Laboratoire de l'Informatique du Parallélisme

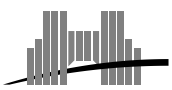
École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON
n° 5668

Distributed computing power : from local function to global computing

Laurent Bienvenu
Christophe Papazian

Février 2003

Research Report N° 2003-15



École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`

Distributed computing power : from local function to global computing

Laurent Bienvenu
Christophe Papazian

Février 2003

Abstract

We show here a natural extension of finite graph automata, by allowing each node of a network to store in its memory some pieces of information that are only bounded by the size of the underlying network (like a unique address). Depending of the power of the new local transition function, we show results about the power of the global function computed by the graph automata. The main result is that the global power is always less than the local computing power and even with very powerful local function (non recursive) we can not compute all global functions (even some primitive recursive ones) : **Distributed computing is limited by its own structure**.

Keywords: Graph automata, Computability, Distributed Algorithms

Résumé

1 Introduction

The main goal of this research is to understand the true power of distributed systems : what they can compute about their own structure, what properties they can recognize in their underlying network, what time is needed to compute or recognize some particular properties. Graph automata is very suitable for such a study because it is a simple and naturel way to modelize distributed computing like networks of computers in computer science, or complex dynamic systems in physics.

We show here some new results about non finite extensions of graph automata : we consider that each automaton in the network is no more finite but as a memory size linked to the size of the network. Those non finite extensions are motivated by two main reasons : as we study complexity and computability in the limits, we can here consider that some natural properties that we effectively use in practical life can be embedded in our model, like each node of the network having a unique address in its memory (implying its memory size to be greater than the logarithm of the size of the network). The second reason is to better understand why the computing power is weaker than a Turing machine : is it due to the limitation of each node (considered as a finite automaton) or a more structural limitation due to the fact that the computing is distributed in a network ?

We show here that distributed computing is limited by its own structural properties.

History of graph automata

Graph automata were first introduced by P. Rosenstiehl [7], under the name of *intelligent graphs*, surely because a network of finite automata is able to know some properties about its own structure. P. Rosenstiehl created some algorithms that find Eulerian paths or Hamiltonian cycles in those graphs, with the condition that every vertex has a fixed degree [8]. These algorithms are called "myopic" since each automaton has only the knowledge of the state of its immediate neighborhood.

A. Wu and A. Rosenfeld ([9] [10]) developed ideas of P. Rosenstiehl, using a simpler and more general formalism : the d-graphs. Hence, a graph automata is formed from synchronous finite automata exchanging information according to an adjacency graph. A. Wu and A. Rosenfeld gave a linear algorithm allowing a graph automata to know if its graph is a rectangle or not.

E. Rémila [5] extended this result to other geometrical structures like cylinders, torii or spheres. Then C. Papazian and E. Rémila show how graph automata can recognize some properties of regularity in their own structure by deciding if their underlying graph is a subgraph of an infinite regular network.

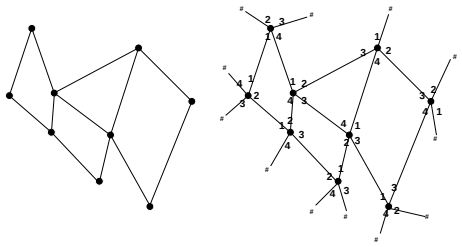


Figure 1: A d-graph (on right) and its underlying graph

2 Definitions

Notation : We will use d, i, j, k, n for integers, e, f for edges, and μ, ν for vertices, G for graphs, $\mathtt{A}$ for automata, and $\mathcal{L}$ for languages.

π_i^j is the projection function of arity j on the i^{th} input : $\pi_3^5(a, b, c, d, e) = c$. When arity is obvious, we do not show it.

For any finite set Q, we note Q^* the set of finite words on the alphabet Q.

2.1 Growing functions

For any growing function φ from $\mathbb{N}$ to $\mathbb{N}$, we can define φ^{-1} such as :

$$\varphi^{-1}(n) = \min\{i \mid \varphi(i) > n\}$$

Of course, if $\lim_{n\to\infty} \varphi(n)$ is infinite, then φ^{-1} is a total growing function from $\mathbb{N}$ to $\mathbb{N}$.

