On automata networks dynamics : an approach based on computational complexity theory

Martín Ríos Wilson

Aix*Marseille université
Initiative d'excellence

THÈSE DE DOCTORAT

Soutenue à la Universidad de Chile, Santiago, Chili
dans le cadre d'une cotutelle avec la Universidad de Chile
le 31 mai 2021 par

Martín Ríos Wilson

On automata networks dynamics : an approach based on computational complexity theory

Discipline
Informatique

École doctorale
ED 184 Mathématiques et informatique

Laboratoire
Laboratoire d'informatique et systèmes (LIS)

LIS
LABORATOIRE D'INFORMATIQUE & SYSTÈMES
UMR 7020

Composition du jury

Enrico Formenti Professeur des universités, Univ. Côte d'Azur	Rapporteur
Eric Goles Professeur des universités, Univ. Adolfo Ibañez	Co-encadrant
Jarkko Kari Professeur des universités, Univ. Turku	Rapporteur
Alejandro Maass Professeur des universités, Univ. Chile	Co-directeur
Ivan Rapaport Professeur des universités, Univ. Chile	Examinateur
Sylvain Sené Professeur des universités, Univ. Aix-Marseille	Co-directeur
Véronique Terrier Maître de conférences, Univ. Caen	Examinatrice
Guillaume Theyssier Chargé de recherche, CNRS (Marseille)	Co-encadrant

UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MATEMÁTICA

ON AUTOMATA NETWORKS DYNAMICS: AN APPROACH BASED ON COMPUTATIONAL COMPLEXITY THEORY

TESIS PARA OPTAR AL GRADO DE
DOCTOR EN CIENCIAS DE LA INGENIERÍA, MENCIÓN MODELACIÓN MATEMÁTICA
EN COTUTELA CON AIX-MARSEILLE UNIVERSITÉ

MARTÍN ALONSO FACUNDO RÍOS WILSON

PROFESORES GUÍA:
ALEJANDRO MAASS SEPÚLVEDA
SYLVAIN SENÉ

PROFESORES CO-GUÍA:
ERIC GOLES CHACC
GUILLAUME THEYSSIER

MIEMBROS DE LA COMISIÓN:
ENRICO FORMENTI
JARKKO KARI
IVAN RAPAPORT ZIMERMANN
VÉRONIQUE TERRIER

Este trabajo ha sido parcialmente financiado por ANID-BECA DOCTORADO NACIONAL 2018- FOLIO N° 21180910, CMM ANID PIA AFB170001 and ANR project ANR-18-CE40-0002

SANTIAGO DE CHILE
2021

I, undersigned, Martín Ríos Wilson, hereby declare that the work presented in this manuscript is my own work, carried out under the scientific direction of Alejandro Maass and Sylvain Sené, and the co-supervision of Eric Goles and Guillaume Theyssier, in accordance with the principles of honesty, integrity and responsibility inherent to the research mission. The research work and the writing of this manuscript have been carried out in compliance with both the french national charter for Research Integrity and the Aix-Marseille University charter on the fight against plagiarism.

According to the cotutelle agreement, this work has also been submitted to the *Universidad de Chile*, but has not been submitted previously either in this country or in another country in the same or in a similar version to any other examination body.

Santiago, Chile, the 2nd of April 2021.

Résumé

Un réseau d'automates (RA) est un réseau d'entités (les automates) en interaction. Ces automates ont un nombre fini d'états possibles et sont reliés les uns aux autres par une structure de graphe appelée graphe d'interaction. Chaque automate évolue au cours du temps discret en fonction des états de ses voisins dans le graphe d'interaction, ce qui définit un système dynamique. Ce travail de thèse explore deux questions principales : a) quel est le lien entre les propriétés dynamiques et calculatoires d'un RA ? et b) quel est l'impact de la topologie du graphe d'interaction sur la dynamique globale d'un RA ?.

