that. Rule 2 handles a deletion.

If we project on the fourth element, the stable configuration are T^*t^* and F^*f^*.

Rule 3 spreads the output $False$ to each agent. It can only come from an agent still waiting for a deletion. If a deletion needs to be done and cannot be, the output $False$ will be spread by fairness.

Rule 4 spreads the output $True$ from T to f agents. If there are no longer deletions to do, there is at least one T in the population, being from the agent that has performed the last deletion. It will spread the output $True$ by fairness. $\square$

The purpose of this chapter is to determine exactly what can be computed with homonym protocols. To find out the answer, we first need to introduce Turing Machines that has inputs analog to homonym protocols.

It needs to associate to each $k \leq f(n)$ a multiset of Σ. To define it properly, we first introduce the notion of f-Symmetry, and then give the definitions of the two classes $MNSPACE$ and $SMNSPACE$ (the second one will be used later, when we accept that the set U of identifiers to be unordered).

Definition 5.7 *A Language $L \in (\Sigma \cup \{\#\})$ is* **f-Symmetric** *if and only if:*

- $\# \notin \Sigma$*;*
- *Words of L are all of the form $w = x_1\#x_2\#\ldots\#x_{f(n)}$, with $|x_1|+|x_2|+\ldots+|x_{f(n)}| = n$ and $\forall i,\ x_i \in \Sigma^+$;*
- *If, $\forall i$, x'_i is a permutation of x_i, and if $x_1\#x_2\#\ldots\#x_{f(n)} \in L$, then $x'_1\#x'_2\#\ldots\#x'_{f(n)} \in L$;*

Each x_i is a non-empty multiset over alphabet Σ.

Let S be a function $\mathbb{N} \to \mathbb{N}$.

We write **MNSPACE(S(n),f(n))**, *or* **MNSPACE(S(n))** *when f is unambiguous, for the set of $f-$symmetric languages recognized by Non Deterministic Turing Machines on space $O(S(n))$.*

We write **SMNSPACE(S(n),f(n))**, *or* **SMNSPACE(S(n))** *when f is unambiguous, for the set of $f-$symmetric languages recognized by Non Deterministic Turing Machines on space $O(S(n))$, where languages are also stable under the permutation of the multisets (i.e. for any permutation σ, $x_1\#x_2\#\ldots\#x_{g(n)} \in L \Leftrightarrow x_{\sigma(1)}\#x_{\sigma(2)}\#\ldots\#x_{\sigma(g(n))} \in L$).*

Remark 5.8 *We have $NSPACE(S(n)) = MNSPACE(S(n), n)$ (as each multiset contains exactly one element) and $SNSPACE(S(n)) = MNSPACE(S(n), 1)$ (as accepting a multiset is exactly being stable under input permutation).*

Proposition 5.9 *The included language is in $MNSPACE(\log n)$.*

Proof: To check if $X_{i+1} \subset X_i$ it suffices to check, for each $s \in \Sigma$, if $|X_{i+1}|_s - |X_i|_s \leq 0$. This can be done on a space $O(\log n)$.

We accept if and only if we did not find i and s such that $|X_{i+1}|_s - |X_i|_s \leq 0$. $\square$

5.2 A Polylogarithmic Number of Identifiers

In this section, we focus on the case where we have $f(n) \geq \log n$ identifiers. We will prove that the size of the population can be computed (by writing it in binary on $\log n$ agents). We will then show how to "read the input" and how to simulate a tape of length $\log n$. To perform that, we will explain a process that ensures that the protocol will at some point do exactly what is expected.

5.2.1 The Size of the Population

In population protocols, the size, that is to say the number of agents, can just be seen written in unary (1 for each agent). However, the protocol can not use the value of the population size with this encoding: an agent that took too much time to interact with the rest of the population risk not to be detected.

We introduce here a way to track the size the population permitting, at some point, to be sure to work on the whole population, not missing anyone anymore.

As in Chapter 4, we will write in binary the size of the input on $\log n$ agents (see Section 4.2.2). In the $CPPOLYLOG$ class, the uniqueness of identifiers helped to chose those $\log n$ agents. With Homonym Population Protocols, the protocol will use the chain (explained in Example 5.3). The size of the population will be written in binary on this chain (it will be possible as $f(n) \geq \log n$ in this part).

Clearly, once such a chain has been constructed, it can be used to store numbers or words, and can be used as the tape of a Turing Machine. We will often implicitly use in our description this trick in what follows. This will be used to simulate Turing machines in an even trickier way, in order to reduce space or identifiers.

Proposition 5.10 *When we have $f(n) \geq \log n$ identifiers, there exists an homonym population protocol that computes the size n of the population: At some point, there are exactly $f(n)$ agents not in a particular state N, all having distinct identifiers. If we align these agents from the highest identifier to the lowest one, we get n written in binary.*

Proof: Informally, the protocol initializes all agents to a particular state A. This protocol will implicitly include the chain construction (instead of being in state L, potential leaders will have their state in $\{A, 0, 1, 2\}$). This protocol counts the number of agents in state A. An agent in state 1 (respectively 0, or 2) with identifier k represents 2^k (respectively 0, or 2^{k+1}) agents counted. Interactions between agents update those counts.

More formally, here are the rules:

$$\begin{array}{lllll}
A_0 \; q_k \rightarrow 1_0 \; q_k & \forall q, k & \qquad & 0_k \; 1_k \rightarrow N_k \; 1_k & \forall k \\
A_k \; 0_0 \rightarrow 0_k \; 1_0 & \forall k \geq 1 & & 1_k \; 1_k \rightarrow N_k \; 2_k & \forall k \\
A_k \; 1_0 \rightarrow 0_k \; 2_0 & \forall k \geq 1 & & 0_{k+1} \; 2_k \rightarrow 1_{k+1} \; 0_k & \forall k \\
0_k \; 0_k \rightarrow N_k \; 0_k & \forall k & & 1_{k+1} \; 2_k \rightarrow 2_{k+1} \; 0_k & \forall k
\end{array}$$

This protocol has 3 steps. (i) At the beginning, all agents are in state A. A state A is transformed into a state 1, by adding 1 to an agent of identifier 0 (the 3 first rules). (ii) Rules 4 to 6 are here to perform in parallel the chain protocol. (iii) The remaining rules correspond to summing together the counted agents, carrying on to the next identifier the 1.

Let v be the function over the states defined as follows for any k: $v(A_k) = 1$, $v(0_k) = v(N_k) = 0$, $v(1_k) = 2^k$, $v(2_k) = 2^{k+1}$. We can notice that the sum (of v values) over all the agents remains constant over the rules. Thus the sum always equals the number of agents in the population.

By fairness, all the A will disappear, the chain will finish, and the 2_k will disappear. Hence, the protocol writes the size of the population on the chain. $\square$

Remark 5.11

- *The previous counting protocol also works with $f(n) = \Omega(\log n)$. Indeed, if for some $\alpha < 1$ we have $f(n) \geq \alpha \log n$, then, using a base $\lceil e^{1/\alpha} \rceil$ instead of a base 2 allows that n can be written on $f(n)$ digits.*

- *We use here the fact that the population knows if an identifier is equal to 0. We could instead work with identifiers in $[a, a+f(n)-1]$. For this, agents store an identifier Id_m corresponding to the minimal one he saw (called here Id_m). An agent with identifier Id and state $i \in \{0,1,2\}$ stores $i \cdot 2^{Id-Id_m}$. When it meets an identifier equals to $Id_m - 1$, it looks for a leader with identifier $Id - 1$ to give it its stored integer.*

From now on, when a proof says that the population uses its size, we assume that the counting protocol has been performed and the protocol uses the chain to access to this information. Once again, the value of the size is encoded in the population (no agent knows it by itself).

5.2.2 Resetting a Computation

The computation of the value of size n (encoded as above) is crucial for the following protocols. As a reminder, the (or a) leader is an agent not in state N with identifier 0 from the previous protocol (or the chain protocol).

We now provide a Reset protocol. This protocol has the goal to reach a configuration such that (i) the protocol of the previous section is over, (ii) all agents but the leader are in state R, and (iii) the leader "knows" when this configuration is reached (i.e. the leader is on a state F making him the belief the reset is done. The last time the protocol turn the leader's state into F, the reset will be done).

This protocol then permits to launch the computation of some other protocols with the guarantee that, by fairness, at some point, all agents were counted, and hence covered.

Definition 5.12 *A* **Reset Protocol** *is a homonym protocol that guarantees to reach a configuration where:*

- *The size of the population has been determined. Hence the population has a unique leader at that point.*
- *There exists a mapping function map* : $\Sigma \rightarrow \{R, F\}$ *such that map of the leader's state is equal to* F *and map of all other agents' state is* R.

This configuration will be the beginning of the next computation. All agents will be ready, at this point, to start the next algorithms.

Proposition 5.13 *There exists a Reset Protocol.*

Proof: The idea of this protocol is to start back the reset protocol each time the leader sees that the counting protocol (of the previous proposition) has not finished yet.

It turns agents in state R. It uses the chain to count the number of agents it turns into state R. If it has turned the same number of agents that the number computed by the first protocol, it turns its state into F.

This mechanism is not as simple as it could look: The protocol uses the fairness to be sure that at some point, it will have turned the right number of agents into state R.

More formally, the set of states is $B_c \times B_c \times \{A, B, C, D, D^p, D'^p, E, F, R, S, W\}$, where B_c is the set of states of the counting protocol (and δ_c its interaction function). The first element of the triplet is for the counting protocol, the second for the counting of agents turned into state R, and the third part is for the reset itself. At the beginning, the first element of each agent is A, the second is 0 and the third is A. We can notice that the leaders for each identifier will be the same on the first and second element, and hence the chains will be the same for the two counts.

The second counting protocol will work a bit differently for the carry over. The leader performs it by itself, walking through the chain, instead letting it doing itself like in Proposition 5.10.

This protocol uses two identifiers slots. The first will be attached to the first counter: the leader stores the greater identifier it heard about (each time it updates it, we consider it has updated its state). The second is attached to the second counter.

Sometimes, rules will be of the form:

$$(q_1, q_2, q_3)\ (q_4, q_5, q_6) \rightarrow (q'_1, q_2, q'_3)\ (q'_4, q_5, q'_6).$$

It will implicitly suppose that $\delta_c(q_1, q_4) = (q'_1, q'_4)$. If we write q_1 (resp q_4) instead of q'_1 (resp q'_4), it implies that $q'_1 = q_1$ (resp $q'_4 = q_4$). We will note $q_{1,id}$ if the identifier id attached to q_1 is relevant (same for q_2).

We will describe δ according to the steps of the process of resetting:

1. First, the leader needs to know when the counting protocol evolved. For this, as soon as an interaction occurred, agents not being a leader go into state W to warn the leader to start back the reset. The leader L_0 then goes into state A on its third element. (We can notice that in δ_c, there is a non trivial interaction with a leader if and only if it is the second element.)

$$\begin{array}{llll}(q_1, q_2, q_3)\ (q_4, q_5, q_6) & \rightarrow (q'_1, q_2, W)\ (q'_4, q_5, q_6) & \text{with } q'_1 \neq q_1 \text{ or } q'_4 \neq q_4 \\ (q_1, q_2, q_3)_0\ (q_4, q_5, W) & \rightarrow (q'_1, q_2, A)_0\ (q'_4, q_5, S) & \text{with } q_1 \neq N\end{array}$$

2 In state A, the leader turns agent into state S. Its goal is to turn them all. At some point (when the leader is the second element of an interaction), it stops and goes to state B.

The idea is that by fairness, if we repeat again and again this process, at some point, the leader will manage to have all other agents turned into state S.

$$\begin{array}{llll}(q_1, q_2, A)_0\ (q_4, q_5, q_6) & \rightarrow (q_1, q_2, A)_0\ (q_4, q_5, S) & \text{with } q_1 \neq N \\ (q_1, q_2, q_3)\ (q_4, q_5, q_6)_0 & \rightarrow\ (q_1, q_2, S)\ (q_4, 1_{\_}, B)_0 & \text{with } q_4 \neq N\end{array}$$

3 In state B, the leader clears the second chain. This way, after the last change from the counting protocol, we are sure that the chain will be cleared and will effectively count all the agents. The identifier attached to the first state gives to the leader the biggest identifier it saw. This way, it knows the last element's identifier in the chain.

The leader just keeps its own bit at 1, as it needs to count itself. After that, the leader goes to state C.

$$\begin{array}{llll}(q_1, q_{2,\_}, B)_0\ (q_4, q_5, q_6) & \rightarrow\ (q_1, 1_0, B)_0\ (q_4, q_5, q_6) & \\ (q_{1_i}, 1_j, B)_0\ (q_4, q_5, q_6)_{j+1} & \rightarrow (q_1, 1_{j+1}, B)_0\ (q_4, 0, q_6)_{j+1} & \text{with } j < i\text{-}1 \\ (q_{1_i+1}, 1_i, B)_0\ (q_4, q_5, q_6)_{i+1} & \rightarrow\ (q_1, 1_{\_}, C)_0\ (q_4, 0, q_6)_{i+1} & \end{array}$$

4 In state C, the leader looks only for S agents. When it finds one, it turns it into a R and adds 1 to the second counter (by going to state D). If it meets another state, it goes back to state A, in order to try to turn again all agents into S and to reset the counter.

$$\begin{array}{llll}(q_1, q_2, C)_0\ (q_4, q_5, S) & \rightarrow (q_1, q_2, D)_0\ (q_4, q_5, R) & \\ (q_1, q_2, C)_0\ (q_4, q_5, q_6) & \rightarrow (q_1, q_2, A)_0\ (q_4, q_5, S) & \text{with } q_6 \neq S\end{array}$$

5 In state D, the leader increases the second counter by 1. If it has a carry, it goes in state D^p as long at it is needed. When the incrementation is over, it goes back to state D until it reaches the end of the chain or find a difference. If it finds a difference, and if it still has to propagate the carry, then it goes in state D'^p.

Here, we see why the agents need another extra slot of identifier: the leader needs to remember what was the last bit he saw (to identify easily the next one to find). The identifier on the first state permits to know until which identifier it has to compare the two counters.

If it reaches the last bit of the chain and the two counters are equal, then the leader believes the reseting is over and goes to state F (until the counting protocol on the first element gets an update, if it happens). If the counter is not yet equal, then it looks for another S to turn into a R.

$$\begin{array}{rlcrll}
(0_i, 0_{\_}, D)_0 & (q_4, q_5, q_6) & \rightarrow & (1_i, 0_{\_}, C)_0 & (q_4, q_5, q_6) & \\
(1_i, 0_{\_}, D)_0 & (q_4, q_5, q_6) & \rightarrow & (1_i, 1_0, D)_0 & (q_4, q_5, q_6) & \\
(1_i, 1_{\_}, D)_0 & (q_4, q_5, q_6) & \rightarrow & (1_i, 0_0, D'^p)_0 & (q_4, q_5, q_6) & \\
(0_i, 1_{\_}, D)_0 & (q_4, q_5, q_6) & \rightarrow & (0_i, 0_0, D^p)_0 & (q_4, q_5, q_6) & \\
(q_{1,i}, q_{2,j}, D)_0 & (a, a, q_6)_{j+1} & \rightarrow & (q_{1,i}, q_{2,j+1}, D)_0 & (a, a, q_6)_{j+1} & \text{with } a \in \{0,1\} \text{ and } j < i\text{-}1 \\
(q_{1,i}, q_{2,j}, D)_0 & (a, 1\text{-}a, q_6)_{j+1} & \rightarrow & (q_{1,i}, q_{2,\_}, C)_0 & (a, 1\text{-}a, q_6)_{j+1} & \text{with } a \in \{0,1\} \\
(q_{1,i+1}, q_{2,i}, D)_0 & (a, a, q_6)_{i+1} & \rightarrow & (q_{1,i+1}, q_{2,\_}, F)_0 & (a, a, q_6)_{i+1} & \text{with } a \in \{0,1\} \\
(q_{1,i}, q_{2,j}, D'^p)_0 & (q_4, 0, q_6)_{j+1} & \rightarrow & (q_{1,i}, q_{2,\_}, C)_0 & (q_4, 1, q_6)_{j+1} & \\
(q_{1,i}, q_{2,j}, D'^p)_0 & (q_4, 1, q_6)_{j+1} & \rightarrow & (q_{1,i}, q_{2,j+1}, D'^p)_0 & (q_4, 0, q_6)_{j+1} & \\
(q_{1,i}, q_{2,j}, D^p)_0 & (0, 0, q_6)_{j+1} & \rightarrow & (q_{1,i}, q_{2,\_}, C)_0 & (0, 1, q_6)_{j+1} & \\
(q_{1,i}, q_{2,j}, D^p)_0 & (1, 0, q_6)_{j+1} & \rightarrow & (q_{1,i}, q_{2,j+1}, D)_0 & (1, 1, q_6)_{j+1} & \text{with } j < i\text{-}1 \\
(q_{1,i}, q_{2,j}, D^p)_0 & (0, 1, q_6)_{j+1} & \rightarrow & (q_{1,i}, q_{2,j+1}, D^p)_0 & (0, 0, q_6)_{j+1} & \\
(q_{1,i}, q_{2,j}, D^p)_0 & (1, 1, q_6)_{j+1} & \rightarrow & (q_{1,i}, q_{2,j+1}, D'^p)_0 & (1, 0, q_6)_{j+1} & \\
(q_{1,i+1}, q_{2,i}, D^p)_0 & (1, 0, q_6)_{i+1} & \rightarrow & (q_{1,i+1}, q_{2,\_}, F)_0 & (1, 1, q_6)_{j+1} &
\end{array}$$

To prove that this protocol will succeed, we know by fairness that the counting protocol will finish at some point. There will be several agents in state W. By fairness, the leader will have seen all of them at some point.