We note as usual $\varphi \in O(\psi)$ iff $\exists n, k \in \mathbb{N}\ \forall i \geq n\ (\varphi(i) \leq k.\psi(i))$
and $\varphi \in o(\psi)$ iff $\forall k \in \mathbb{N}\ \exists n \in \mathbb{N}\ \forall i \geq n\ (k.\varphi(i) \leq \psi(i))$

2.2 A d-graph

A graph is a pair $G = (V, E)$ where V is the set of vertices and E is the set of edges. E is a subset of V^2. We only consider graphs with no loops (for all $\nu \in V$ $(\nu, \nu) \notin E$), and symmetric (if $(\nu, \mu) \in E$ then $(\mu, \nu) \in E$). If V is finite, we say that the graph G is finite.

Let d be a fixed integer such that $d \geq 2$. A d-graph is a 3-tuple (G, ν_*, ρ), where $G = (V, E)$ is a symmetric connected graph with only two kinds of vertices : vertices of degree 1 (which are called #-vertices) and vertices of degree d; ν_* is a d-vertex of G which is called the leader, or the general; ρ is a mapping from E to $\{1, 2, ..., d\}$ such that, for each d-vertex ν of V, the partial mapping $\rho(\nu, .)$ is injective. The subgraph G' of G induced by the set of d-vertices is called the underlying graph of G.

From any graph G' of degree at most d, we can construct a d-graph (G, ν_0, ρ) whose underlying graph is G'. For each vertex, we call an up-edge, an edge that links this vertex to another vertex that is closer from the leader than itself.

2.3 Graph automata

Intuitively, a graph automata is a network of automata. We put a copy of the same automata on each vertex of a graph, and we obtain a synchronous dynamic system (similar to cellular automata) which is a natural model of networks. The leader vertex will initiate the process of recognizing.

A d-automaton is a triplet $\mathtt{A} = (Q, \Sigma, \delta)$ with :

- Q is a set of state (the memory of automaton) with 5 particular states : An accepting state q_A, a rejecting state $q_R \neq q_A$, an initial state q_I, a quiescent state q_0 and a "not here" state $q_\#$.
- Σ is a **finite** set of messages with particular messages : σ_I^i with $1 \leq i \leq d$ are the initial messages. σ_0 is the "do nothing" message. $\sigma_\#$ is the empty message.
- δ is the transition function from $Q \times \Sigma^d$ into $Q \times \Sigma^d$ such that
  - q_0 is quiescent $(\delta(q_0, (\sigma_0)^d) = (q_0, (\sigma_0)^d))$
  - $q_\#$ is unreachable $(\forall q, q' \in Q\ \forall \sigma, \sigma' \in \Sigma^d$ with $\delta(q, \sigma) = (q', \sigma')\ \ q \neq q_\# \Rightarrow q' \neq q_\#)$
  - $q_\#$ is greedy $(\forall q, q' \in Q\ \forall \sigma, \sigma' \in \Sigma^d$ with $\delta(q, \sigma) = (q', \sigma')\ \ q = q_\# \Rightarrow q' = q_\#)$
  - q_A and q_R are greedy.

We say that $\mathtt{A}$ is finite if Q is finite.

A graph automaton is a set $M = (G_d, \mathtt{A})$ with G_d a d-graph, and $\mathtt{A}$ a finite d-automaton. A configuration C of M is a mapping from the set of vertices of G_d to $Q \times \Sigma^d$, such that $C(\nu) = (q_\#, \sigma_\#^d)$ iff ν is a #-vertex.

We compute a new configuration from a previous one by applying the transition function δ simultaneously to each vertex of G_d, by reading its state and the messages previously computed by its neighbors : if we consider a vertex ν and its neighbors ν_i such that $\rho(\nu, \nu_i) = i$ then

$$C_{new}(\nu) = \delta\left(\pi_1(C(\nu)), \pi_{\rho(\nu_1,\nu)+1}(C(\nu_1)), ..., \pi_{\rho(\nu_d,\nu)+1}(C(\nu_d))\right)$$

We note $C \vdash C_{new}$.