Pour aborder la première question, une notion de complexité calculatoire est définie au regard de problèmes de décision liés à la dynamique des RA. De même, une notion de complexité dynamique est définie en termes de l'existence d'attracteurs de période exponentielle. Un lien fort entre ces deux définitions est présenté qui met en exergue le concept de simulation entre familles de RA. Dans ce contexte, la complexité se caractérise d'un point de vue localisé en étudiant l'existence de structures appelées gadgets qui satisfont deux propriétés : i) ils peuvent interagir localement de manière cohérente comme des systèmes dynamiques et ii) ils sont capables de simuler un ensemble fini de fonctions définies sur un ensemble fini.

La deuxième question est quant à elle abordée dans le contexte des RA "freezing". Un RA est "freezing" s'il y a un ordre sur les états de telle sorte que l'évolution de l'état de n'importe quel automate ne diminue pas quelle que soit l'orbite. Un problème général de model-checking capturant de nombreux problèmes de décision classiques est présenté. De plus, lorsque trois paramètres de graphe, le degré maximum, la largeur arborescente et la taille de l'alphabet sont bornés, un algorithme parallèle efficace résolvant le problème mentionné est donné. De plus, il est montré que ce problème est peu susceptible d'être FPT (fixed-parameter tractable) lorsque le paramètre de largeur arborescente ou celui de taille de l'alphabet sont considérés comme unique paramètre.

Abstract

An automata network (AN) is a network of entities, each holding a state from a finite set and related by a graph structure called an interaction graph. Each node evolves according to the states of its neighbors in the interaction graph, defining a discrete dynamical system. This thesis work explores two main questions : a) what is the link between dynamical and computational properties of an AN? and b) what is the impact of the interaction graph topology on the global dynamics of an AN?.

In order to tackle the first question a notion of computational complexity of an AN family is defined in terms of the computational complexity of decision problems related to the dynamics of the network. On the other hand, dynamical complexity of a particular AN family is defined in terms of the existence of attractors of exponential period. A strong link between these two last definitions is presented in terms of the notion of simulation between AN families. In this context, complexity is characterized from a localized standpoint by studying the existence of structures called coherent gadgets which satisfy two properties : i) they can locally interact in a coherent way as dynamical systems and ii) they are capable of simulating a finite set of functions defined over a fixed finite set.

Finally, the second question is addressed in the context of a well-known family called freezing automata networks. An AN is freezing if there is an order on states such that the state evolution of any node is non-decreasing in any orbit. A general model checking problem capturing many classical decision problems is presented. In addition, when three graph parameters, the maximum degree, the treewidth and the alphabet size are bounded, a fast-parallel algorithm that solves general model checking problem is presented. Moreover, it is shown that the latter problem is unlikely to be fixed-parameter tractable on the treewidth parameter as well as on the alphabet size when considered as single parameters.

Publications

The present work is based in the following articles:

- **Journals**
  - **2021.** On the Complexity of Asynchronous Freezing Cellular Automata, Eric Goles, Diego Maldonado, Pedro Montealegre, Martín Ríos Wilson, Information and Computation,104764, ISSN 0890-5401.
  - **2021.** Generating Boolean functions on totalistic automata networks, Andrew Adamatzky, Eric Goles, Pedro Montealegre, Martín Ríos Wilson, International Journal of Unconventional Computing, (to appear).
  - **2020.** On the effects of firing memory in the dynamics of conjunctive networks, Eric Goles, Pedro Montealegre, Martín Ríos Wilson, Discrete and Continuous Dynamical Systems - A, 40(10): 5765-5793.
- **Preprints**
  - **2021.** On Symmetry versus Asynchronism: at the Edge of Universality in Automata Networks, Martín Ríos Wilson, Guillaume Theyssier, arXiv:2105.08356. https://arxiv.org/abs/2105.08356
  - **2020.** On the impact of treewidth in the computational complexity of freezing dynamics, Eric Goles, Pedro Montealegre, Martín Ríos Wilson, Guillaume Theyssier, arXiv:2005.11758. https://arxiv.org/abs/2005.11758
- **Conferences**
  - **2021.** On the impact of treewidth in the computational complexity of freezing dynamics, Eric Goles, Pedro Montealegre, Martín Ríos Wilson, Guillaume Theyssier, Computability in Europe 2021: Connecting with Computability. Accepted.
  - **2019.** On the effects of firing memory in the dynamics of conjunctive networks, Eric Goles, Pedro Montealegre, Martín Ríos Wilson, International Workshop on Cellular Automata and Discrete Complex Systems (pp. 1-19). Springer, Cham.
- **Book chapters**
  - **2021.** Computing the Probability of Getting Infected: On the Counting Complexity of Bootstrap Percolation, Pedro Montealegre and Martín Ríos Wilson. In "Automata and Complexity: Essays presented to Eric Goles on the occasion of his 70th birthday" (to appear).