Let consider a configuration appearing infinitely often. We will show that the reset point can be reached from it (and then will be reached by fairness).

- If the leader has its third state equal to A, then we have it met all the agents to turn them into S. Then, we go into state B in a population where all other agents are in state S.

- If the leader is in state B, the leader has a deterministic path as there are only single agents for each identifier in the chain. We then reach C, keeping the number of S in the population.

- If the leader is in state B, we have two cases:

  - All agents are in state S. It can be the case only if the leader's last operation was becoming a C from a B, hence the second counter is equal to 1. Then, if the leader repeats the actions (turn an agent from S to R + increment the counter), at some point the two counters will match, and the leader will reach the state F.

  - At least one agent is not in state S. We have the leader interact with it to go in state A.

- If the leader is in state $\{C, C^p, C'^p\}$, we can finish Step 5. The leader will then be in state C or F.

- If the leader is in state F, then the two counters must be equal. Hence, the leader turned exactly the right number of agents from state S to R, the reset is performed.

The population will reach at some point the desired configuration. □

Now, when we want to run a computation, the leader starts it just after being in state F with the reset protocol. Each time the leader turns agent into a S, it resets its state according to the function ι of the protocol that will be run.

Hence, once the reset is over, the population is reset as wished.

Remark 5.14 *Notice that after a first reset, the leader can turn at any moment the whole population in a state R: It just has to count the number of agents it turns until it has turned the size of the population (written on the chain) -1.*

This protocol is the strength needed to be sure to have access to all the input at some point. The weakness of population protocols is that L can be simulated, but with a probability of failure: Epidemics manages to touch with high probability each agent, but not surely.

Here, counting each agent touched and checking that it matches the size of the population permits to perform 100% effective epidemics.

5.2.3 Access to the Input

In order to simulate the reading of the input tape, we first introduce a protocol that computes the number of agents that had the input $s \in \Sigma$ and was given the identifier Id.

This is a sub-protocol that can be used at any moment by the main protocol. Hence, the information cannot be kept definitively, and has to be given precisely at the requested moment. We cannot accept any error, as it could not be detected on time to correct the computation.

Proposition 5.15 *When we have $f(n) = \Omega(\log n)$ identifiers, if the reset protocol has finished, for all input $s \in \Sigma$ and for all $Id \leq f(n)$, there exists a protocol that writes on the chain the number of agents initialized as s_{Id}.*

Proof: We suppose that the population is already reseted to the state R.

We do not give here the exact protocol, only how it works:

0. The agents will use 4 states: one for the counting protocol (that is assumed to be already finished, as the reset one is), one for the counting of agents in the input s_{Id}, and one for counting again the total. The last element is here to know when the leader has met every agent to check if they have started in state s_{Id} or not. The last state is in $\{Y, N\}$ and is equal to Y if and only if the leader already counted it.

1. The leader looks for an agent it has not recounted again (i.e. with its 4th state equals to N). When it meets one, its switches N to Y, it looks if its input was s_{Id} or not. If it is, it increments the second and the third counter, otherwise it increments only the third.

2. The leader then looks if the first and the third counter are equal. If not, it goes back to step 1, if yes the computation is over.

Since the counting protocol is over (if not, the population will be reseted again and again until the counting is over), the size is known. With that, we are sure to have counted each agent started in state s_{Id}, as the leader must have seen each agent in this protocol before finishing it. □

Remark 5.16 *In other words, if at some moment, the population needs to know the number of agents which started in the state s_{Id}, this is possible.*

5.2.4 Turing Machine Simulation

With all these ingredients we will now be able to access to the input easily. We will then also use the chain to simulate a tape of a Turing Machine.

Here is a weaker bound than the one we will obtain. The idea of this proof helps to understand the stronger result.

Proposition 5.17 *Any language in $MNSPACE(\log n, \log n)$ can be recognized by an homonym population protocol with $\log n$ identifiers.*

Proof: The main idea of this proof is to use the chain as a tape for a Turing Machine. To simulate the tape of the Turing Machine, we store the position where the head of the Turing machine is by memorizing on which multiset the head is (via the corresponding identifier) and its relative position inside this multiset: the previous protocol will be used to find out the number of agents with some input symbol in the current multiset, in order to update all these information and simulate the evolution of the Turing Machine step by step.

More precisely, let $M \in MNSPACE(\log n, \log n)$. There exists some $k \in \mathbb{N}$ such that M uses at most $k \log n$ bits for each input of size n. To an input $x_1\#x_2\#\ldots\#x_{f(n)}$ we associate the input configuration with, for each $s \in \Sigma$ and for each $i \leq f(n)$, $|x_i|_s$ agents in state k with the identifier $(i-1)$, $|x_i|_s$ being the number of s in x_i.

The idea is to use the chain as the tape of the Turing Machine. We give k bits to each agent, so that the protocol has a tape of the good length (the chain is of size $\log n$). We just need to simulate the reading of the input (the writing will be intuitively be performed by the leader keeping track the identifier of the agent where the machine's head is). The protocol starts by counting the population and resetting agents after that.

We assume that symbols on Σ are ordered. Since the language recognized by M is $\log n$-symmetric, we can reorganize the input by permuting the x_i's such that the input symbols are ordered (i.e. $\Sigma = \{s1, s2, \ldots, \}$ and $x_i = s1s1\ldots s1s2\ldots$).

Here are the steps to perform the simulation of reading the tape:

0. The chain contains two counters. The leader also stores an identifier Id and a state s. The first counter stores the total of s_{Id} computed at some point by the protocol of Proposition 5.15. The second counter c_2 is the position the reading head. The simulated head is on the c_2th s of x_{Id+1}.

1. At the beginning of the protocol, the population counts the number of $s1$, where $s1$ is the minimal element of Σ. c_2 is initialized to 1.

2. When the machines needs to go right on the reading tape, c_2 is incremented. If c_2 equals c_1, then the protocol looks for the next state s' in the order of Σ, and count the number of s'_{Id}. If this value is 0, then it takes the next one. If s was the last one, then the reading tape will consider to be on a #.

   If the reading head was on a #, then it looks for the successor identifier of Id, and counts the number of $s1$. If Id was maximal, the machine knows it has reached the end of the input tape.

3. The left movement process is similar to this one.

This protocol can simulate the writing on a tape and the reading of the input.

To simulate the non deterministic part, each time the leader needs to make a non deterministic choice between two possibilities, it looks for an agent. If the first agent the leader meets has its identifier equal to 1, then the leader does the first choice, otherwise it choses the second one. When the computation is over, if it rejects, it reset the simulation and starts a new one.

By fairness, if there is a path of non deterministic choices for the machine, the protocol will at some point use it and accept the input, as would do M. If not, as all the agents will stay in a refusing state, the protocol will reject the input.

This protocol simulates M. $\square$

Corollary 5.18 *Let f such that $f(n) = \Omega(\log n)$. Any language in $MNSPACE(f(n), f(n))$ can be recognized by an homonym population protocol with $f(n)$ identifiers.*

Proof: We use the same protocol (which is possible as the size of the population can be computed). Since the chain of identifiers has a length of $f(n)$, we have access to a tape of size $f(n)$. $\square$

5.2.5 Polylogarithmic Space

We prove now the exact characterization of what can be computed by our model: functions computable by Turing Machines on polylogarithmic space. To prove it, we first prove several propositions. The combination of the three following results permit to conclude the main theorem.

Proposition 5.19 *Let f such that $f = \Omega(\log n)$. Let k be a positive integer.*

Any language in $MNSPACE\left(\log^k n, f(n)\right)$ can be recognized by a protocol with $f(n)$ identifiers.

Proof: The idea here is that, by combining several identifiers together, we get much more identifiers available, increasing the chain and space of computation: Indeed, if we combine m identifiers together in a m-tuple, we get $f(n)^m$ possible identifiers.

First the population performs the computation of the size of the population. Second, it gets a chain of all the identifiers. Third, the leader then creates a counter of m identifiers, initialized at 0^m (seen as the number $0 \dots 0$ written in base $f(n)$). Fourth, it looks for an agent in state N and gives it its stored m-tuple, then increases its m-tuple. As soon as it has finished (by giving $f(n)^m$ or n identifiers, depending on what happens first), the protocol can work on a tape of space $f(n)^m$.

Since $f(n) = \Omega(\log n)$, there exists some m such that $f(n)^m \geq \log^k n$. □

Proposition 5.20 *Let f such that there exists some real $r > 0$ such that we have $f(n) = \Omega(\log^r n)$.*

Any language in $\bigcup_{k \geq 1} MNSPACE(\log^k n, f(n))$ can be recognized by an homonym population protocol with $f(n)$ identifiers.

Proof: We only need to treat the counting protocol when $r < 1$ (the case $r = 1$ is treated in Proposition 5.19, the case $r > 1$ is a direct corollary of this proposition). Let $l = \lceil \frac{1}{r} \rceil$. We will use a l-tuple for each agent. When agents realize that $f(n)$ might be reached and they need more space, they use the tuple, storing the maximal identifier Id_1. If at some point, they realize that a higher identifier Id_2 exists, they just do a translation of the numbers stored in the chain.

With $f(n)^l = \Omega(\log n)$ identifiers and the right basis to write the size, we can be sure to have enough space to compute the size of the population. We can then use previous protocols using $(k \cdot l)$-tuples to use the required space. □

Proposition 5.21 *Consider a predicate computed by a protocol with $f(n)$ identifiers. Assume that $f(n) = O(\log^l n)$ for some $l \geq 1$.*

The predicate is in $MNSPACE(\log^k n, f(n))$ for some positive integer k.

Proof: We need to prove that there exists a Turing Machine that can compute, for any given input x, the output of protocol P.

From definitions, given some input x, P outputs the output y on input x if and only if there exists a finite sequence $(C_i)_{i \in \mathbb{N}}$, starting from an initial condition C_0 representing x, that reaches at some finite time j some configuration C_j with interpretation y, and so that any configuration reachable from C_j has also interpretation y.

This latter property can be expressed as a property on the graph of configurations of the protocol, i.e. on the graph whose nodes are configurations of n agents, and whose edges corresponds to unique step reachability: one must check the existence of a path from C_0 to some C_j with interpretation y so that there is no path from C_j to some other C' with interpretation different from y.

Such a problem can be solved in $NSPACE(\log N)$ where N denotes the number of nodes of this graph of configurations. A similar proof has already been used in Remark

1.41. Indeed, guessing a path from C_0 to some C_j can easily be done in $NSPACE(\log N)$ by guessing intermediate nodes (configurations) between C_0 and C_j. There remains to see that testing if there is no path from C_j to some other C' with interpretation different from y can also be done in $NSPACE(\log N)$ to conclude.

But observe that testing if there is a path from C_j to some other C' with interpretation different from y is clearly in $NSPACE(\log N)$ by guessing C'. From Immerman-Szelepcsnyi's Theorem [Imm88, Sze88] we know that $NSPACE(\log N) = co-NSPACE(\log N)$. Hence, testing if there is no path from C_j to some other C' with interpretation different from y is indeed also in $NSPACE(\log N)$.

It remains now to evaluate N: For a given identifier i, an agent encodes basically some basic state $b \in B$, and d identifiers $u_1, u_2, \ldots, u_d$. There are at most n agents in a given state $(i, b, u_1, u_2, \ldots, u_d)$. Hence $N = O(n^{|B| \cdot f(n)^{d+1}})$. In other words, the algorithm above in $NSPACE(\log N)$ is hence basically in $MNSPACE((|B| \cdot f(n)^{d+1}) \log n, f(n))$, which is included in $MNSPACE(\log^k n, f(n))$ for some k. □

Theorem 5.22 *Let f such that for some r, we have $f(n) = \Omega(\log^r n)$. The set of functions computable by homonym population protocols with $f(n)$ identifiers corresponds exactly to $\bigcup_{k\geq 1} MNSPACE(\log^k n, f(n))$.*

5.3 The Rest of the Hierarchy

5.3.1 Population with n^ϵ Identifiers

One can go from n^ϵ (with $\epsilon > 0$) to a space of computation equivalent to the case where $f(n) = n$: We just need to use a k-tuple of identifiers, as in Proposition 5.19.

Theorem 5.23 *Let f such that there exists some $k \in \mathbb{N}$ such that $f(n) \geq n^{1/k}$. The set of functions computable by homonym population protocols with $f(n)$ identifiers corresponds exactly to $MNSPACE(n \log n, f(n))$.*

Remark 5.24 *This result does not need the two restrictions of knowing if an identifier is equal to 0 or if two identifiers are consecutive. The result holds when the set of possible identifiers U is chosen arbitrarily and when the restrictions over the rules are those in [GR09].*

5.3.2 Population with $o(\log n)$ Identifiers

This time, we consider we have really few identifiers. To write the size of the population in binary, we need to differentiate $\log n$ agents. With $o(\log n)$ identifiers, it is no longer possible. Because of that, the counting protocol and the reset protocol can no longer be used to simulate Turing Machines.

In this section, we consider two cases: $f(n) = o(\log n)$ and a constant number of identifiers. We have a characterization when this number is constant: it leads to population protocols. In the general case, the population is more powerful, but we do not have any exact characterization.

Theorem 5.25 *Let f such that for some $k \in \mathbb{N}$, we have, for all n, $f(n) = k$.*

The set of functions computable by homonym population protocols with $f(n)$ identifiers corresponds to the semilinear sets over $\Sigma \times [0, k-1]$.

Proof: Each agent tries to put in its internal state the value of its identifier by using the following algorithm:

- If the identifier is 0, then the agent puts 0.
- If an agent knowing its identifier id meets an agent with identifier $Next(id)$, the second knows its identifier by incrementing id.

From this, we get that it is possible to compute semilinear predicates. Indeed, agents know now their initial state and can run the corresponding population protocol.

The proof that only semilinear sets can be computed is quite simple: we see a community protocol as a population protocol where the identifiers are directly included in the states. More precisely, $Q' = B \times [1, k] \times ([1, k] \cup \{\_\})^{d+1}$, and $\delta' = \delta$. □

Proposition 5.26 *There exists some non semilinear predicates over $\Sigma \times [1, k]$ and some function $f = o(\log n)$ such that there exists a homonym population protocols with $f(n)$ identifiers that computes this predicate.*

Proof: For example, it is possible to compute the predicate asking if the agents with an even identifier are in majority.

To compute this, it suffices to determine for each agent if its identifier is even or odd. The way to compute it is as follows:

- If the identifier is 0, then the agent remembers its identifier is odd.
- If an agent knowing its parity meets an agent with an identifier that is the directly next one, the second knows its parity by switching the other's one.

We then run the protocol $[x_{\text{even identifier}} > x_{\text{odd identifier}}]$. □

Remark 5.27 *Another counter-example is that we can compute the following predicate*

$$\left[\sum_{id \leq id_{max}/2} a_i x_{s_i,id} - \sum_{id > id_{max}/2} b_i x_{s_i,id} \geq c \right]$$

where $x_{s_i,id}$ is the number of agents with input s_i and identifier id. This predicate corresponds to a threshold predicate when we take into consideration whether the identifier of the agent is in the first or second half of the present ones.

This protocol computes the threshold predicate with first value b_i for each agent with non 0 identifier and input s_i, and a_i for the agents with identifier 0 and input s_i.

To compute the medium identifier, we have $d = 2$. To an agent $q_{i,j,k}$, we have i: agent's own identifier, j: medium candidate, k: $2j$ or $2j+1$. At the beginning, if the identifier is 0, we have $q_{0,0,0}$, else we get $q_{i,\_,\_}$. The state q means "I need to increment k if $k+1$ is present". The state q^{++} means "I need to increment both j and k if $k+1$ is present". Hence, each time we increment twice k, we increment once k.

The rules are:

$$\begin{array}{llcll}
q_{i,\_,\_} & q'_{0,j',k'} & \rightarrow & q_{i,0,0} & q'_{0,j',k'} \\
q_{i,j,k} & q'_{k+1,j',k'} & \rightarrow & q^{++}_{i,j,k+1} & q'_{k+1,j',k'} \\
q^{++}_{i,j,k} & q'_{k+1,j',k'} & \rightarrow & q^{+}_{i,j,k+1} & q'_{k+1,j',k'} \\
q^{+}_{i,j,k} & q'_{j+1,j',k'} & \rightarrow & q_{i,j+1,k} & q'_{j+1,j',k'}
\end{array}$$

As soon as an agent in input s_i realizes its identifier is smaller or equal to its j, it adds $a_i - b_i$ to its state if possible (else, it waits an occasion to add it to another agent).

By fairness, all agents will know at some point if their identifier was greater or smaller to half the highest one, and then the leader will be able to compute the right output.

5.4 Conclusion

With the homonym population protocols, we opened a landscape of submodels depending on the ratio of identifiers present in the population. The characterizations obtained in this chapter for the cases $f(n) = O(1)$, $f(n) = \Theta(\log^r n)$ and $f(n) = \Theta(n^\epsilon)$ are summarized in Table 5.1.

In particular, two tools were important for these results. The first is the capacity to store the size of the input cleverly enough to simulate a Turing Machine with $\log n$ identifiers. The second one is the use of tuples.

We introduced a strong hypothesis about the rules in this chapter: δ uses is able to know if two identifiers are consecutive. This is used in the cases $f(n) = \Theta(n^\epsilon)$ and $f(n) = o(\log n)$. I believe that without this hypothesis, the capacity to work with the size of the population will be lost. Recall that this hypothesis is actually not necessary for the case $f(n) = \Theta(n^\epsilon)$.