Hence, we have a synchronous model, with local memory. The *initial* configuration C_I is a configuration where all vertices are quiescent, except the leader :

- $C_I(\nu_*) = (q_I, \sigma_I^1, ..., \sigma_I^d)$
- $C_I(\nu) = (q_0, (\sigma_0)^d)$ for all other non #-vertices
- $C_I(\nu) = (q_\#, (\sigma_\#)^d)$ for all #-vertices

We say that a configuration C is reachable if there is a finite sequence of configuration $(C_1, C_2, ..., C_n)$ such that $C_I \vdash C_1$, $C_i \vdash C_{i+1}$ and $C_n \vdash C$. We note $C_I \vdash^* C$.

A configuration C is accepting-terminal if $\pi_1(C) = q_A$ or C is rejecting-terminal if $\pi_1(C) = q_R$. Hence, a d-graph is accepted if $C_I \vdash^* C$ with C accepting-terminal and rejected with C rejecting-terminal (due to the greediness of q_A and q_R, this two facts exclude each other).

Now we want to link the memory size of the automata on our networks to the size of the networks. We have many good reasons to build such a link. A practical reason is to allow a theoretical study of networks algorithm where we are allowed to manipulate logarithm-size data into the memory of one of the nodes of the network. Such an hypothesis implies that the memory size of automata is at least greater than $k.\log(n)$ (with n the number of nodes in the network). A theoretical reason is to better understand the structure of network computing. What memory size do we really need to compute some recognizing tasks ? This article will answer to those questions.

Intuitively, a φ-family is a single automaton with growing memory size. When using a φ-family in a network, we use in fact the automata A_n in the family, where n is the number of vertices in the network. This is a modelisation of the fact that we assure each automaton on the network to be able to encode in its memory some particular pieces of information about the network, **whatever** *the size of the considered network is.*

A φ-family of d-automata is a set $\mathcal{F}_\Delta = (\mathtt{A}_i)_{i\in\mathbb{N}}$ such that :

- φ is a space-constructible growing function in $\mathbb{N}^\mathbb{N}$.
- $\mathtt{A}_i = (Q_i, \Sigma, \delta_i)$ where $Q_i = Q_0^{\varphi(i)}$ (Q_0 is a finite set of states) and $\delta_i = \Delta_{|Q_i \times \Sigma^d}$ with Δ function from $Q^* \times \Sigma^d$ into $Q^* \times \Sigma^d$. Δ is the global transition function of the family.

Hence, the function φ is the magnitude of the size of the memory of the automata.

A language L of graphs is recognize by a family $\mathcal{F}$ iff every graph of L with n vertices is recognized by $\mathtt{A}_n \in \mathcal{F}$, and $\mathtt{A}_n$ rejects all other graphs with n vertices.

So we have a computing model for a natural decision problem. Inputs are d-graphs, machines are the finite d-automata (if we consider only the classical model) or the φ-family (for the extended model), and answers are *Yes* the graph is recognized, *No* the graph is not recognized or no answer.

3 The classical model : $\varphi(n) = 1$

When we consider φ as a constant function, each φ-family corresponds to a single finite d-automaton. This is in fact the classical model of recognition on graph automata.

The main algorithmic problem of this model is that finite automata can not store in their memory an arbitrary number. So we can not give an address to each node of the network and we have to consider the space-time of the graph automata to make some computations dealing with the number of vertices or some structural properties of the underlying network.

Many results have been proved in this model and interested readers are invited to look at the bibliography to understand the basics of recognizing by finite automata ([9],[10],[5] and see [2] and [3] for more complex algorithms). We present here one of the most important (and easiest) algorithmic process in recognizing by graph automata : the linearisation. It allows to consider the set of vertices as a virtual tape (like a turing machine with one tape bounded by the size of the input for example), and it implies a canonical order on vertices (allowing intuitively to consider "`for each nodes do ...`" loops in our network algorithms).

3.1 Building a tree : the virtual tape

The best way to build the virtual tape is to build a breadth first search tree. First the leader vertex sends a signal `Build_Tree` to each of its neighbors. Then, the first time a vertex receives a signal `Build_Tree` from a neighbor ν it keeps into its memory that ν is its father (if it receives multiple signal `Build_Tree` at the same time, it can choose one, and ignores the others), and then sends the signal `Son` to its father and the signal `Build_Tree` to its others neighbors. When a vertex receives the signal `Son` from one of its neighbors μ, it keeps into its memory that μ is one of its son in the tree.

Hence, a depth first search in this tree implies a canonical order on vertices (without encoding address into their memory), and allows to consider the set of vertices ordered, like a tape.

As this process can be done with finite automata, it can be used we any φ-family. Hence, we can build a tape of size $n.k.\varphi(n)$ for any graph with n vertices.

4 The numbered model : $\varphi(n) = \log(n)$

Considering $\varphi(n) = \log(n)$ is a natural question. First, when each automaton has a memory of size $k.\log(n)$, we can store address in their memory and then, we can consider the graph automata like "real" networks. Moreover, we can encode the entire graph by adjacency lists in the entire memory, as the adjacency list is of size $n.\log(n)$ and we have n automata of size $\log(n)$: the virtual tape is now enough large to store the entire graph. log is the least function to have this property.

The fact that the set of messages is finite is not a restriction. It does not prevent an automaton from transfering all its (unbound) memory to one of its neighbors. One can just make the transfer of all bits one after the other by using messages.

4.1 The power of the global transition function Δ

If we consider any log-family, with Δ an arbitrary function in $\mathbb{N}^{\mathbb{N}}$, the model is of course more powerful than a Turing machine, as we could consider non recursive Δ.

Lemma 1 *There is a non recursive language recognized by a* log*-family*

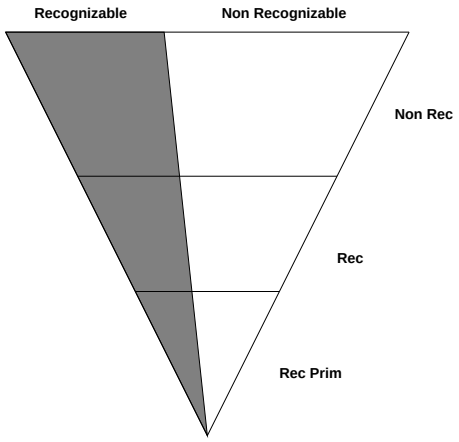


Figure 2: The class recognizable by a log-family

Proof. $\mathcal{K}$ is a non recursive language on $\mathbb{N}$. Consider the following language $\widetilde{\mathcal{K}}$ on the set of d-graphs : $\widetilde{\mathcal{K}} = \{G \mid G \text{ is a cycle of } n \text{ vertices and } n \in \mathcal{K}\ \}$.

$\widetilde{\mathcal{K}}$ is obviously non recursive, but can be recognized in the following way :

1. Every vertex verifies it has exactly two neighbors. If it is not the case, there is a reject message that is sent (in all directions) to the leader to reject the graph.
2. At the same time, the leader send its memory (in fact 1) to one of its neighbors. When a vertex receives the memory of one of its neighbors, it adds 1 and sends the result to its other neighbor, until the leader vertex receive n, which will be the size of the cycle.
3. We can then use the Δ transition to know if n is in $\mathcal{K}$ or not.

So $\widetilde{\mathcal{K}}$ is recognized by a log-family.

Corollary 1 *For any computability class $\mathcal{C}$, some language of $\mathcal{C}$ can be recognized by a* log*-family*

This result can be seen as the following figure. We saw that we can recognized complex languages. We will proof that we can not recognize all languages, even among the simplest. So this model defines some transversal class of recognition.