Finally, we proved that with $f(n) = o(\log n)$, the model is stronger than population protocols. We did not find any characterization neither any clue of what the computational power is in this case. The two protocols in Proposition 5.26 and Remark 5.27 show that we can at least do semilinear predicates taking into consideration modulo of the identifier and/or rational bound of the identifier depending on the highest one present in the population. We can even see in Remark 5.27l that the d identifier slots can be cleverly used.

Theorem 1.46 makes me believe that we cannot compute a lot more than semilinear sets, as a space $o(\log \log n)$ is enough to simulate $o(\log n)$ identifiers.

Chapter 6

Various Other Results

In the two previous chapters, we worked with agents using ordered identifiers as tools to compute functions. In this chapter, we gather results when we provide other tools. We start with the addition of Turing Machines to agents. This corresponds to the Passively Mobile Protocols [CMN+11]. Results from Chapter 5 permit to fill a gap.

We then consider a small variant of Community Protocols. What happens if the identifiers can no longer be ordered. We show that the model does not loose any space of computation. It is actually only the input that is weakened, as it has to be symmetric.

Finally, we end with a new variant. Instead of adding Turing Machines to agents, each one now contains a stack. We provide a protocol that simulate any Turing Machine, permitting to have an exact characterization of what can be computed.

6.1 Passively Mobile Protocols

We now show how constructions in Chapter 5 improve the results about the model from [CMN+11]. This section treats the case $S(n) = O(\log\log n)$ in the passively mobile protocol model. (a recap of the Hierarchy is in Table 6.1 bellow). $PMSPACE$ has been defined in Definition 1.43.

Theorem 6.1 *$PMSPACE(\log\log n) = \bigcup_{k\geq 1} SNSPACE(log^k n)$.*

Proof: **1.** $\bigcup_{k\geq 1} SNSPACE(log^k n) \subset PMSPACE(\log\log n)$.

The idea of this proof is quite simple: Let $M \in SNSPACE(log^k n)$. We can notice that $SNSPACE(log^k n) \subset MNSPACE(log^k n, \log n)$ (as the space of computation is the same and symmetric is equivalent to be a single multiset). From Theorem 5.22, there is a population protocol computing M. We will simulate it. With space $O(\log\log n)$, we can simulate a population protocol with $O(2^{\log\log n}) = O(\log n)$ identifiers.

Indeed, to create $\log n$ identifiers, we adapt a bit the counting protocol. At the beginning, each agent has identifier 0. When two agents with the same identifier meet, if each one contains the integer 1, the first switch its integer to 0, the other increases its own identifier.

Space per agent $S(n)$	Computational power
$O(1)$	Semilinear Sets [AAER07, AAD+04]
$o(\log\log n)$	Semilinear Sets [CMN+11]
$\Theta(\log\log n)$	$\bigcup_{k\in\mathbb{N}} SNSPACE(\log^k n)$ Theorem 6.1
$\Omega(\log n)$	$SNSPACE(nS(n))$ [CMN+11]

Table 6.1: Passively mobile protocols [CMN+11] with n agents and space $S(n)$ per agent.

We then just need to simulate the behavior of each agent as if they have started with their created identifier. It requires a space of size $|B| + (d+1)\log\log n$ plus some constant, which is enough.

2. $PMSPACE(\log\log n) \subset \bigcup_{k\geq 1} SNSPACE(log^k n)$: The proof is similar to the one of Theorem 5.21: It is a question of accessibility in the configurations graph. We need to compute the number N of possible configurations.

For each agent, there are $|Q|$ possible states and 4 tapes of length $\alpha\log\log n$ for some α. Hence, there are $|Q| \times |\Gamma|^{4\alpha\log\log n}$ possible states for each agent.

Now $|\Gamma|^{4\alpha\log\log n} = |\Gamma|^{\log(\log^{4\alpha} n)} = \left(\log^{4\alpha} n\right)^{\log|\Gamma|}$

For each possible state, there are at most n agents sharing it.

Hence, $N = O\left(n^{|Q|\times\left(\log^{4\alpha} n\right)^{\log|\Gamma|}}\right)$.

The accessibility can be computed by a machine in space complexity $O(\log N)$, which means a space $O\left(|Q| \times \left(\log^{4\alpha} n\right)^{\log|\Gamma|} \log n\right) = O(\log^k n)$ for some $k \in \mathbb{N}$. □

With a similar proof, we can get the following result that gives a good clue for the gap between $\log\log n$ and $\log n$:

Corollary 6.2 *Let f such that $f(n) = \Omega(\log\log n)$ and $f(n) = o(\log n)$.*

$$SNSPACE(2^{f(n)}f(n)) \subset PMSPACE(f(n)) \subset SNSPACE(2^{f(n)}\log n).$$

6.2 Community Protocols with Unordered Identifiers

We discuss in this section whether assuming that identifiers can not be compared is a restriction: Sometimes, it is not possible to have an order between identifiers (for example if they are symbols). We mostly prove this is not.

The difference in this model with community protocol's model is that we no longer have ordered stability over the rules (rules just need to be stable under permutations over the identifiers).

To prove our result, we first start by stating that it is possible to organize agents in a chain. The idea will then to use this chain to simulate community protocols. Notice that the chain is not this time used to simulate the tape of a Turing machine, but to provide an order that is used through Schonhage machines [Sch80] to simulate a Turing machine (via the results of [GR09]). Observe in particular that a direct use of the chain as a tape would not yield to $SNSPACE(n \log n)$ but something closer to $SNSPACE(n)$. We can see n different identifiers as a space $\log n$, since we can have between n and $2n$ identifiers with $\log n$ bits.

6.2.1 Leader Election and Chaining

Definition 6.3 *The* **Chaining Problem** *is to find a protocol such that eventually*

- *all agents form a chain corresponding to a doubly linked list : each agent has two slots* Next *and* Previous *containing an identifier. All these slots will refer to the structure of the chain.*
- *for each agent, its* Next *(resp.* Previous*) slot points to its successor (resp. its predecessor).*
- *Only one agent has no predecessor (its Previous slot is equal to empty) : it is therefore the first element, or the* **Head***, of the chain.*
- *Only one agent has no successor : it is the last element, or* **Tail***, of the chain.*
- *Each identifier appears in exactly one $Next$ slot except for the head and in exactly one $Previous$ slot except for the tail.*

Proposition 6.4 *There exists a protocol that solves the Chaining Problem.*

Proof: We will sketch a protocol building a chain among all the agents. The main idea of this protocol is to concatenate any doubly linked list at end of any another doubly linked list in a distributed manner. Each agent knows some information about the chain's structure: its memory contains the identifier of the head, the tail, its successor and its predecessor. Moreover, the head agent of the chain handles the concatenation process and updates all the information when the chain's structure changes.

To do so, at the beginning, each agent corresponds to a chain with one element. A concatenation process starts when two head agents meet. A leader election is performed in order to decide which chain to append to the end of the other list : one of them will become an internal agent of the new chain and its chain will be added at the tail of the winning head's chain. Without loss of generality, we assume that chain C will be at end of chain A.

During the interaction of the two heads, the head of chain C updates the local state by setting its $Previous$ slot to the $Tail$ slot of the head of chain A. In the same way, the head of chain A updates its $Tail$ slot to the $Tail$ slot of the head of chain C corresponding to the end of the new chain.

From now on, only the head of chain A (corresponding to the head of the new chain) scans the new list in order to update the chain's information (new head, and new tail). Moreover when it meets the tail of chain A, it notifies this agent about who is its successor. To perform this, each head needs to store the previous head of the second chain.

This scan can be performed by fairness assumption : after each updating, the head stores which is the next element of the list. The fairness assumption makes possible the interaction between the head and the next element. The fact that the head only updates allows it to detect the end of the updating process in order to be ready to start a new concatenation. Note that during this scan, the head of the chain cannot take part in another concatenation.

□

6.2.2 Computing SNSPACE($n \log n$)

Proposition 6.5 *Let g be a function of $SNSPACE(n \log n)$. There exists a protocol that computes g.*

Proof: Let g be a function of $SNSPACE(n \log n)$.

We have $SNSPACE(n \log n) \subset NSPACE(n \log n)$. Theorem 1.40 states that g is computed by a community population protocol P: this protocol uses an order on the identifiers and all agents have a unique identifier and d slots (each slot can store one identifier).

Our protocol can be split into two parts. The first part is to introduce an order on the identifiers in the population of agents and to elect a leader. To do this, a chain is built using a protocol that solves the Chaining Problem (see Proposition 6.4) : this order on the identifiers corresponds to the relative position of the agents in the chain (the closest to the leader we are, the smaller we are). The leader is the head of the chain.

When the leader believes that the chaining protocol is over, it restarts its second part (by reseting each agent in the chain).

The second part is to simulate the community protocol by selecting two agents using the previous order. To do this, the leader handles each interaction between two agents for the community protocol P. To do so, we will describe how the leader handles each interaction of the community protocol by a 3-stage process. The aim is to finds two agents, to get the order between the identifiers, and to produce the new states from it.

More specifically:

1. First, the leader chooses the two agents A and B that will interact and store there local state. These two agents are the next two agents that meet the leader. Note that if the leader consecutively meets the same agent, it decides that the interaction in the simulated community protocol is between itself and the previous agent (it permits the leader to be part of interactions). At each interaction, the leader stores all identifier slots and the local state of the two selected agents.

2. Second, from the first stage, the leader has received two lists of $d+1$ identifiers (one per agent). Now, it sorts each identifiers appearing in the lists according to the identifier order. To perform this, it scans the chain from the head until the tail.

3. Finally, the leader locally performs the interaction between agents A and B by applying the transition function of P. Afterwards, the leader looks for the two agents who interact in order to update there local state.

By repeating this process again and again, our protocol can simulate the community population protocol.

To any sequence of this protocol we can associate a sequence of protocol P: Once the chaining protocol is over, each time the leader selects two agents A and B, our protocol acts as if A and B were the interacting agents in protocol P. If a sequence of this protocol is fair, then a sequence of protocol P is also fair.

Since the function computed by protocol P is stable under permutation of the input (it is in $SNSPACE$), the output will always be the same, whatever the order induced by the chain is. The created protocol computes g, which concludes this proof. □

Theorem 6.6 *The set of functions that can be computed by a community protocol where identifiers cannot be compared is exactly $SNSPACE(n \log n)$.*

Proof: The first inclusion is proven by Proposition 6.5.

The second inclusion uses the fact that a protocol with unordered identifiers is a community protocol (as the rules restriction is compatible to community protocol's rules restrictions). Hence, the computed function must be in $NSPACE(n \log n)$. This function must be also stable under permutation, as the identifiers cannot be ordered (it implies that input cannot be ordered neither).

Hence, we get that the computed function is in the subset of $NSPACE(n \log n)$ stable under permutation of the input, that is to say $SNSPACE(n \log n)$. □

6.2.3 With $\sqrt[k]{n}$ Incomparable Identifiers

Theorem 6.7 *The set of functions that can be computed by a protocol with $\sqrt[k]{n}$ incomparable identifiers is exactly $SMNSPACE(n \log n, \sqrt[k]{n})$.*

Proof: The idea is again to use a k-tuple of identifiers as identifiers, in a spirit of what has been done in Proposition 5.19. As identifiers cannot be compared, the multisets in the input cannot be ordered. The language must remain stable under permutation of the multisets. □

6.3 Population Protocols with a Stack

6.3.1 Model

We now consider a variant of the population protocol model where agents have stacks (i.e. we work with stack automata instead of automata). This time, each agent can store a stack of tokens. At the beginning, this stack is empty. Each agent knows if its stack is empty or not. If a stack is not empty, then the agent knows what is the top element of its stack.

During an interaction, agents can remove the top element of the stack and/or add a new one.

More formally, here is the definition of Population Protocols with a stack:

Definition 6.8 *From the original model, we add the stack alphabet T, and δ is modified to handle the top elements of the stack.*

A **Population Protocol with a Stack** *is given by seven elements $(Q, T, \Sigma, \iota, Y, \omega, \delta)$ where:*

- *B is a finite set of states.*
- *T is a finite set of tokens.*
- *Σ is the finite set of entry symbols.*
- *ι is a function $\Sigma \to B$.*
- *Y is the finite set of possible outputs.*
- *ω is a function $B \to Y$.*
- *δ is a function $Q^2 \to (B \times (T \cup \{0, -1\}))^2$.*

Each agent has its state in $B \times T^*$. An initial state of an agent with input $s \in \Sigma$ is $\iota(s)$, with an empty stack. Hence, a configuration is a multiset over the infinite set $B \times T^*$. During an interaction, agents do not have access to their full state: they do not know the full content of their stacks. $Q = B \times (T \cup \{\perp\})$ corresponds to the visible state: Agents only know their state in B and if their stack is empty or not. If their stack is not empty, then they know what is the top element of the stack.

If an agent is in state $(b, u) \in B \times T^*$, then the visible state is:

- *$(b, \perp)$ if $u = \epsilon$*
- *(b, u_0) when $u = u_0 \dots u_k$.*

We use the following notation: $(q_1, t_1), (q_2, t_2) \to (q'_1, t'_1), (q'_2, t'_2)$ where $t'_1 \in T \cup \{-1, 0\}$: such a rule means that first agent switches its state to q'_1. If $t'_1 \in T$, the agent puts on top of its stack the token t_1, else it puts nothing. If $t'_1 = -1$, then it removes the top token. If $t'_1 = 0$, the agent does nothing. The second agent's update works the same way.

The interpretation of agents does not depend on the tokens in the stack.

Any Turing Machine can be simulated by an automaton with 2 stacks [Min67]. Since we got as many stacks as agents, it seems intuitive to be able to simulate a Turing Machine with this model, as we have way more stacks than 2. The important thing here is to create carefully a protocol.

The inputs must be stable under permutation, as their is no orders among agents.

Theorem 6.9 *Any Deterministic Symmetric Turing Machine can be simulated by a Population Protocol with a Stack.*

The simulation protocol is separated in several sub-protocols. Here is a quick explanation of each protocol's idea:

1. **Collect of Inputs**: Each agent starts in the state L with its input placed in its stack as a token. A leader election is performed. At the same time, the leader gathers all the inputs in the form of its tokens: At some point, there will be a single agent storing all the inputs on its stack, each other agents being in state N with an empty stack.

2. **Stack Duplication**: The Leader duplicates its stack: It looks for two agents to perform that (that will be named Co and Pa, the leader becoming Cu). Each time it removes a token, it puts the same in the two other agents. Co will become the new Leader L_c, the second one, called the simulating agent M, will perform the simulation.

3. **Turing Machine Simulation**: The simulating agent M looks for a second agent, and they get the names M_1 and M_2. As they both have a stack, they will be able to simulate the Turing Machine. M_1's stack will represent the left part of the tape, M_2's one simulates the right.

4. **Reset Protocol**: As it may have several simulations running at the same time, the Reset Protocol is here to detect it and restart a run. When two Leaders L_c meet, both knowing that a simulation is performing, both of them finds 2 simulating agents to ask them to stop. Then the leaders merge their stack using Step 1.

5. **Output Transmission**: When the simulation is over, M_1 looks for each agent and gives them the output, which corresponds to the second element in their states.

More specifically: let $M = (Q_M, \Gamma, b, \Sigma, \delta_M, q_0, F)$ be a symmetric Turing Machine. We build the following protocol:

- $B = (\Sigma \cup B_1 \cup B_2 \cup B_3 \cup B_4) \times \{0, 1\}$. The set B_i is explained in the section corresponding to the ith step.
- $T = \Gamma$.
- Σ is the same.
- $\forall s \in \Sigma$, $\iota(s) = s$.
- $\omega(a, b) = True$ if and only if $b = 1$.
- δ is explained through the descriptions of each steps.

To simplify the transitions, the token an agent state is written only if it is relevant (otherwise, we write _). The output part of the agents appears only when it is relevant: in Step 5.

6.3.2 Collect of Inputs

$$B_1 = \{L, N, R, G\}$$

$$\begin{array}{rlll}
(s,\perp) & (b,t_2) \rightarrow (L,s) & (b,0) & \forall s\in\Sigma,\ \forall b\in B,\ \forall t_2\in T\\
(L,\_) & (L,t_2) \rightarrow (R,t_2) & (G,-1) & \\
(R,\_) & (G,t_2) \rightarrow (R,t_2) & (G,-1) & \\
(R,\_) & (G,\perp) \rightarrow (L,t_2) & (N,0) &
\end{array}$$

The first rule is here to start the protocol: we want each agent to start by putting on their empty stack their input. At that moment, they are leaders.

Each Leader agent stores some of the inputs. When two leaders meet, one goes to state G and empties its stack and "give" the tokens to the other one (who goes into state R) until G's stack gets empty.

Here are two invariants over each C_i starting from a same input of size n:

$$\forall s \in \Sigma, \text{ the number of tokens } s \text{ in agents in state in } \{L, R, G\} + |(s,\perp)| = n \tag{6.1}$$

$$|R| - |G| = 0 \tag{6.2}$$

By fairness, we can always reach a state where each agent are in state L and N, by using repeatedly the third and the fourth rule (as we always have as much G than R). If we have at least two L agents, we can remove definitively one by using the second rule. and use the empty process.

Then, by fairness, we will eventually reach a point where a single agent stores the whole input on its stack, each other agent being in state N with an empty stack. As we cannot know when this point is reached, the four other steps contain rules to reset and come back to this point when we discover the presence of two leaders.