5 The complete model : $\varphi(n) = n.\log(n)$

If we now consider $\varphi(n) = n.\log(n)$, we will be able to encode the entire graph (for example, by adjacency lists) into the memory of a single automaton (the leader). As we will see, $n.\log(n)$ is the least function that have this property and it will imply the total powerfulness of the class of $n.\log(n)$-family.

Lemma 2 *The number of connected graphs (with n vertices) of degree 3 is ultimately greater than $n^{\frac{n}{5}}$.*

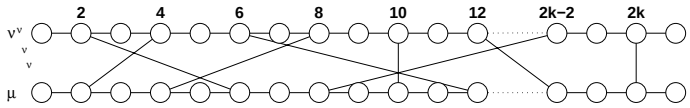


Figure 3: Example of G_σ

Proof. We build a family $\mathcal{F}$ of graphs that has enough large cardinality. Consider the graphs $G_\sigma = (V, E)$ ($n = |V|$) such that

$$V = \left\{\nu_1, \nu_2, ..., \nu_{\lfloor \frac{n}{2} \rfloor}, \mu_1, \mu_2, ..., \mu_{\lceil \frac{n}{2} \rceil}\right\}$$

$$E = \left\{(\nu_i, \nu_{i+1}) \mid 1 \leq i < \left\lfloor \frac{n}{2} \right\rfloor\right\} \cup \left\{(\mu_i, \mu_{i+1}) \mid 1 \leq i < \left\lceil \frac{n}{2} \right\rceil\right\} \cup \left\{(\nu_{2i}, \mu_{2\sigma(i)}) \mid 1 \leq i \leq \left\lfloor \frac{n}{4} \right\rfloor\right\}$$

where σ is a permutation of $\{1, ..., \lfloor \frac{n}{4} \rfloor\}$.

It is easy to see that each graph of this family can not be isomorphic to more than one another. So there is at least $\frac{1}{2}(\frac{n}{4})!$ graphs with n vertices. This is ultimately larger than $n^{\frac{n}{5}}$.

Corollary 2 *There is a constant k_G such that we need $k_G.n.\log(n)$ bits to encode a d-graph with n vertices (and $k_G \geq \frac{1}{5}$).*

As we can encode the entire underlying network into the memory of a single automaton, we can deduce the following result :

Lemma 3 *All languages can be recognized by a $n.\log(n)$-family*

Proof. Just build a depth first search tree, create a numbering of the vertices and each vertex sends to the leader the adjacency list of its neighbors. Then use Δ to decide if the graph is in the language.

But we can extend the previous lemma into the following equivalence

Theorem 1 *φ is a growing function from $\mathbb{N}$ into $\mathbb{N}$. All languages can be recognized by a φ-family iff $n\log(n) \in \mathcal{O}(\varphi(n))$.*

Proof.

If $n\log(n) \in \mathcal{O}(\varphi(n))$, using the lemma 3, we can immediately deduce that all languages can be recognized by a φ-family.

In the other way, if we suppose $n\log(n) \notin \mathcal{O}(\varphi(n))$, we have to prove that there is a primitive recursive language that is not recognized by any φ-family.

The number of possible transition functions for automata with a memory size less than $\lambda.\varphi(n)$ is $2^{\left(\lambda.\varphi(n).2^{\lambda.\varphi(n)}\right)}$.

Using the lemma 2, we know that we have at least $2^{n^{\frac{n}{5}}}$ different parts of d-graphs with n vertices. But, using hypothesis, we have for any λ (with in fact $\lambda = \log(|Q_0|)$), $2^{n^{\frac{n}{5}}} \notin \mathcal{O}(2^{\left(\lambda.\varphi(n).2^{\lambda.\varphi(n)}\right)})$. So, for any λ and any n_0 there is a $n_1 > n_0$ such that "there is a (finite) set of d-graphs (with n_1 vertices) that is not recognized by any φ-family with memory size less than $\lambda.\varphi(n)$" (this is the $\diamond$ property).