6.3.3 Stack Duplication

$$B_2 = \{Co, Co^i, Cu^i, Cu^o, Cu^{Pa}, Cu^{Co}, L_c\}$$

$$\begin{array}{rrrr}
(L,\_) & (N,\_) \rightarrow & (Cu^i,0) & (Co^i,0)\\
(Cu^i,\_) & (Co^i,\_) \rightarrow & (L,0) & (N,0)\\
(Cu^i,\_) & (N,\_) \rightarrow & (Cu^o,0) & (Pa,0)\\
(Cu^o,t_1) & (Co^i,\_) \rightarrow & (Cu^{Pa},0) & (Co,t_1)\\
(Cu^{Co},t_1) & (Co,\_) \rightarrow & (Cu^{Pa},0) & (Co,t_1)\\
(Cu^{Pa},t_1) & (Pa,\_) \rightarrow & (Cu^{Co},-1) & (Pa,t_1)\\
(Cu^{Co},\perp) & (Co,\_) \rightarrow & (Cu^{Pa},0) & (L_c,0)\\
(Cu^{Pa},\perp) & (Pa,\_) \rightarrow & (N,0) & (M,0)
\end{array}$$

We change the first invariant (6.1) into:

$$\forall s \in \Sigma, \text{ the number of tokens } s \text{ in agents in state in } \{L, R, G, Cu^X, Co^X, L_c\} + |(s, \bot)| = n \tag{6.1'}$$

We also have four new invariants:

$$|Cu^i| + |Cu^o| - |Co^i| = 0 \tag{6.3}$$

$$|Cu^{Co}| + |Cu^{Pa}| - |Co| = 0 \tag{6.4}$$

$$|L_c| - |M| = 0 \tag{6.5}$$

$$|Cu^o| + |Cu^{Pa}| + |\{(Cu^{Co}, t), t \in T\}| - |Pa| = 0 \tag{6.6}$$

We will only prove now that, if we are in a configuration with one L and all other agents are in state N, we will reach by fairness a configuration with one agent in state L_c (leader copied), one agent in state M containing all the inputs in its stack and all other agents in state N.

From LN^{n-1}, only the first rule apply, then the third will eventually occur by fairness (the only other rule that can be applied is the second that is here to handle the case where all agents are in state Cu and Co, and this rule cancel the first one).

From the third rule, the population will only be able to perform rule 4. After that, rules 5 and 6 will be performed alternately until the stack of Cu (for Cut) gets emptied. At that point, the last two rules will finish the process.

Each time a token is removed from Cu, the same token is added to Co (for Copy) and to Pa (for Paste). With this, we can be sure that at the end of the process, the stack from L has been duplicated on L_c and M. The first agent is here to keep the stack in case a new leader is detected, the second one will start a computation with the next step.

6.3.4 Turing Machine Simulation

$$B_3 = (\{M_1, M_1^r\} \times Q_M) \cup (\{M_1^r, M_1^c\} \times Q_M \times \Gamma) \cup \{M_2\}$$

$$\begin{array}{rlrll}
(M, \_)\ (N, \_) & \rightarrow & ((M_1, q_0), 0)\ (M_2, 0) & & \\
((M_1, q), t_1)\ (M_2, \_) & \rightarrow & ((M_1, q'), -1)\ (M_2, t_1') & & \text{with } \delta_M(q, t_1) = (q', t_1', -1) \\
((M_1, q), t_1)\ (M_2, \_) & \rightarrow & ((M_1^c, q', t_1'), -1)\ (M_2, 0) & & \text{with } \delta_M(q, t_1) = (q', t_1', 0) \\
((M_1^c, q', t_1'), \_)\ (M_2, \_) & \rightarrow & ((M_1, q'), t_1')\ (M_2, 0) & & \\
((M_1, q), t_1)\ (M_2, \_) & \rightarrow & ((M_1^r, q', t_1'), -1)\ (M_2, 0) & & \text{with } \delta_M(q, t_1) = (q', t_1', +1) \\
((M_1^r, q', t_1'), \_)\ (M_2, \_) & \rightarrow & ((M_1^r, q'), t_1')\ (M_2, 0) & & \\
((M_1^r, q'), \_)\ (M_2, t_2) & \rightarrow & ((M_1, q'), t_2)\ (M_2, -1) & &
\end{array}$$

The new invariant here is:

$$|M_1| - |M_2| = 0 \tag{6.7}$$

The second invariant (6.5) in the previous section becomes:

$$|L_c| - |M| - |M_1| = 0 \tag{6.5'}$$

An agent in state M looks for a N to start the simulation. $((M_1, q), t)$ will contain the left part of the tape (i.e. what is there on the left of the head of the tape) and the current state q of the simulated Turing Machine and the read element symbol t, M_2 contains the right part.

If we have $\delta(q,t) = (q', t', \epsilon)$, rule 2 handles the case $\epsilon = -1$ by removing the top element and putting t' at the beginning of M_2. Rule 3 and 4 handles $\epsilon = 0$ by removing the top token remembering t', then adding it at the end of the stack of M_1. Finally, rule 5, 6 and 7 treats $\epsilon = +1$ by removing the top element, putting t' then removing the top element of M_2 to add it at the beginning of M_1.

Once the simulation has reached a state $f \in F$, M_1 will spread the output to each agent through the rules of the next step.

6.3.5 Reset Protocol

$$B_4 = \{L_d, L_{d2}, S\}$$

$$\begin{array}{rlll}
(L_c, \_)\ (L_c, \_) & \rightarrow (L_d, 0)\ (L_d, 0) & & \\
(L, \_)\ (L_c, \_) & \rightarrow (L, 0)\ (L_d, 0) & & \\
(L_d, \_)\ (M, \_) & \rightarrow (L, 0)\ (S, 0) & & \\
(L_d, \_)\ ((M_1, \_), \_) & \rightarrow (L_{d2}, 0)\ (S, 0) & & \\
(L_{d2}, \_)\ ((M_2, \_), \_) & \rightarrow (L, 0)\ (S, 0) & & \\
(S, t_1)\ (b, t_2) & \rightarrow (S, -1)\ (b, 0) & \forall s \in \Sigma,\ \forall b \in B,\ \forall t_1, t_2 \in T \\
(S, 0)\ (b, t_2) & \rightarrow (N, 0)\ (b, 0) & \forall s \in \Sigma,\ \forall b \in B,\ \forall t_1, t_2 \in T
\end{array}$$

(6.1') evolves again:

$$\forall s \in \Sigma, \text{ the number of tokens } s \text{ in agents } \{L, R, G, Cu^X, Co^X, L_c, L_d, L_{d2}\} + |(s, \perp)| = n \tag{6.1$^{(2)}$}$$

We get also an update of two other invariants:

$$|L_c| + |L_d| - |M| - |M_1| = 0 \tag{6.5$^{(2)}$}$$

$$|M_1| + |L_{d2}| - |M_2| = 0 \tag{6.7'}$$

When two leaders meet, with one having a stack copied, the leaders with a copied stack look for a M or a M_1 to stop the computation. If the leader meets a M_1, it looks also for a M_2 before going back to state L. The Ms goes into stack suppression mode with the state S.

6.3.6 Output Transmission

$$\begin{array}{llll}
((M_1, f), \_) & ((N, \_), \_) & \rightarrow (((M_1, f), 1), 0) & ((N, 1), 0) & f \in F_1 \\
((M_1, f), \_) & ((L_c, \_), \_) & \rightarrow (((M_1, f), 1), 0) & ((L_c, 1), 0) & f \in F_1 \\
((M_1, f), \_) & ((M_2, \_), \_) & \rightarrow (((M_1, f), 1), 0) & ((M_2, 1), 0) & f \in F_1 \\
((M_1, f), \_) & ((N, \_), \_) & \rightarrow ((M_1, f), 0), 0) & ((N, 1), 0) & f \in F_0 \\
((M_1, f), \_) & ((L_c, \_), \_) & \rightarrow ((M_1, f), 0), 0) & ((L_c, 1), 0) & f \in F_0 \\
((M_1, f), \_) & ((M_2, \_), \_) & \rightarrow ((M_1, f), 0), 0) & ((M_2, 1), 0) & f \in F_0
\end{array}$$

This part is quite simple: when a tape has finished its computation, it looks for each agent to give it the output. When there is only one leader, M_1 and M_2 have computed the right output, and will provide the right output to all of the agents (the population have the configuration $L_c M_1 M_2 N^{n-3}$).

6.3.7 Correctness of the Protocol

Lemma 6.10 *Let $(C_i)_{i \in \mathbb{N}}$ be a fair execution over an input of size n. There exists some integer $I \in \mathbb{N}$ such as C_I is composed by:*

- *a leader L, with a stack containing exactly a multiset corresponding to the input.*
- *The other agents in state N with an empty stack or in state S.*

Proof: We can first notice that $|\{L, L_c, L_d, Cu^i, Cu^o, Cu^{Pa}, Cu^{Co}, R\} \cup \Sigma|$ can only decrease, and will always be strictly positive. This value converges. Let's prove by contradiction that it will converge to 1:

Let's suppose their exist some configuration C appearing infinitely many often such as $|\{L, L_c, L_d, Cu^i, Cu^o, Cu^{Pa}, Cu^{Co}, R\} \cup \Sigma| > 2$. Let's prove we can reach a configuration with two elements in $\{L, L_c\}$.

- If there exists a $s \in \Sigma$, by applying the first rule with any agent, we get a L.
- If there exists a Cu^i, there must be a Co^i (by invariant (6.3)). The second rule in 6.3.3 permits to obtain a L.
- If there exists a Cu^o, there must be a Co^i and a Pa (invariants (6.3) and (6.6)). By applying rule 4 in 6.3.3, then repeating rules 5 and 6 until Cu's stack get empty, and then applying 7, we get a L_c.
- If there exists a Cu^{Pa} or a Cu^{Co}, there must be a Co and a Pa (by (6.4) and (6.6)). We repeat rules 5 and 6 then 7 of 6.3.3 to get a L_c.
- If we have a L_d, from ($6.5^{(2)}$), there must be a M or a M_1 in the population. We apply rules 3 or 4 of Section 6.3.5 to get a L.

- If there exists a R, by (6.2), there is a G. We apply again and again rule 3 of 6.3.2 until G's stack gets empty, then we apply the fourth to get a L.

So now we have two elements in $\{L, L_c\}$. If we have 2 L, second rule of 6.3.2 and then the process written just above permits to remove one. Else, the Reset step permits to transform any L_c into a L.

We can hence always make decrease $|\{L, L_c, L_d, Cu^i, Cu^o, Cu^{Pa}, Cu^{Co}, R\} \cup \Sigma|$, so by fairness it must happen.

It proves the fact that if a configuration appears infinitely many often, it is such that $|\{L, L_c, L_d, Cu^i, Cu^o, Cu^{Pa}, Cu^{Co}, R\}| = 1$.

There exist some I such as, in C_I, $|\{L, L_c, L_d, Cu^i, Cu^o, Cu^{Pa}, Cu^{Co}, R\} \cup \Sigma| = 1$. We choose I minimal.

The only rule that can have permitted to reach C_I is the second of Section 6.3.2. From this, we can be sure that there is a L in the configuration.
We can deduce that $|\{L_c, L_d, Cu^i, Cu^o, Cu^{Pa}, Cu^{Co}, R\} \cup \Sigma = 0|$. The invariants permit from this to get that the only possible states for the other agents are N and S.

By invariant ($6.1^{(2)}$), all the input symbols and only them are in the stack of the last L.
$\square$

The proof of the correctness here is now simply to show that the rules, from C_I, ensure that the population will simulate as wanted the Turing Machine.

This can be verified from the deterministic aspect of the behavior of the population from configuration C_I:

The Ss will empty their stack by fairness. At some point, L will find a N and another one before meeting again the first one. From this, the stack duplication will be performed.

M will then look for a N to turn it into a M_2, and then starts the Machine simulation, that cannot be interfered, as they will not be other Ms in the population.

When the simulation will be over, by fairness, M_1 will provide the right output to each other agents (i.e. M_2, L_c and all the Ns).

This configuration is stable, and has the right interpretation.

This concludes the proof of the protocol.

6.3.8 Non Deterministic Turing Machines

Actually, we have an exact characterization of what can be computed:

Theorem 6.11 *Population protocols with a stack compute exactly the set of languages stable under permutation computable by non deterministic Turing Machines.*

Proof: **Compute any NDTM:** Previous construction simulates any Deterministic Turing Machine. We need to adapt it a bit to add the non determinism.

To handle the non deterministic choice, as in other proofs, it suffices to use the symmetric rule for the second choice. To the pairs of the form $(M_1 \ldots)$ $(M_2 \ldots)$ in the Turing Machine Simulation (Section 6.3.4), the simulation performs the first possible choice. To the pairs of the form of the form $(M_2 \ldots)$ $(M_1 \ldots)$ the simulation performs the second one.

As long as the simulation rejects the input, when the simulation is over, the protocol duplicates a new time the input and tries a new simulation. As long as simulation is running M_1 spreads the output $False$. As soon as a simulation is accepting, M_1 carries the output $True$ to all the agents.

NDTM simulates: The proof is the same as each time we use a non deterministic Machine to simulate a protocol (see Remark 1.41 and Proposition 5.21 for example). Here, as we do not limit the space of computation, it is even easier. □

6.4 Conclusion

We filled the gap in Passively Mobile Protocols, characterizing what can be exactly computed by agents carrying Turing Machines with a space $O(\log\log n)$.

We then considered a quick variation of the community protocols where identifiers are no longer ordered. Even with some homonymy ($O(n^\epsilon)$ identifiers), the model still simulates Turing Machines with tapes of size $O(n\log n)$.

Remains the question of what can be computed if we have $\Theta(\log n)$ different identifiers. The protocols provided in Chapter 5 cannot work if the identifiers are unordered.

Finally, we proved that the addition of stacks is as powerful as it can be expected: any nondeterministic Turing Machines can be simulated.

Part IV

Weakening of Turing Machines: the Complexity of Transitions

Chapter 7

Rusted Turing Machines

This chapter is about ***Rusted Turing Machines*** (RTM). We introduce the notion of *pivotal transition*, that is transitions that change the internal state. We study what can be computed with a finite number of pivotal transitions, whatever the size of the input is.

We provide several computable languages, including semilinear sets, regular languages, context-free languages... We provide a comparison with the $CPPOLYLOG$ class. We then prove that the halting problem is decidable for Rusted Turing Machines and provide an upper bound of the busy beaver problem of the model.

7.1 The Rusted Turing Machine Model

The study of the class $CPPOLYLOG$ (see Chapter 4) and the research of a characterization led us to the consideration of the following model. The motivation comes directly from the Section 4.4.3, where we tried to compare the model to some kind of Turing Machine.

The description was not as precise at this time, and the following question came: What can compute a Turing Machine that can change its own state a polylogarithmic number of times? We can notice, from the items of Theorem 4.25, that only items 3 and 4 looked to be difficult.

We managed to find a Machine that write the size of the input (or any subset) in a logarithmic number of steps... And even in a constant number of steps. At this point, this kind of Machine looked to be more powerful. And finally, we realized than in only 2 changes of internal state of a Machine, it is possible to compute the language $(ab)^n$. At this point, it looked obvious that our two models were different, but it was enough to motivate the question of this section: What can be computed by a Turing Machine that can change only a constant number times its internal state, whatever the size of the input is?

Turing Machines with a single state has been studied in [Her69, Sao95]. These works are close, as we can see our model as a finite succession of runs of one state Turing Machines.

We start this chapter with useful definitions and informations about the model. We then provide computable functions and sets. In particular, computing the size of the input, in a similar way that with *CPPOLYLOG* and Homonym Population Protocols. A parallel with the latter class is given. We then study a comparison with Rational Languages and Context-Free grammars. We finish with some decidability results, providing an upper bound of the Busy Beaver of the model.

Some of the studies in this chapter has been performed by Marc Chevalier, student of ENS Lyon, under my supervision during an L3 internship.

7.1.1 Definitions

Definition 7.1 *The* **Transition Graph** *of a Turing Machine $M = (Q, \Gamma, B, \Sigma, \delta, q_0, F)$ is the graph with vertexes corresponding to Q such that we have an edge $q_1 \to q_2$ if there exist some $s, s' \in \Sigma$ and some m such that $\delta(q_1, s) = (q_2, s', m)$.*

A Turing Machine on a single tape $M = (Q, \Gamma, B, \Sigma, \delta, q_0, F)$ is called a **Rusted Turing Machine** *if its transition graph is acyclic when we exclude loops.*

We call **RTM** *the class of these machines.*

Example 7.2 *The following machine is in RTM and computes the sorted language (see Definition 1.38):*

- $Q = \{q_i\}_{i \leq k} \cup \{q_\top, q_\bot\}$
- $\Gamma = \{B, s_1, s_2, \ldots, s_k\}$
- $\Sigma = \{s_1, s_2, \ldots, s_k\}$
- $F = \{q_\top\}$
- *δ works as follows (the rules not provided never happen):*

$$\begin{array}{ll} \delta(q_0, s_i) = (q_i, s_i, \longrightarrow) & \text{with } i \leq k \\ \delta(q_i, s_j) = (q_j, s_j, \longrightarrow) & \text{with } i \leq j \leq k \\ \delta(q_i, s_j) = (q_\bot, s_j, \texttt{HALT}) & \text{with } j < i \leq k \\ \delta(q_i, B) = (q_\top, B, \texttt{HALT}) & \text{with } i \leq k \end{array}$$

The transition graph for the case $\Sigma = \{s_1, s_2, s_3\}$ can be seen in Figure 7.1.