So we recursively build a language $\mathcal{L} = \bigcup L_i$ such that $\lambda = 1$ and $n_0 = 1$ we obtain n_1 and L_1 (that is not recognized by any automaton with memory

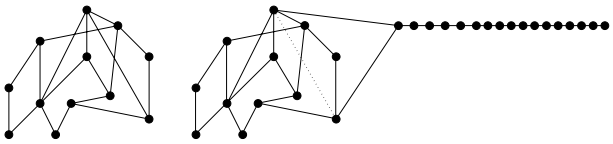


Figure 4: Transformation Θ : On the left G, on the right an element of $\Theta(G)$

size less than $1.\varphi(n)$), then having L_i and n_i we obtained (by $\diamond$ property using $\lambda = i + 1$) L_{i+1} and n_{i+1}.

$\mathcal{L}$ is not recognized by any φ-family, because for any φ-family, there is a λ such that the memory of the automaton is less than $\lambda.\varphi(n)$, so it does not recognize $L_{x>\lambda}$.

Corollary 3 *If $\varphi(n) \in o(n \log(n))$ is a growing primitive recursive function there is a primitive recursive language $\mathcal{L}$ that is not recognized by any φ-family.*

Proof. By using hypothesis, there is $k \in \mathbb{N}$ such that $\lambda_n = \frac{n.\log(n)}{k.\varphi(n)}$ and for all n we have $2^{\left(\lambda_n.\varphi(n).2^{\lambda_n.\varphi(n)}\right)} < 2^{n^{\frac{n}{5}}}$ .Then the recursive method we give to build $\mathcal{L}$ is primitive : For any size n, just try all possible transition function and find one L_n that is not recognized by a φ-family with memory size less than $\lambda_n.\varphi$. As $\lim_{n\to\infty} \lambda_n = +\infty$, we effectively build a non recognizable primitive recursive language.

Finally, we show that we can precise the structure of the computational power of the φ-families for $\varphi \in o(n.\log(n))$, and define a hierarchy of network recognizing.

Theorem 2 (Theorem of network recognizing hierarchy) *Let φ and ψ be two growing function in $o(n.\log(n))$ such that $\varphi \in o(\psi)$. Let $\mathcal{C}_\psi$ (respectively $\mathcal{C}_\varphi$) the class of languages that can be recognized by one ψ-family (respectively a φ-family). Then $\mathcal{C}_\varphi \subsetneq \mathcal{C}_\psi$.*

Proof. We will build a language that will be recognizable by a ψ-family and that will be not recognizable by any φ-family.

We first define a transformation Θ from languages of graphs into languages of graphs such that $\Theta(\bigcup A_i) = \bigcup \Theta(A_i)$ and $\Theta(\{G = (V, E)\})$ is the set of graphs $G' = (V', E')$ such that

$\S_1$ $V' = V \cup W$ with $W = \{w_1, w_2, ..., w_m\}$.

$\S_2$ $|V'| \geq \max[\ (2|V| + 1)\ ,\ (\psi^{-1}(|V| \log |V|))\ ]$.

$\S_3$ $E' = (E \setminus \{(\nu, \nu')\}) \cup \{(\nu, w_1), (\nu', w_1)\} \cup \{(w_i, w_{i+1})\}$ with (ν, ν') an arbitrary edge of V.

Informally, we take a graph and add a long band such that we can effectively encode the graph into a memory size of $\psi(|V'|) \geq |V| \log |V|$. Due to condition $\S_2$, we can easily find where is the band and where is the original graph, as $|W| > |V|$.

So the process of recognizing $\Theta(\mathcal{L})$, for any language $\mathcal{L}$, by a ψ-family is as follow :

- First we have to find where is w_1. We can use the linear time bridge detection algorithm on graph automata as explained in [10]. Then verify, for each bridge, that we have on one side a branch that is longer that the size of the other side. This can be easily done in linear time by finite automata. If we don't find such a branch, the graph is rejected. By this first step of detection we verify $\S_1$, $\S_3$ and $|W| > |V|$.
- Then, each vertex of V will send its adjacency list to the leader (assuming that ν and ν' the neighbors of w_1 in V are neighbors). If the leader can not store in its memory the lists of adjacency of the entire graph, it means that we do not have $|V'| \geq \psi^{-1}(|V| \log |V|)$, and the graph is rejected.
- As all the graph is now encoded into the memory of the leader, just apply properly defined Δ to know if the graph is in $\mathcal{L}$.