In the general case, the transition graph is indeed acyclic, as, for any $i \leq k$, q_i can only lead to some q_j with $j \geq i$, to $q_\top$ or to $q_\bot$ (those two last states leading to nowhere, as they are halting states). Hence each path of $q_{i_1} \to q_{i_1} \to \ldots \to q_{i_l}$ is such that $(i_j)_{j \leq l}$ is an increasing sequence.

The state q_i means "I now accept to see only symbols s_j, with $j \geq i$". Each time we see a s_i, starting from this point, we want to see only s_j with $j \geq i$. This is handled by the two first rules.

As soon as we see a contradiction (i.e. s_j with $j < i$), we reject and halt.

If we reach the symbol B, we did not see any problem with the input, we accept it and halt.

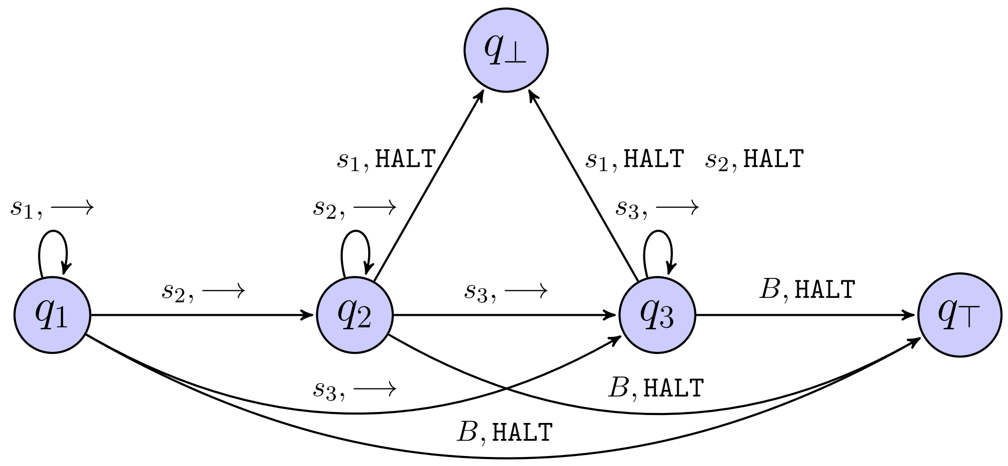

Figure 7.1: The acyclic graph for Example 7.2 with $\Sigma = \{s_1, s_2, s_3\}$

Definition 7.3 *A transition $\delta(q,a) = (q', a', m)$ is said* **Pivotal** *if $q \neq q'$.*

We denote $piv(M,x)$ the **Number of Pivotal Transitions** *performed by the Turing Machine M on the input word x.*

We denote $piv_M : \mathbb{N} \to \mathbb{N}$ the function such that $piv_M(n) = \max\limits_{x \in \Sigma^n} piv(M,x)$. We call this function the **Pivotal Complexity** *of the Turing Machine M. It is a function that associates to an integer n the maximum number of pivotal transitions of input of size n.*

We call $PIV(f(n))$ the class of Turing Machines of Pivotal Complexity in $O(f(n))$.

We provide now two quick results about this new complexity consideration. The first is here to motivate the works done in Section 7.3, by trying to improve the pivotal complexity. The second is the proof of equivalence of the RTM class with $PIV(1)$.

Proposition 7.4 *The set of rational languages is in $PIV(n)$.*

Proof: It suffices to translate the automaton recognizing the language into a Turing Machine. The machine always updates its state according to the rules of the automaton and goes right. Once the blank symbol B reached, it accepts if and only if the current state is accepting for the automaton. □

Proposition 7.5 *RTM corresponds exactly to $PIV(1)$.*

Proof: A machine in RTM has a Pivotal Complexity bounded by the size of its graph, as the transition graph is acyclic. Hence it has a constant Pivotal Complexity.

Let $M \in PIV(1)$. We then know there exists some k that bounds above the number of pivotal transitions, whatever the input is. Lets build the Machine M_k that simulates M and counts the number of pivotal transitions of M rejecting as soon as $k+1$ pivotal transitions happen.

M_k has only two differences with M:

- $Q_k = \{q_\bot\} \cup Q \times [1,k]$
- $\begin{cases} \delta_k((q,j),a) = ((q,j),a',m) & \text{with } \delta(q,a) = (q,a',m) \\ \delta_k((q,j),a) = ((q',j+1),a',m) & \text{with } \delta(q,a) = (q',a',m),\ q \neq q' \text{ and } j < k \\ \delta_k((q,k),a) = (q_\bot,a',m) & \text{with } \delta(q,a) = (q',a',m) \text{ and } q \neq q' \end{cases}$

This Machine is in RTM. The paths of the transition graphs correspond exactly to the paths of length at most k in M. M_k gives the same output that M for inputs of number of pivotal transitions bounded by k. As k bounds the number of pivotal transitions of each input, it then computes the same language that M.

Hence, M is in RTM. □

Remark 7.6 *We spent time to determine a function depending on the size of the alphabet and on the size of the transition graph that gives an upper bound of the pivotal complexity of machines in $PIV(1)$. We did not find any, and we even believe that in cannot be computed.*

7.1.2 Motivations of the Restrictions

We introduce here two results, the first to motivate the use of a single tape. The second one, extracted from Shannon in [Sha57], is here to show that using only 2 states is not enough to have a restrictive model.

Proposition 7.7 *There exists a Universal Turing Machine* with *two tapes of Pivotal Complexity 1.*

Proof: The proof is quite simple: A Universal Turing Machine can be simulated. Only one cell of the second tape is used: we write on it the current state of the simulated machine. At each transition, instead of switching the current state, we just change the cell of the second tape.

Our machines stop when the simulation also stops. Our machine will then provide the same output. □

Theorem 7.8 ([Sha57]) *There exists an universal Turing Machine on a single tape using two states.*

7.1.3 Notations and Suppositions

From now, we consider that the tape delimits the input with the symbol μ_L on its left and μ_R on its right. This is not an improvement of the power of the model, as it is possible to start each of our computations by adding these symbols and come back on the first letter of the input.

In the provided machines for some theorems and propositions, Q will always be of the form $\{q_i\}_{i\leq k} \cup \{q_\top, q_\bot\}$ for some $k \in \mathbb{N}$. Our machines will always stop in states $q_\top$ and $q_\bot$ in

Q. They correspond to the accepting and rejecting states. These states will never have rows in the table of δ, as they are halting states. $F = \{q_\top\}$ for all of our machines. The graphs will always be acyclic, as $\delta(q_i, a) = (q_j, a', m)$ implies that $i \leq j$, and the computation halts as soon as we enter in state $q \in \{q_\top, q_\bot\}$.

δ will be described by arrays as follows:

$s \in \Gamma$ \ $q \in Q$	q_0
a	q', a', m

The second row means that $\delta(q_0, a) = (q', a', m)$. The third element m will always take its value in $\{\longrightarrow, \longleftarrow, 0\}$.

An empty square in the matrix happens if the couple state/symbol cannot occur.

7.2 A Link with CPPOLYLOG

7.2.1 The Size of the Input

As with the *CPPOLYLOG* model, we found here a way to compute and encode the size of the population. We first found a method in $\log n$ pivotal transitions: Each time, we marked twice the number of cells previously marked, and we count how many time we did it.

After some more research, we finally found a way to compute the size with only a finite number of pivotal transitions. The idea is to simulate a counter on the left side of the tape. At the beginning each cell of the input is marked. Each time we go left, we increment the counter, each time we go right, we unmark one more cell. We repeat this process until we reach the right part of the input. More precisely:

Theorem 7.9 *There exists a machine in RTM that writes in binary on the left of the input the number of marked cells in the input. The smallest bit (corresponding to 2^0) is the right one.*

Proof: This time, as we will write the size on the left side of the input, there will not be the symbol μ_L just before the input. We want to count the number of marked cells (at the beginning, the whole input is marked).

A marked cell will have the symbol M, an unmarked one will be with symbol R or L. To ignore the unmarked cells, we will use the following rules (these rules will be inplicit in the next protocols each time we want to work only on marked cells):

$s \in \Gamma$ \ $q \in Q$	$\forall q \in \{q_0, q_1\}$
R	$q, L, \longrightarrow$
L	$q, R, \longleftarrow$

The management of the right part is simple: Each time the machine sees a marked cell, it unmarks it and turns it into state R, then goes left to find the left part of the tape. When we reach the end of the input μ_R, we go to state q_1.

$q \in Q$ $s \in \Gamma$	q_0
M	$q_0, R, \longleftarrow$
μ_R	$q_1, \mu_R, \longleftarrow$

The management of the left part remains is quite instinctive. The first digit read corresponds to 2^0. If we have 0 (a blank symbol B is seen as a 0), we increment it, if it is equal to 1, we write 1_R then go left and repeat until we see a 0. We switch this 0 to 1, then we turn to 0 each bits 1_R seen by walking back to the right. This process corresponds exactly to incrementing a binary counter.

More formally, here are the rules:

$q \in Q$ $s \in \Gamma$	q_0
B	$q_0, 1, \longleftarrow$
0	$q_0, 1, \longleftarrow$
1	$q_0, 1_R, \longrightarrow$
1_R	$q_0, 0, \longleftarrow$

Finally, to put back our head at the beginning of the input, we add this last rule: As soon as a digit is seen, the input starts just on the right cell. The machine then halts in state q_2.

$q \in Q$ $s \in \Gamma$	q_1
$\forall s \in \{L, R\}$	$q_2, s, \longleftarrow$
$\forall s \in \{0, 1\}$	$q_2, s, \longrightarrow$

$\square$

Corollary 7.10 *It is possible, for any $b \geq 2$, to write the number of marked cells in base b.*

Proof: We just need to adapt the counter by providing the following rules in the third matrix:

$q \in Q$ $s \in \Gamma$	q_0
B	$q_0, 1, \longrightarrow$
$\forall i < b - 1$	$q_0, i + 1, \longrightarrow$
$b - 1$	$q_0, (b - 1)_R, \longleftarrow$
$(b - 1)_R$	$q_0, 0, \longrightarrow$

$\square$

Remark 7.11 *This protocol actually enlightens how to perform 4 different things:*

- *How to increment or decrement a counter written in base b each time we go on it, without performing any pivotal transition.*

  Decrementing is symmetric. We use the following rules:

$s \in \Gamma$ \ $q \in Q$	q_0
B	$q_1, B, halt$
$\forall i > 0$	$q_0, i-1, \longrightarrow$
0	$q_0, 0_R, \longleftarrow$
0_R	$q_0, b-1, \longrightarrow$

- *How to increment or decrement a counter written in unary each time we go on it, without performing any pivotal transition.*

Notice that these counters can be used on the left or on the right (we just need to reverse the arrows to change from where we go to the counters and change the sense of lecture of the number).

7.2.2 Semilinear Sets

We prove here that our model is at least as strong as the classic Population Protocols, by proving that any semilinear set can be computed by a Rusted Turing Machine. To perform that, we create a Machine for each basic predicate and prove the boolean combination stability.

Proposition 7.12 *Any Threshold predicate can be computed by a RTM.*

As a reminder, a threshold predicate is of the form $[\sum a_i|x|_{s_i} \geq b]$, with $\Sigma = \{s_1, \ldots, s_k\}$ and $a_1, \ldots, a_k, b \in \mathbb{Z}^{k+1}$.

Proof: We first apply the following transformation to each agent: For each x_i such that $a_i \geq 0$, we replace x_i by (a_i, R), and for the other x_i, we replace their inputs by $(R, -a_i)$.

The computation uses then 3 steps:

1. Compute $\sum\limits_{a_i>0} a_i|x|_{s_i}$ in unary on the left of the input on the first element of a couple (i.e. B becomes $(1, B)$).

2. Compute $\sum\limits_{a_i<0} a_i|x|_{s_i}$ in unary on the left of the input on the second element of a couple (i.e. B becomes $(B, 1)$ and $(1, B)$ becomes $(1, 1)$).

3. We then check in b pivotal transitions if there is at least b $(1, B)$ after the last B on the left of the tape. If it is the case, we accept, else we reject.

$\square$

Proposition 7.13 *Any Modulo predicate can be computed by a RTM.*

As a reminder, a modulo predicate is of the form $[\sum a_i|x|_{s_i} \equiv b[c]]$*, with* $\Sigma = \{s_1, \ldots, s_k\}$ *and* $a_1, \ldots, a_k, b, c \in \mathbb{N}^{k+2}$.

Proof: We first apply the following transformation to each agent: For each x_i, we replace x_i by a_i. We suppose that each a_i is in $[0, c-1]$. We also write the number 0_l on the left of the input. We do this with these rules:

$s \in \Gamma$ \ $q \in Q$	q_0
a	$q_0, a, \longleftarrow$
μ_L	$q_1, 0_l, \longrightarrow$

The idea of this machine is:

- To perform on the left side a modulo counter. We initialize it at 0. Each time we go left to the input, we increment the number seen. If we reach c, then we reset it to 0. It corresponds to the following rules for δ:

$s \in \Gamma$ \ $q \in Q$	q_1
i_l, with $i < c-1$	$q_1, (i+1)_l, \longrightarrow$
$(c-1)_l$	$q_1, 0_l, \longrightarrow$

- To decrease a_i times each a_i. Each time a decrement is performed, we go left to increment the modulo counter. Once the a_i reaches 0, we "unmark" it. It can be done by the following rules:

$s \in \Gamma$ \ $q \in Q$	q_1
a, with $a > 0$	$q_1, a-1, \longleftarrow$
0	$q_1, L, \longrightarrow$
L	$q_1, R, \longleftarrow$
R	$q_1, L, \longrightarrow$

- To check that we have the right modulo at the end. For this, once μ_R is reached, we just need to go find the modulo counter:

$s \in \Gamma$ \ $q \in Q$	q_1	q_2
μ_R	$q_2, \mu_r, \longleftarrow$	
L		$q_2, R, \longleftarrow$
b_l		$q_\top, \mu_L, \longrightarrow$
d_l, with $d \neq b$		$q_\perp, \mu_L, \longrightarrow$

$\square$

We have the two basic predicates, we just need now to prove that our model is stable under boolean computation:

Proposition 7.14 *RTM is stable under boolean combinations.*

Proof: To show this result, it suffices to prove the stability under complementary and union.

Negation: We just need here to inverse $q_\top$ and $q_\bot$ in our machines. More generally, we take $F' = Q \setminus F$.

Union: We just have to duplicate the states: We first change each state cell $s \in \Sigma \cup \{\mu_L, \mu_R\}$ with (s, s). We then come back at the beginning of the input.

We run as if we were only on the first element of the couple. Each time we see a B, we treat it as it was a (B, B). When the computation is over, if we are in state $q_\bot$, we come back to the agent right to the only agent of the form $(\_, \mu_L)$ and launch the computation of the second machine on the second element. Again, any B is seen as a (B, B).

This computation will accept the input if and only if it would have been accepted by one of the two machines. □

With the basic tools and the boolean stability, we get the following result:

Theorem 7.15 *Any Semilinear set is in RTM.*

7.2.3 Polylogarithmic Pivotal Complexity

Here is a theorem that provides our first motivation to work on restrictions over the number of pivotal transitions. It proves that the Turing Machines described in Theorem 4.25 have a polylogarithmic pivotal complexity. The Machine described in the next proposition is exactly the same that the one of Theorem 4.25.

Proposition 7.16 *Let a Turing Machine on alphabet Γ recognizing the language L having the following restrictions: There exist some $k \in \mathbb{N}$ such that*

- *The machine has 4 tapes. The first one is for the input x.*
- *The space of work is restricted as follows:*
  - *The first tape uses only the input space of $|x|$ cells.*
  - *The 2nd and the 3rd use at most a space of $\log |x|$ cells.*
  - *The 4th uses at most a space of $\log^k |x|$ cells.*
- *The Machine can only do at most $\log^k |x|$ unitary operations among the following one:*
  1. *A regular Turing Machine step.*
  2. *Mark/Unmark the cells that have the symbol $\gamma \in \Gamma$.*
  3. *Write in binary on the 2nd tape the number of marked cells.*

4. *Go to the cell of the number written on the 3rd tape if this number is smaller than $|x|$.*
5. *Mark/Unmark all the cells left to the pointing head on the first tape.*
6. *Turn into state γ' all the marked cells in state $\gamma \in \Gamma$.*

Then there exists some k' such that $L \in PIV(\log^{k'} n)$.

The result looked obvious to us, as most of the operations listed uses a single pivotal step (at this moment, we thought that computing the size of the population would use $O(\log n)$ pivotal transitions). When we realized that the size could be computed in a single pivotal transition, we obtained the next result, that provides a bound above of RTM.

The differences with the Machine described in the previous proposition are written in bold.

Theorem 7.17 *Let a Turing Machine on alphabet Γ recognizing the language L having the following restrictions: There exists some $k \in \mathbb{N}$ such that*

- *The machine has 4 tapes. The first one is for the input x.*
- *The space of work is restricted as follows:*
  - *The first tape uses only the input space of $|x|$ cells.*
  - *The 2nd and the 3rd use at most a space of $\log |x|$ cells.*
  - *The 4th uses at most a space of **k** cells.*
- *The Machine can only do at most **k** unitary operations among the following one:*
  1. *A regular Turing Machine step.*
  2. *Mark/Unmark the cells that have the symbol $\gamma \in \Gamma$.*
  3. *Write in binary on the 2nd tape the number of marked cells.*
  4. *Go to the cell of the number written on the 3rd tape if this number is smaller than $|x|$.*
  5. *Mark/Unmark all the cells left to the pointing head on the first tape.*
  6. *Turn into state γ' all the marked cells in state $\gamma \in \Gamma$.*

Then $L \in RTM$.