So $\Theta(\mathcal{L})$ is recognizable by some ψ-family. But, for the same cardinality reasons as the proof of theorem 1, $\Theta(\mathcal{L})$ will not be recognizable by any φ-family.

In fact, as we can see from the proof, the transformation Θ is not useful by adding vertices to the graph (to allow more global memory size), but by locally increasing the size of the memory of each automaton. Hence, if we consider graph automata with finite automata, such a transformation is useless.

6 To conclude

At last, we ca say that the local power (computational power of Δ) is never entirely transmitted to the global computational power of the network, as the only case where the global power equals the local power is when we can centralize all pieces of information about the network into a single node.

The main result of this research is the following idea : To allow the detection of more properties in a network, you have to add memory to at least one node (the leader), and it's useless to add more nodes.

So, we can represent the structure of recognizable classes by the following figure. If you only consider recursive local transition function Δ, the figure is the same with removing the top part of non recursive functions.

Hence, some properties on graphs, even among the simplest, resist to be recognized by distributed computing.

References

[1] C. Papazian, E. Rémila, *Graph Automata Recognition*, Research report, LIP (1999)

[2] C. Papazian, E. Rémila, *Linear Time Recognizer for Subsets of* $\mathbb{Z}^2$, Proceedings of Fundamentals of Computation Theory, Springer LNCS 2138, **FCT** 2001, 400-403

[3] C. Papazian, E. Rémila, *Hyperbolic Recognition by Graph Automata* International Colloquium on Automata, Languages and Programming, Springer LNCS 2380, **ICALP** 2002, 330-342

[4] J. Mazoyer, C. Nichitiu, E. Rémila, *Compass permits leader election*, Proceeding of **SODA**, 948-949, (1999)

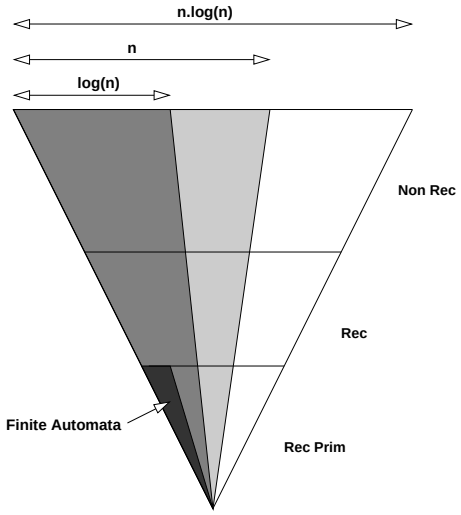


Figure 5: Structure of recognizable classes by φ-families

[5] E. Rémila, *Recognition of graphs by automata*, Theoretical Computer Science **TCS** 136, 291-332, (1994)

[6] E. Rémila, *An introduction to automata on graphs*, Cellular Automata, M. Delorme and J. Mazoyer (eds.), Kluwer Academic Publishers, Mathematics and Its Applications 460, 345-352, (1999).

[7] P. Rosensthiel, *Existence d'automates finis capables de s'accorder bien qu'arbitrairement connectés et nombreux*, Internat. Comp. Centre 5, 245-261 (1966).

[8] P. Rosensthiel, J.R Fiksel and A. Holliger, *Intelligent graphs: Networks of finite automata capable of solving graph problems* , R. C. Reed, Graph Theory and computing, Academic Press, New-York, 210-265 (1973).

[9] A. Wu, A. Rosenfeld, *Cellular graph automata I*, Information and Control 42, 305-329, (1979).

[10] A. Wu, A. Rosenfeld, *Cellular graph automata II*, Information and Control 42, 330-353, (1979).