Proof: The 3 tapes will be simulated left to the input. The machine first find $\log n$ by computing the size in binary (we hence get the logarithm written in unary). We do it twice.

The first $\log n$ cells will be for tape 2, the second $\log n$ cells for tape 3. The following ones will be for tape 4. We will see the tapes reversed (we read them from the right to the left instead of the usual reading direction).

Let describe now how each item can be performed with a constant number of pivotal transition:

1. We need at most 5 pivotal transitions to see the symbol on each cell of each tape, and we then need at most two pivotal transitions by tape to update it.

2. We do not need a pivotal transition during a walk through the tape to mark/unmark the cells.

3. It corresponds exactly to compute the size of the input on a subset of symbols of Γ.

4. It corresponds to write in unary the number written in the 3rd tape on the 1st one.

5. Same remark than for 2.

6. Same again.

$\square$

7.3 Rational Languages

7.3.1 The Language (ab)*

The result in this section is a parallel to the impossibility result in Section 4.3.2.

Proposition 7.18 *The language $(ab)^*$ is in RTM.*

Proof:

The idea of the machine is to walk through the input, performing one step backward each first time we see a b cell. We count how much time we went on each cell.

At the end of the walk, if we stepped twice on each cell, the input is of the form $(ab)^n$. If not, the input is not.

More formally, here is the Machine:

- $Q = \{q_0, q_1, q_\top, q_\perp\}$
- $\Gamma = \{B, \mu_L, \mu_R, a, (a,1), (a,2), b, (b,1), (b,2)\}$
- $\Sigma = \{a, b\}$
- $F = \{q_\top\}$
- δ works as follows:

$s \in \Gamma$ \ $q \in Q$	q_0	q_1
a	$q_0, (a,1), \longrightarrow$	$q_\perp, a, \texttt{HALT}$
$(a,1)$	$q_0, (a,2), \longrightarrow$	$q_\perp, (a,1), \texttt{HALT}$
$(a,2)$	$q_\perp, (a,2), \texttt{HALT}$	$q_1, a, \longleftarrow$
b	$q_0, (b,1), \longleftarrow$	$q_\perp, b, \texttt{HALT}$
$(b,1)$	$q_0, (b,2), \longrightarrow$	$q_\perp, (b,1), \texttt{HALT}$
$(b,2)$	$q_\perp, (b,2), \texttt{HALT}$	$q_1, b, \longleftarrow$
μ_R	$q_1, \mu_R, \longleftarrow$	
μ_L	$q_\perp, \mu_L, \texttt{HALT}$	$q_\top, \mu_L, \texttt{HALT}$

We provide here the run on a right and a wrong output. First on *abab*, then on *abaabb*. The first will help to understand why the machine accepts right outputs, the second shows what happens when we have two consecutive a or b.

Run on input *abab*:

q_0 :	. . . B B μ_L	**a**	b	a	b	μ_R B B . . .
q_0 :	. . . B B μ_L	$(a,1)$	**b**	a	b	μ_R B B . . .
q_0 :	. . . B B μ_L	**(a,1)**	$(b,1)$	a	b	μ_R B B . . .
q_0 :	. . . B B μ_L	$(a,2)$	**(b,1)**	a	b	μ_R B B . . .
q_0 :	. . . B B μ_L	$(a,2)$	$(b,2)$	**a**	b	μ_R B B . . .
q_0 :	. . . B B μ_L	$(a,2)$	$(b,2)$	$(a,1)$	**b**	μ_R B B . . .
q_0 :	. . . B B μ_L	$(a,2)$	$(b,2)$	**(a,1)**	$(b,1)$	μ_R B B . . .
q_0 :	. . . B B μ_L	$(a,2)$	$(b,2)$	$(a,2)$	**(b,1)**	μ_R B B . . .
q_1 :	. . . B B μ_L	. . .	. . .	. . .	. . .	μ_R B B . . .
q_1 :	. . . B B $\boldsymbol{\mu_L}$	a	b	a	b	μ_R B B . . .
$\mathbf{q_\top}$:	. . . B B $\boldsymbol{\mu_L}$	a	b	a	b	μ_R B B . . .

Run on input *abaabb*:

q_0 :	. . . B B μ_L	**a**	b	a	a	b	b	μ_R B B . . .
q_0 :	. . . B B μ_L	$(a,1)$	**b**	a	a	b	b	μ_R B B . . .
q_0 :	. . . B B μ_L	**(a,1)**	$(b,1)$	a	a	b	b	μ_R B B . . .
q_0 :	. . . B B μ_L	$(a,2)$	**(b,1)**	a	a	b	b	μ_R B B . . .
q_0 :	. . . B B μ_L	$(a,2)$	$(b,2)$	**a**	a	b	b	μ_R B B . . .
q_0 :	. . . B B μ_L	$(a,2)$	$(b,2)$	$(a,1)$	**a**	b	b	μ_R B B . . .
q_0 :	. . . B B μ_L	$(a,2)$	$(b,2)$	$(a,1)$	$(a,1)$	**b**	b	μ_R B B . . .
q_0 :	. . . B B μ_L	$(a,2)$	$(b,2)$	$(a,1)$	**(a,1)**	$(b,1)$	b	μ_R B B . . .
q_0 :	. . . B B μ_L	$(a,2)$	$(b,2)$	$(a,1)$	$(a,2)$	**(b,1)**	b	μ_R B B . . .
q_0 :	. . . B B μ_L	$(a,2)$	$(b,2)$	$(a,1)$	$(a,2)$	$(b,2)$	**(b,1)**	μ_R B B . . .
q_0 :	. . . B B μ_L	$(a,2)$	$(b,2)$	$(a,1)$	$(a,2)$	**(b,2)**	$(b,1)$	μ_R B B . . .
$q_\perp$:	. . . B B μ_L	$(a,2)$	$(b,2)$	$(a,1)$	$(a,2)$	**(b,2)**	$(b,1)$	μ_R B B . . .

Let prove more formally now the correctness of this machine:

- If the input is in $(ab)^*$, then the first run shows exactly how it will happen. The input will be accepted.

- If the input starts with letter b, the head will go left, then see a μ_L, and will reject.
- If the machine walks through two consecutive a, a $(a, 1)$ will be on the tape when the machine goes to state q_1 (if it does, otherwise, it has rejected the input, as $q_\top$ can only be reached from q_1), and then the machine will reject the input when the head will see it.
- If the machine walks through two consecutive b, a $(b, 2)$ will be written, then the head will come back on it being in state q_0 and will reject the input.
- If the input is in $(a^*b)^*a$, the last a will become a $(a, 1)$, then the head will come on it in state q_1 and will reject the input.

$\square$

Corollary 7.19 *$RTM \not\subset CPPOLYLOG$.*

Proof: Theorem 4.16 states that $(ab)^* \notin CPPOLYLOG$. As we proved $(ab)^* \in RTM$, we get the absence of inclusion. $\square$

For the other inclusion, we have the following conjecture:

Conjecture 7.20 *$CPPOLYLOG \not\subset RTM$.*

This conjecture is motivated by the fact that we do not believe that the multiplication is computable with a finite number of pivotal transitions (for example, the set of inputs A^iB^jn with $n \in \{0, 1\}^*$ such as n is the binary version of $i \times j$). This set is in $CPPOLYLOG$, as it suffices to write in binary the number of A and the number of B, perform the multiplication in binary, and compare the result to n.

7.3.2 The Language u^*

Actually, we have a stronger result: for any word $u \in \Sigma^*$, u^* can be computed. To show this, we first prove it on a weaker version, when $u = a^k$ for some $k > 0$.

Proposition 7.21 *For any $k > 0$, the language $(a^k)^*$ is in RTM.*

Proof: With the modulo predicate computable, it is quite easy to prove the result. We provide here a different solution that will permit to understand how to compute the more general language u^*.

This algorithm perform round trips on the tape, each time seeing one more cell. Each cell counts the number of time we passed on it from right modulo k. The L or R on the state is here to remind on which direction we need to go.

The input is accepted if, after we reached the μ_R, the first letter has a counter 0, as it should have been walked through a multiple of k times.

Here is the formal machine. We will not give an exact proof of its correctness, but a sketch of main ideas.

- $Q = \{q_0, q_1, q_2, q_\top, q_\bot\}$
- $\Gamma = \{B, \mu_L, \mu_R, a\} \cup \{a\} \times [0, k-1] \times \{L, R\}$
- $\Sigma = \{a\}$
- $F = \{q_\top\}$
- δ works as follows (the interactions not listed cannot occur):

$q \in Q$ / $s \in \Gamma$	q_0	q_1	q_2
a	$q_0, (a,0,R), \longleftarrow$		
(a,i,L), with $i < k-1$	$q_0, (a,i+1,R), \longleftarrow$		
$(a,k-1,L)$	$q_0, (a,0,R), \longleftarrow$		$q_\top, (a,k-1,L)$, `HALT`
(a,i,R)	$q_0, (a,i,L), \longrightarrow$		
μ_L	$q_0, \mu_L, \longrightarrow$	$q_2, \mu_L, \longrightarrow$	
μ_R	$q_1, \mu_R, \longleftarrow$		
x, with $x \neq \mu_L$		$q_1, x, \longleftarrow$	
x, with $x \neq (a,k-1,L)$			$q_\bot, x$, `HALT`

Here is a quick view of it works on a^4 on language $(a^3)^*$:

q_0 :	B μ_L	**a**	a	a	a	μ_R B
q_0 :	B $\boldsymbol{\mu_L}$	$(a,0,R)$	a	a	a	μ_R B
q_0 :	B μ_L	**(a,0,R)**	a	a	a	μ_R B
q_0 :	B μ_L	$(a,0,L)$	**a**	a	a	μ_R B
q_0 :	B μ_L	**(a,0,L)**	$(a,0,R)$	a	a	μ_R B
q_0 :	B $\boldsymbol{\mu_L}$	$(a,1,R)$	$(a,0,R)$	a	a	μ_R B
q_0 :	B μ_L	**(a,1,R)**	$(a,0,R)$	a	a	μ_R B
q_0 :	B μ_L	$(a,1,L)$	**(a,0,R)**	a	a	μ_R B
q_0 :	B μ_L	$(a,1,L)$	$(a,0,L)$	**a**	a	μ_R B
q_0 :	B μ_L	$(a,1,L)$	**(a,0,L)**	$(a,0,R)$	a	μ_R B
q_0 :	B μ_L	**(a,1,L)**	$(a,1,R)$	$(a,0,R)$	a	μ_R B
q_0 :	B $\boldsymbol{\mu_L}$	$(a,2,R)$	$(a,1,R)$	$(a,0,R)$	a	μ_R B
q_0 :	B μ_L	...	...	...	a	μ_R B
q_0 :	B μ_L	$(a,2,L)$	$(a,1,L)$	$(a,0,L)$	**a**	μ_R B
q_0 :	B μ_L	$(a,2,L)$	$(a,1,L)$	**(a,0,L)**	$(a,0,R)$	μ_R B
q_0 :	B μ_L	$(a,2,L)$	**(a,1,L)**	$(a,1,R)$	$(a,0,R)$	μ_R B
q_0 :	B μ_L	**(a,2,L)**	$(a,2,R)$	$(a,1,R)$	$(a,0,R)$	μ_R B
q_0 :	B $\boldsymbol{\mu_L}$	$(a,0,R)$	$(a,2,R)$	$(a,1,R)$	$(a,0,R)$	μ_R B
q_0 :	B μ_L	...	...	...	...	μ_R B
q_0 :	B μ_L	$(a,0,L)$	$(a,2,L)$	$(a,1,L)$	$(a,0,R)$	$\boldsymbol{\mu_R}$ B
q_1 :	B μ_L	...	...	...	...	μ_R B
q_1 :	B $\boldsymbol{\mu_L}$	$(a,0,L)$	$(a,2,L)$	$(a,1,L)$	$(a,0,R)$	μ_R B
q_2 :	Bμ_L	**(a,0,L)**	$(a,2,L)$	$(a,1,L)$	$(a,0,R)$	μ_R B
$\boldsymbol{q_\bot}$:	Bμ_L	**(a,0,L)**	$(a,2,L)$	$(a,1,L)$	$(a,0,R)$	μ_R B

$\square$

With the previous machine, we can generalize to create a machine that computes, for any $u \in \Sigma^+$, u^*.

Theorem 7.22 *For any $u \in \Sigma^+$, the language u^* is in RTM.*

Proof: We have $u = u_1 u_2 \dots u_k$. This machine works in two steps:

1. We first run the $(a^k)^*$ machine with $k = |u|$ and by keeping track the input letter. When we go on state q_1 (i.e. when we walk on a μ_R), we go to step 2.
2. Each cell of the input contains now a number: 0 on the last cell, 1 on the previous,... $i[k]$ on the ith cell starting from the right.

   We just need to check that, for each letter (s, i, L) on the tape (with $s \in \Gamma$ and $0 \le i < k$), $s = u_{k-i}$. The input is in u^* if and only if for each cell of the input we have $s = u_{k-i}$.

□

7.3.3 The Language $(u + v)^*$

Here are some results about a study of the languages $(u+v)^*$. To answer the question "Is any rational language in RTM?", we thought that solving the general case $(u+v)^*$ would help a lot.

Unfortunately, we did not manage to find a general result. Here is a small list of what subclass of languages we succeeded to compute in RTM:

Proposition 7.23 *The following languages are in RTM:*

- $(\sum u_i)^*$ *with* $\forall i, j$, $|u_i| = |u_j|$.
- $\left(\sum a_i^{b_i}\right)^*$ *with* $a_i \in \Sigma$ *and* $b_i > 0$.
- $(\sum u_i)^*$ *with* $|u_i| \in \Sigma_i^*$, *and* $\forall i \neq j$, $\Sigma_i \cap \Sigma_j = \emptyset$.

The machines corresponding to these languages will not be provided in this document.

7.3.4 Star Height

As we struggled to deal with languages of Star Height one, we wondered if this could be a limit of our model. It appeared that not.

First, here is a definition of the star height of a regular expression and of a language. All these definitions come from Jacques Sakarovitch's book Elements of Automata Theory [Sak09].

Definition 7.24 ([Sak09]) *The* **Star Height of a Regular Expression** *E, noted $h[E]$, is computed as follows:*

if $E = 0$, $E = 1$ or $E = a \in \Sigma$	$h[E] = 0$
if $E = E' + E''$ or $E = E' \times E''$	$h[E] = max(h[E'], h[E''])$
if $E = F^*$	$h[E] = 1 + h[F]$

The **Star Height of a Language** *corresponds then to the minimal Star Height among the regular expressions that correspond to this language.*

Theorem 7.25 ([Sak09]) *The language W_q over $\{a, b\}^*$ consisting of words whose number of a's is congruent to the number of b's modulo 2^q has star height q.*

Proposition 7.26 *For any $q \in \mathbb{N}$, there exists a language of star height q in RTM.*

Proof: It suffices to see that W_q corresponds to the class $[|x|_a \equiv |x|_b[2^q]]$.

The modulo machine of Proposition 7.13 uses more than 2^q symbols for Γ and a constant number of pivotal transition, whatever q is. There is also a machine that uses a constant number of symbols in Γ and $O(q)$ pivotal transitions:

We compute the number of a in binary, then the number of b. We then just need to check that the first q digits are the same to accept the input. □

7.3.5 Context-Free Languages

Actually, even if we do not know if every rational language is in RTM, we actually now than some languages in RTM are not rational. We provide here two of them, one is trivial, the second one being more interesting.

Proposition 7.27 *There exists $L \in RTM$ that is not a rational language.*

Proof: We have $(a^n b^n)_{n\in\mathbb{N}} \in RTM$.

To prove it, it suffices to notice that it is the intersection between the sorted language (with $s1 = a$ and $s2 = b$) and the language satisfying the Presburger's predicate $[|x|_a = |x|_b]$. □

Remark 7.28 *Actually, RTM is also not included in Context-Free Languages:*

The language $(a^n b^n c^n)_{n\in\mathbb{N}}$ is in RTM.

Definition 7.29 *The* **Well-Formed Parentheses** *language corresponds to the words in $\{[,]\}^*$ such as the parenthesizing is good. More formally, it corresponds to the following context-free grammar:*

$$\begin{cases} S \to SS \\ S \to [S] \\ S \to [\,] \end{cases}$$

Proposition 7.30 *The well-formed parentheses language is in RTM.*

Proof: The principle of the machine is to go right as long as we see an opening parenthesis [, marking them. Each time we see a closing parenthesis], we "delete it" (turning it into a L) and go right. When we come back on a marked [, we delete it and go right.

If we reach at some point μ_L, it means that we saw more] than [, and the machine rejects. When we reach μ_R, we then check that there are only deleted cells on the tape. We accept if and only if it is the case (if there are non deleted cells remaining, it only can be a [marked, as each] have been deleted).

This machine works on the principle that at any point, the number of [seen is bigger or equal to the number of] seen, and that this sum must be equal to 0 at the end. Here, we meet too early μ_L if and only if there is a suffix of the input with more] than [. If at the end there is at least one [not deleted, it implies that the sum is greater than 0. $\square$

Remark 7.31 *We did not find any machine for the case of two parentheses. I believe there is not.*

7.4 Decidability

7.4.1 Halting Problem

We focus on the result from [Sao95]. Saouter proved that, in the case of a one-state Turing Machine, the halting problem is decidable. Hence, from this result, we get the decidability of halting for Rusted Turing Machine.

Theorem 7.32 ([Sao95]) *The halting problem for the one-state Turing Machines is decidable.*

Theorem 7.33 *The halting problem for Rusted Turing Machines, that is to say determining if a Rusted Turing Machine stops given some input x, is decidable.*

Proof: Resolving the question "Will the machine reach a pivotal transition?" is equivalent to the halting problem for one-state Turing Machines: instead of looking for an halting state, we look for either an halting state, either for a pivotal transition.

As the transition graph is acyclic, we know that we need to repeat the halting problem at most $|Q|$ times. $\square$

We use the fact here that we have a upper bound on the number of pivotal transition (being the size of Q). In fact, it is not even necessary to know it:

Corollary 7.34 *The halting problem for Turing Machines in $PIV(1)$ is decidable.*

Proof: Theorem 7.32 permits to decide if the machine will change its state at some point, will halt or will loop.

We just actually need to repeat this decidability again and again until either we realize that the machine does not halt or until we reach an halting state.

We do not know how many times the process will be repeated, but we know that it will use a finite number of times the process, and hence this decision machine halts. □

7.4.2 Maximum of Cells Reached

We have just stated that the halting problem is decidable. We will provide now an upper bound on the number of cells reached in a single pivotal transition. To prove that it is exponential in the size of the input, we first provide a few definitions from the article of Saouter [Sao95].

We work here before that a pivotal transition happens.

Definition 7.35 *Let M be a Turing Machine and x an input written on a tape. We call the* **Number of Solicitations on the Left** *on input x (or from cell y) the number of times the reading head of the machine goes left to the first cell of the input x from the first cell of x (or goes left to cell y).*

We call the **Orbit** *of symbol s on state q the sequence $(s_i, d_i)_{1 \leq i \leq j}$ such that $s_0 = s$, $\forall i < j$, $\delta(q, s_i) = (q, s_{i+1}, d_{i+1})$ and $\delta(q, s_j) = (q', s', d')$ with $q \neq q'$ or $d' = \mathtt{HALT}$. If j does not exist, the orbit is noted $(s_i, d_i)_{i \in \mathbb{N}}$. We chose to have $d_i \in \{-1, 0, +1, \mathtt{HALT}\}$.*

For $k \in \mathbb{N}$, we note $\mathbf{L}(k)$ the number of $d_i = -1$, with $i \leq k$. Symmetrically, we note $\mathbf{R}(k)$ the number of $d_i = +1$, with $i \leq k$. $L(k)$ (resp. $R(k)$) corresponds to the number of times we went left (resp. right) in the first k elements of the orbit.

Remark 7.36

- *Starting from now, we consider that for any $i < j$, $d_i \neq 0$, as a Turing machine where we go directly from s_i to s_{i+2} would be equivalent in that case.*

  Hence, we can notice that for any $k \leq j$, $L(k) + R(k) = k$.

- *From now on, we work on the orbit of the blank symbol B (i.e. $s = B$).*

- *We consider that the input starts on cell 0. The first B on its left is cell -1....*

Proposition 7.37 *If there exists some k such that $\sum_{i \leq k} d_i < 0$, then after at most $\frac{k+1}{2}$ solicitations on the left of the input, the reading head diverges, and the machine never halts.*

Proof: Let chose k minimal. We have $d_k = -1$, $\sum_{i \leq k-1} d_i = 0$ and $L(k-1) = R(k-1) = \frac{k-1}{2}$. We try to perform $\frac{k+1}{2}$ solicitations on the left from the input.

Let show by induction on $n \in \mathbb{N}$ the following result:

On the cell $-n$, there are two possible options:

Case 1 There were at least $\frac{k+1}{2}$ solicitations on the left from the cell $1-n$.

Case 2 At some point, we had at most $\frac{k+1}{2}$ solicitations on the left from the cell $1-n$, and we went left to cell $-n$ and never walked back on it.

Case n=1 : It is true by definition, as we try to perform $\frac{k+1}{2}$ from the right. Either we managed to do it, either the head never came back.

Case n+1 : Let differentiate according to which case we were with n:

Case 1 Let consider the moment when we walked through the cell $-n$ from the right the $\frac{k+1}{2}$th time. The cell is in state s_i.

The next $k-i$ times we go on this cell, we must each time go left, as we reached $R(i) = R(k)$. Actually, by definition of k, if we come back $k-i$ times, as we will be on s_k, we will go left a $\frac{k+1}{2}$th time. (We can notice that we never came back on cell $1-n$, and hence did not increase any number of solicitations of cell $-j$ for $j < n$).

If we do not come back $k-i$ times to $-n$, we actually got exactly $\frac{k+1}{2}$ solicitations on the left from $1-n$, it means that we are actually also in Case 2. The next point will solve it.

We notice that whatever happens, the number of solicitations from $1-n$ cannot become greater than $\frac{k+1}{2}$.

Case 2 We suppose that the point when we never go left to cell $-n$ is reached. Each time now that we go on cell $-n-1$, we go left. Either it happens less than $\frac{k+1}{2}$ times and then we go left to cell $-n-1$ and never walk back on it, either we did at least $\frac{k+1}{2}$ solicitations on the left from $-n$.

This proof actually proves that on each cell, we perform at most $\frac{k+1}{2}$ solicitations on the left (see end of case 1 to get a conviction). This induction also implies that we go on each cell of the left part of the tape, and we do not come back on them at some point.

Hence, the machine diverges and never halts. □

Here are some properties for the cells the machine visited on the left part of the tape:

Proposition 7.38 *We suppose that for any k, $\sum_{i \leq k} d_i \geq 0$. We have the following properties:*

1. *If the machine went on cell $-n$, and its head is now on a cell $m \geq 0$, let s_k the corresponding symbol on cell $-n$. We have $R(k) > L(k)$.*

2. *If the head of the machine is now on a cell $m \geq 0$, let s_k the symbol on cell $-n$ and s_l the symbol on cell $-n-1$. We have $L(k) = R(l)$.*

3. *If the machine went on cell $-n-1$, and its head is now on a cell $m \geq 0$, let s_k the symbol on cell $-n$ and s_l the symbol on cell $-n-1$. We have $k - l \geq 2$.*

Proof:

1. The last time we went on cell $-n$, we must have gone right (i.e. $d_k = +1$), else, the head should be left to cell $-n$. We have $\sum\limits_{i \leq k-1} d_i \geq 0$.

   Hence, $R(k) - L(k) = \sum\limits_{i \leq k} d_i \geq 0 + 1 > 0$.

2. We started the computation right to cell $-n$, and we are now again right to this cell. Hence, each time we went left to cell $-n$, we must have come back right to cell $-n-1$ (else we would still be left to cell $-n-1$).

   Hence, $L(k) = R(l)$.

3. As we work on integers, we have $R(k) \geq 1 + L(k)$ (same for l). From Remark 7.36, we have $k = R(k) + L(k)$ (same for l). Hence:

$$\begin{aligned} k - l &= R(k) + L(k) - R(l) - L(l) & \\ &\geq 1 + 2L(k) - R(l) - L(l) & \text{as } R(k) \geq 1 + L(k) \\ &\geq 1 + R(l) - L(l) & \text{as } L(k) = R(l) \\ &\geq 2 & \text{as } R(l) \geq 1 + L(l) \end{aligned}$$

□

From this, we finally get a bound above of the number of cells visited after a fixed number of solicitations on the left from the input:

Proposition 7.39 *We suppose that for any k, $\sum\limits_{i \leq k} d_i \geq 0$.*

If after n solicitations on the left from the input, the head is on cell $m \geq 0$, the machine visited at most n cells on the left of the input.

Proof: Let s_k be the symbol on the cell -1 now. We have $R(k) = n$, as we came back exactly once right to -1, to come back after a solicitation. Hence, as $R(k) > L(k)$ and $k = R(k) + L(k)$, $k \leq 2n - 1$.

Each visited cell's index (i.e. i if it is in state s_i) is inferior by at least 2 to the index of the cell on its right. Hence, there are at most $\frac{2n-1}{2}$ cell visited left to cell -1. We get the result of at most n cell visited. □

Remark 7.40 *Actually, n is reached when $\forall i$, $d_{2i+1} = +1$ and $d_{2i} = -1$, which corresponds for example to the rules:*

$q \in Q$ $s \in \Gamma$	q
B	$q, R, \longrightarrow$
R	$q, L, \longrightarrow$
L	$q, R, \longleftarrow$

With all these results, we get the following theorem:

Theorem 7.41 *Before a single first pivotal transition, a Turing Machine writes on at most* $|\Gamma|^{|x|}$*, where* x *is what is written on the tape (without any blank symbol* B *on* x*). We assume that the head starts somewhere on the input.*

Proof: We assume here that $d_1 = +1$ to use the previous results directly. If not, the study is symmetric, but we work on the right of the tape instead of its left. We will differentiate three scenarios.

If there exist some k such that $\sum_{i \leq k} d_i < 0$, we know from Proposition 7.37 that after at most $k/2$ solicitations on the left, the head diverges. Hence, if we managed to perform a pivotal transition, the number of blank symbol B cells reached by the machine is finite, whatever the size of x. It only depends on the number of solicitations on the left from the input and the orbit of B.

If there exist some k such that $d_k =$ `HALT`, after at most k solicitations on the left from the input, either the machine diverged, either it stopped. Again, the number of B cells reached by the machine is finite, whatever the size of x.

Now, we consider the interesting case: for any $k \in \mathbb{N}$, $\sum_{i \leq k} d_i \geq 0$.

Either after a fixed number of solicitations, the machine diverges, either it always come back on a cell $m \geq 0$. The first case is not what interests us, we will then consider the case where the always come back.

From Proposition 7.39, the number of cell visited depends only on the number of solicitations on the left from the input. We never can go right to the input (if we do, we diverge as $d_1 = +1$).

If at some point, when we come back from a solicitation on cell 0, on the $|x|$ cells (0 to $|x|$) we have a word in $\Gamma^{|x|}$ that already appeared, the machine will never reach a pivotal transition: It cannot come from the left part, as there is no pivotal transition in the orbit of B, and we will loop on the words written on the input part of the tape.

Hence, we can only have at most $|\Gamma|^{|x|}$ different words seen after the come back of a solicitation. If we perform a pivotal transition, it happens after at most $|\Gamma|^{|x|}$ solicitations. From Proposition 7.39, we will have written on at most $|\Gamma|^{|x|}$ B cells. $\square$

Remark 7.42 *We can actually reach* $(|\Gamma|-3)^{|x|}$*: It corresponds to see the input as a number written in base* $|\Gamma| - 3$*, and perform a -1 operation each time we go on the input part. On the output part, we then perform the rules given in Remark 7.40.*

From Theorem 7.41, if we iterate the bound for each pivotal transition, we get a boundary above of the number of the size of the used tape at the end of the computation of a Rusted Turing Machine. This boundary is brutal.

Corollary 7.43 *Consider M be a Rusted Turing Machine, let $a = |\Gamma| + 1$ and let x be an input. An upper bound on the number of cells used by M on the input x is:*

$$a^{a^{\cdot^{\cdot^{\cdot^{a^{|x|}}}}}}$$

where a appears $|Q|$ times.

Proof: We have the following upper bound for the number m_n of non B cells after a pivotal transition on a tape of n used cells:

$$m_n \leq n + |\Gamma|^n \leq (|\Gamma| + 1)^n$$

We then get the result by iterating $|Q|$ times the previous upper bound (as there can be at most $|Q|$ pivotal transitions). $\square$

7.4.3 A machine not in RTM

From previous section, we get an upper bound of the Busy Beaver of the Rusted Turing Machines.

Definition 7.44 *We note $\mathfrak{BB}_{RTFM}(a, b)$ the* **Busy Beaver of Rusted Turing Machines** *with $a = |Q|$ states and $b = |\Gamma|$ symbols. It corresponds to the maximum number of of non-blank symbols finally on the tape from the input empty on these machines.*

Proposition 7.45 *We have $\mathfrak{BB}_{RTFM}(a-1, b-1) \leq b^{b^{\cdot^{\cdot^{\cdot^{b}}}}}$, where b appears a times.*

Proof: From q_0, we must go directly to some q_1 writting a new symbol.
We then use the upper bound of the previous corollary on an input of size 1. $\square$

With this upper bound, we know that machines that will wright on more than this number of cells cannot be in RTM.

Theorem 7.46 *The function that write, from input integer x, $\mathfrak{BB}_{RTFM}(x, x)$ is not in RTM.*

Proof: If this function could be computed by a machine in RTM, let m be the maximum between the number of states of this machine and the number of symbols the machine can write. $\square$

7.5 Conclusion

7.5.1 Resume

We introduced a new complexity consideration, the pivotal complexity. With a constant number of pivotal transitions, a Turing Machine is able to recognize at least:

- The sorted language.
- Semilinear predicates.
- $(u+v)^*$ with some conditions over u and v.
- a^nb^n and $a^nb^nc^n$.
- Well-formed paratheses language.

We proved that it is possible to write down in binary the size of the input, which permitted to simulate a similar machine that the one introduced with $CPPOLYLOG$ in Theorem 4.25.

We proved that the halting problem can be decided, and provided an upper bound of the number of cells that can be reached by a Rusted Turing Machine.

7.5.2 Open Questions

Remains open the question of what cannot be computed. Is there some rational language that is not in RTM? The star height result (Proposition 7.26) makes me think that the set of rational languages is included in RTM. But the absence of solution for the general case of $(u+v)^*$ does not help to give any certitude.

For the well-formed parentheses, we solved the case of a single type of parentheses. The case with two or more different parentheses is still open.

We are still looking for an explicit language not computable by a Rusted Turing Machine.

We also raised an interesting question in Remark 7.6: If $M \in Piv(1)$, is there a majoration of the pivotal complexity depending on the size of Q and Σ?

Conclusion

It is now time to conclude this message. We introduced variations and restrictions of models.

To Population Protocols, we added rational behaviors (throw game theory and trust) and studied the weakening of the new versions, providing protocols and some characterization results.

To Community Protocols, we analyzed what can be computed if we restrict the number of expected interactions. We then worked on a case where identifiers are no longer unique. Each time, protocols have been provided, and some computational results have been found.

We worked with the addition of other tools to agents. We shew what happens when we provide Turing Machine with space $O(\log \log n)$, unordered identifiers or Stack automata.

We finished with the introduction of a new class of Turing Machine. We gave some machines, and finally provided an upper bound on the space used by these machines.

We summarize now the main results and provide a landscape of open questions on the works of this document.

Weakening of Population Protocols

We have introduced two new restrictions of the model of Population Protocols.

Game Theory Consideration

We interpreted population's interactions as games in Chapter 2. We introduced the pavlovian behavior to understand how the players would update their strategies.

We proved that with our model, it is possible to compute the following predicates:

- $[x_A \geq 3]$.
- The Majority Protocol $[x_A > x_B]$.
- $[x_A \geq 2^k]$ for any $k \in \mathbb{N}$.

We did not find any exact characterization of the computational power of the pavlovian populations protocols, but we raised two conjectures. The first is about modulo predicates, the second about counting agents:

Conjecture 2.38 *There is no pavlovian population protocol that computes a modulo predicate of the form $[x_A \equiv b[c]]$ for any $c > b \geq 0$.*

Conjecture 2.39 *There is a pavlovian population protocol that computes the threshold predicate $[x_A \geq b]$ for any $b \in \mathbb{N}$.*

An Opinion Consideration

In Chapter 3 each agent has an opinion linked to their internal state, with the restriction that two interacting agents with the same opinion cannot change it. We provided an exact characterization of what this restriction computes.

Theorem 3.15 *The partitions that can be computed by trustful population protocols that trust the output is such that each class of the partition can be defined as a boolean combination of 0-threshold predicates.*

To obtain this result, we studied the restriction where opinion and output are the same. We did not find an exact characterization of this subrestriction, but we have an idea of it:

Conjecture 3.23 *Trustful population protocols that trust the output compute exactly sets that can be described by disjunctions of the form:*

$$\bigcup_{0\leq i<k}\left(\bigcap_{h(i)<j\leq h(i+1)}(y_j\cdot x>0)\cap\bigcap_{j\leq h(i)}(y_j\cdot x=0)\right)$$

Where $k \leq l \leq d$, each $y_i \in \mathbb{N}^d$, and $h : [0,k] \to [0,l]$ is a strictly increasing function with $h(0) = 0$.

The geometric interpretation is that the computed sets are convex strict cones united recursively in their frontiers by computable sets.

We proved that the model is frequency based, and that the frequencies can be computed if agents carry a Turing Machine. The question of what can trustful agents can compute with passively mobile protocols is open.

Weakening of Community Protocols

Two new restrictions of Community Protocols have been studied.

A Speed Consideration

Chapter 4 studied what happens with community protocols when we want at most $O(n \log^k n)$ expected interactions to compute a predicate for some $k \in \mathbb{N}$.

We proved that this restriction is weaker than Community Protocols, as the rational language $(ab)^*$ is not in the set.

We provided a the tightest bound we found of what can be computed:

Theorem 4.25 *Let a Turing Machine on alphabet Γ recognizing the language L having the following restrictions: There exists some $k \in \mathbb{N}$ such that*

- *The machine has 4 tapes. The first one is for the input x.*
- *The space of work is restricted as follows:*
  - *The first tape uses only the input space of $|x|$ cells.*
  - *The 2nd and the 3rd use at most a space of $\log |x|$ cells.*
  - *The 4th uses at most a space of $\log^k |x|$ cells.*
- *The Machine can only do at most $\log^k |x|$ unitary operations among the following one:*
  1. *A regular Turing Machine step.*
  2. *Mark/Unmark the cells that have the symbol $\gamma \in \Gamma$.*
  3. *Write in binary on the 2nd tape the number of marked cells.*
  4. *Go to the cell of the number written on the 3rd tape if this number is smaller than $|x|$.*
  5. *Mark/Unmark all the cells left to the pointing head on the first tape.*
  6. *Turn into state γ' all the marked cells in state $\gamma \in \Gamma$.*

Then we have $L \in CPPOLYLOG$.

This bound raises the question: Is there any language in $CPPOLYLOG$ that cannot be computed by this class of Turing Machines?

We also found the following set inclusions:

Theorem 4.22 & 4.24

$$POLYLOGTIME \subsetneq CPPOLYLOG \subset NSPACE(n \log n) \cap \bigcup_{k\in\mathbb{N}} SPACE(n \log^k n)$$

An Homonymy Consideration

Chapter 5 considered the case where, in the community protocols model, identifiers might no longer be unique. We added the hypothesis that agents are aware when two identifiers are consecutive.

A hierarchy depending on the ratio $f(n)$ of identifiers present in the population has been found. The hierarchy is summarized in the following table:

$f(n)$ identifiers	Computational power
$O(1)$	Semilinear Sets Theorem 5.25
$\Theta(\log^r n)$ with $r \in \mathbb{R}_{>0}$	$\bigcup_{k\in\mathbb{N}} MNSPACE\left(\log^k n\right)$ Theorem 5.22
$\Theta(n^\epsilon)$ with $\epsilon > 0$	$MNSPACE(n \log n)$ Theorem 5.23
n	$NSPACE(n \log n)$ [GR09]

We proved that with $\log n$ identifiers, it is possible to simulate a non deterministic Turing Machine on a tape of size $\log n$.

With $f(n)$ identifiers, we were able to create for any $k \in \mathbb{N}$ $f(n)^k$ identifiers. This permits to have most of the results with $f(n) \geq \log^r n$ for any $r > 0$.

Finally, we left an open question: what can exactly be computed for the case $f(n) = o(\log n)$. We know, for example, it is possible to decide if the identifier is even or odd, but we have no clue for any exact characterization.

Population Protocols with other Tools

Chapter 6 brought some considerations for cases where agents carry tools that are not ordered identifiers.

The result for $f(n) = \Theta(\log^r n)$ with homonym population protocols permitted, in Section 6.1, to answer the open question from the Passively Mobile Protocols model: "What can be computed when each agent carries a Turing Machine with a space $O(\log\log n)$?" in the Passively Mobile Agents model.

The result filled the hole in the Hierarchy summarized in the following table:

Space per agent $S(n)$	Computational power
$O(1)$	Semilinear Sets [AAER07, AAD+04]
$o(\log\log n)$	Semilinear Sets [CMN+11]
$\Theta(\log\log n)$	$\bigcup_{k\in\mathbb{N}} SNSPACE(\log^k n)$ Theorem 6.1
$\Omega(\log n)$	$SNSPACE(nS(n))$ [CMN+11]

In Section 6.2, we studied community protocols with unordered identifiers.

We provided an exact characterization of the restriction:

Theorem 6.7 *The set of functions that can be computed by a protocol with $\sqrt[k]{n}$ incomparable identifiers is exactly $SMNSPACE(n \log n, \sqrt[k]{n})$.*

We opened the question of what happens when the population shares $O(\log n)$ identifiers. The answer is known in the case of ordered identifiers, bu the protocols constructed no longer work here.

We considered in Section 6.3 what happens when agents can carry a stack of tokens. We proved formally that this model can simulate any non deterministic Turing Machine, by showing how to handle a major problem with Population Protocols: When an agent finally starts to interact with the rest of the population.

Theorem 6.11 *Population protocols with a stack compute exactly the set of languages stable under permutation computable by non deterministic Turing Machines.*

A Transition Consideration with Turing Machines

We considered in Chapter 7 a new element for complexity over Turing Machine: the number of pivotal transitions. More precisely, we focused on on Rusted Turing Machines, the class of machines where the number of pivotal transition does not depend on the input's size.

Even if the model looks weak, we provided machines that computes:

- The sorted language.
- Semilinear predicates.
- $(u+v)^*$ with some conditions over u and v.
- a^nb^n and $a^nb^nc^n$
- Well-formed parentheses language.

These results proved that the model cannot be easily compared to regular languages, context-free languages.

These results opened two questions: Are there rational languages not in RTM? Is the Well-formed parentheses language with two or more different parentheses in RTM?

We proved that the halting problem for this class of machines is decidable. We then gave an upper bound of the number of cells reached with a single pivotal transition:

Theorem 7.41 *Before a single first pivotal transition, a Turing Machine writes on at most $|\Gamma|^{|x|}$, where x is what is written on the tape (without any blank symbol B on x). We assume that the head starts somewhere on the input.*

This result permitted to have an upper bound of the Busy Beaver of the model.

The quest of an explicit languages that cannot be computed by this model is still open.

Personal Bibliography

The articles and journal articles that are fully treated in this thesis are marked with a *.

Journal

*1. Olivier Bournez, Jérémie Chalopin, Johanne Cohen, Xavier Koeger, and Mikaël Rabie. Population protocols that correspond to symmetric games. *International Journal of Unconventional Computing (IJUC)*, 2013.

Conferences

1. Olivier Bournez, Jérémie Chalopin, Johanne Cohen, Xavier Koegler, and Mikaël Rabie. Computing with pavlovian populations. In *On Principles Of Distributed Systems (OPODIS)*, 2011.

2. Shlomi Dolev, Panagiota N. Panagopoulou, Mikaël Rabie, Elad Michael Schiller and Paul G. Spirakis. Rationality authority for provable rational behavior. In *Principles of Distributed Computing, (PODC)* 2011. Brief announcement.

3. Marin Bougeret, Henri Casanova, Mikaël Rabie, Yves Robert and Frédéric Vivien. Checkpointing strategies for parallel jobs. In *High Performance Computing Networking, Storage and Analysis*, 2011.

*4. Olivier Bournez, Jonas Lefèvre, and Mikaël Rabie. Trustful population protocols. In *Distributed Computing (DISC)*, 2013.

*5. Olivier Bournez, Johanne Cohen, and Mikaël Rabie. Homonym population protocols. *NETYS*, 2015.

Bibliography

[AAC$^+$05] Dana Angluin, James Aspnes, Melody Chan, Michael J. Fischer, Hong Jiang, and René Peralta. Stably computable properties of network graphs. In *Distributed Computing in Sensor Systems (DCOSS)*, 2005.

[AAD$^+$04] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *Principles of Distributed Computing (PODC)*, 2004.

[AAE06] Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing (DISC)*, 2006.

[AAER07] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing (DISC)*, 2007.

[AAFJ05] Dana Angluin, James Aspnes, Michael J. Fischer, and Hong Jiang. Self-stabilizing population protocols. In *Conference on Principles of Distributed Systems (OPODIS)*, 2005.

[AAI$^+$12] Sergio Arévalo, Antonio Fernández Anta, Damien Imbs, Ernesto Jiménez, and Michel Raynal. Failure detectors in homonymous distributed systems (with an application to consensus). In *International Conference on Distributed Computing Systems*, 2012.

[AR07] James Aspnes and Eric Ruppert. An introduction to population protocols. In *Bulletin of the EATCS*, 2007.

[Axe84] Robert M. Axelrod. *The Evolution of Cooperation.* Basic Books, 1984.

[BBB13] Joffroy Beauquier, Peva Blanchard, and Janna Burman. Self-stabilizing leader election in population protocols over arbitrary communication graphs. In *Principles of Distributed Systems (OPODIS)*, 2013.

[BBBD11] Joffroy Beauquier, Peva Blanchard, Janna Burman, and Sylvie Delaët. Computing time complexity of population protocols with cover times - the zebranet example. In *Stabilization, Safety, and Security of Distributed Systems (SSS)*, 2011.

[BBK09] Joffroy Beauquier, Janna Burman, and Shay Kutten. Making population protocols self-stabilizing. In *Stabilization, Safety, and Security of Distributed Systems, (SSS)*, 2009.

[BBRR12] Joffroy Beauquier, Janna Burman, Laurent Rosaz, and Brigitte Rozoy. Non-deterministic population protocols. In *Principles of Distributed Systems, (OPODIS)*, 2012.

[BCC$^+$11] Olivier Bournez, Jérémie Chalopin, Johanne Cohen, Xavier Koegler, and Mikaël Rabie. Computing with pavlovian populations. In *On Principles Of Distributed Systems (OPODIS)*, 2011.

[BCC$^+$13] Olivier Bournez, Jérémie Chalopin, Johanne Cohen, Xavier Koeger, and Mikaël Rabie. Population protocols that correspond to symmetric games. *International Journal of Unconventional Computing (IJUC)*, 2013.

[BCR15] Olivier Bournez, Johanne Cohen, and Mikaël Rabie. Homonym population protocols. *NETYS*, 2015.

[Bin91] Ken Binmore Binmore. *Fun and Games - A Text on Game Theory.* D. C. Heath & Co., 1991.

[BLR13] Olivier Bournez, Jonas Lefèvre, and Mikaël Rabie. Trustful population protocols. In *Distributed Computing (DISC)*, 2013.

[Bré01] Pierre Brémaud. *Markov Chains, Gibbs Fields, Monte Carlo Simulation, and Queues.* Springer-Verlag, New York, 2001.

[CFL09] Claudio Castellano, Santo Fortunato, and Vittorio Loreto. Statistical physics of social dynamics. *Reviews of modern physics*, 2009.

[Cha09] Bernard Chazelle. Natural algorithms. In *Symposium on Discrete Algorithms (SODA)*, 2009.

[CHKB35] Constantin Carathéodory, Ernst Hölder, Rolf Klötzler, and Hermann Boerner. *Variationsrechnung und partielle Differentialgleichungen erster Ordnung.* BG Teubner Leipzig, 1935.

[CMN$^+$11] Ioannis Chatzigiannakis, Othon Michail, Stavros Nikolaou, Andreas Pavlogiannis, and Paul G. Spirakis. Passively mobile communicating machines that use restricted space. In *International Workshop on Foundations of Mobile Computing*, 2011.

[CMS10] Ioannis Chatzigiannakis, Othon Michail, and Paul G. Spirakis. Algorithmic verification of population protocols. In *on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 2010.

[DFG$^+$11] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, Anne-Marie Kermarrec, Eric Ruppert, and Hung Tran-The. Byzantine agreement with homonyms. In *Symposium on Principles of Distributed Computing (PODC)*, 2011.

[DFGR06] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Eric Ruppert. When birds die: Making population protocols fault-tolerant. In *Distributed Computing in Sensor Systems, (DCOSS)*, 2006.

[DFT12] Carole Delporte-Gallet, Hugues Fauconnier, and Hung Tran-The. Homonyms with forgeable identifiers. In *on Structural Information and Communication Complexity (SIROCCO)*, 2012.

[DGG$^+$02] Martin E. Dyer, Leslie Ann Goldberg, Catherine S. Greenhill, Gabriel Istrate, and Mark Jerrum. Convergence of the iterated prisoner's dilemma game. *Combinatorics, Probability & Computing*, 2002.

[DK65] DJ Daley and DG Kendall. Stochastic Rumours. *IMA Journal of Applied Mathematics*, 1965.

[DLB$^+$13] G.A. Di Luna, R. Baldoni, S. Bonomi, and Ioannis Chatzigiannakis. Counting the number of homonyms in dynamic networks. In *on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 2013.

[EK10] David Easley and Jon Kleinberg. *Networks, crowds, and markets*, 2010.

[ER61] Paul Erdös and Alfréd Rényi. On a classical problem of probability theory. *Publ. Math. Inst. Hung. Acad. Sci.*, 1961.

[FKL07] Pierre Fraigniaud, Amos Korman, and Emmanuelle Lebhar. Local MST computation with short advice. In *Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2007.

[FKP11] Pierre Fraigniaud, Amos Korman, and David Peleg. Local distributed decision. In *Foundations of Computer Science (FOCS)*, 2011.

[FL96] Drew Fudenberg and David K. Levine. *The Theory of Learning in Games.* Number 624. 1996.

[FMP04] Laurent Fribourg, Stéphane Messika, and Claudine Picaronny. Coupling and self-stabilization. In *Distributed Computing (DISC)*, 2004.

[Gil92] D.T. Gillespie. A rigorous derivation of the chemical master equation. In *Physica A*, 1992.

[GR09] Rachid Guerraoui and Eric Ruppert. Names trump malice: Tiny mobile agents can tolerate byzantine failures. In *Automata, Languages and Programming*, 2009.

[Her69] Gabor T Herman. The uniform halting problem for generalized one-state turing machines. *Information and Control*, 1969.

[Het00] Herbert W. Hethcote. The mathematics of infectious diseases. *SIAM Review*, 2000.

[HOT11] Julien M Hendrickx, Alex Olshevsky, and John N Tsitsiklis. Distributed anonymous discrete function computation. *Automatic Control, IEEE Transactions on*, 2011.

[HS03] Josef Hofbauer and Karl Sigmund. Evolutionary game dynamics. *Bulletin of the American Mathematical Society*, 2003.

[Imm88] Neil Immerman. Nondeterministic space is closed under complementation. In *Structure in Complexity Theory Conference*, 1988.

[JSW11] Aaron D. Jaggard, Michael Schapira, and Rebecca N. Wright. Distributed computing with adaptive heuristics. In *Innovations in Computer Science (ICS)*, 2011.

[KK88] D. Kraines and V. Kraines. Pavlov and the prisoner's dilemma. *Theory and Decision*, 1988.

[MCS11] Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. Mediated population protocols. *Theor. Comput. Sci.*, 2011.

[MCS12] Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. Terminating population protocols via some minimal global knowledge assumptions. In *on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 2012.

[Min67] Marvin L. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

[Mur02] James Dickson Murray. *Mathematical Biology. I: An Introduction.* Springer, third edition, 2002.

[NS93] Martin Nowak and Karl Sigmund. A strategy of win-stay, lose-shift that outperforms tit-for-tat in the Prisoner's Dilemma game. *Nature*, 1993.

[OR94] Martin J. Osbourne and Ariel Rubinstein. *A Course in Game Theory.* MIT Press, 1994.

[Pre29] Mojzesz Presburger. Uber die Vollstandig-keit eines gewissen systems der Arithmetik ganzer Zahlen, in welchemdie Addition als einzige Operation hervortritt. *Comptes-rendus du I Congres des Mathematicians des Pays Slaves*, 1929.

[Sak09] Jacques Sakarovitch. *Elements of Automata Theory.* Cambridge University Press, New York, NY, USA, 2009.

[Sao95] Yannick Saouter. Halting problem for one-state turing machines. 1995.

[Sch80] Arnold Schönhage. Storage modification machines. *SIAM Journal on Computing*, 1980.

[Sha57] Claude E Shannon. A universal turing machine with two internal states. *Automata studies*, 1957.

[Sze88] Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Inf.*, 1988.

[Wei95] Jörgen W. Weibull. *Evolutionary Game Theory.* The MIT Press, 1995.

Titre : La puissance d'affaiblissements :
Ce qui est calculable avec Protocoles, Populations et Machines

Résumé:

Les Protocoles de Populations sont un modèle de calcul utilisant un nombre fini d'agents. Chaque agent est un outil avec une faible mémoire qui interagit par paire pour que la population calcule un résultat global. Des restrictions sur les règles d'interaction, et une considération sur le temps de calcul ont été explorées.

Dans cette thèse, on considère deux nouvelles restrictions sur les règles apportant un comportement plus rationnel aux agents. Les règles dites Pavloviennes font un pont avec la théorie des jeux. Les règles dites Confiantes permettent de donner une cohérence vis à vis des fréquences.

L'ajout d'outils aux agents des protocoles de population a été également étudié. Les Protocoles de Communauté correspondent à l'ajout d'uniques identifiants. Donner des Machines de Turing à chaque agent, c'est le modèle des Mobiles Passifs.

On s'intéresse à une restriction sur le temps avec le premier modèle. On voit aussi ce qu'il se passe si on ajoute de l'Homonymie aux identifiants. Pour le second modèle, on remplit une lacune à propos de l'espace de calcul qu'on offre à chaque agent. On s'intéresse également à ce qui peut être calculé si on donne à chaque agent une pile.

De manière orthogonale, on travaille sur une nouvelle variante des Machines de Turing : que se passe-t-il quand une machine ne peut avoir qu'un nombre fini de transitions charnières, quelle que soit la taille de l'entrée ? On donne quelques langages calculables, on prouve que l'arrêt est décidable et on fournit une majoration du Castor Affairé.

Summary:

Population Protocols is a model of computation over a finite set of agents. It corresponds to a model of devices with a weak memory interacting pairwisely to communicate and compute a global result. Restrictions over the interaction rules, and time consideration have been studied.

In this thesis, we consider two new restrictions over the rules that brings a more rational behaviour to agents: Pavlovian rules make a parallel with game theory and Trustful rules permit to add a coherence in terms of frequencies.

Adding tools to population protocols agents have been studied. Giving unique identifiers corresponds to Community Protocols, adding Turing Machines corresponds to Passively Mobile Model.

We provide time consideration to the first model, and then look at what happens when there can be Homonym agents. We fill a gap in the second model, depending on the computation space we provide to each agent. We also prove what can be computed if we give a stack to each agent.

Orthogonally, we consider a new variant of Turing Machines: What happens when Turing Machines can only have a finite number of pivotal transitions, whatever the size of the input is. We give some computable languages, prove that the halting problem can be decided and give a majoration of the Busy Beaver of the model.