Contents

List of Tables

List of Figures

Introduction

An automata network is a collection of n entities, called automata (and sometimes also referred as nodes or vertices), which are somehow related. These relations between the different entities are represented by a graph structure and are defined in this document by local functions which take values in a finite set, usually denoted by Q and called the alphabet. From a global standpoint, these local functions can be seen as a global function acting on set Q^n and defining a dynamical system (which can be deterministic or not deterministic). The most common and natural way to define the dynamics of an automata network is just considering deterministic local rules for each node in the network. However, it is also interesting to consider the non-deterministic case, which also includes probabilistic and asynchronous models. In particular, a way to introduce asynchronism consists in updating only some of the nodes in the network at a time step and leave the state of the others unchanged. Then, for each time step, there exists an assignation of nodes that will update their states at this step. Such assignations define an *update scheme*.

Observe that, the automata network model can be considered as a *topological non-uniform* generalization of (finite) cellular automata. Automata networks have been used as modelling tools in many areas [36] and they can also be considered as a distributed computational model with various specialized definitions like in [79, 80].

Generally speaking, two main research lines can be identified in the context of the study of this model. The first one is related to the dynamics that an automata network can have according to the updating modes. It focuses particularly on analyzing their richness in terms of, for instance, the period of attractors compared to the size of the network and, in this sense, the role that graph topology plays in restricting or allowing complex dynamical behavior. The second one is more focused on the computation abilities of the network. In this approach, decision problems related to the dynamics of the network are proposed. A very interesting one, that we study in this thesis, is called the *prediction problem*. Roughly speaking, one would want to see if it is possible to predict the dynamical behavior of the network. More precisely, a particular node in the network is fixed and an initial configuration is given for the rest of the nodes in the network. We would like to know if at some given time step t (or maybe if eventually) the state of this node will change or if it will remain as in the initial configuration. Of course, there is always a trivial solution to this problem, which consists in simply simulating the network for t time steps (or if there is no given time limit, we simulate until reaching an attractor) and see if our objective node has changed. Thus, a natural question in this context is if we can *beat* simulation in someway by finding an algorithm that is more efficient than simply simulating. Generally speaking, if the dynamics is extremely

difficult to predict (there is no way of beating simulation) the system is complex. Contrarily, if there exists some particular algebraic or graph topology property which we can exploit to design an algorithm to decide prediction problem in an efficient way (compared to the trivial solution) then, the system is not complex. Within this framework, an interesting observation concerning local dynamical behavior arises. Observe that there are some similarities between prediction problem and, in the context of Boolean circuit families, the well-known circuit value problem. This problem consists in, given a Boolean circuit and an assignation of its inputs, evaluating the circuit and read some fixed output value. In this sense, in many study cases we would like to show that there exists a suitable reduction from this problem to the prediction problem. In fact, given some class of automata networks (which can be defined by a particular type of rule) this task is somehow translated to the task of simulating in some coherent way the gates of the circuit. This is achieved by using some particular collection of subnetworks in the class that *implements* (understanding this as some sort of simulation implemented using its dynamics) each different type of gate in the circuit and which can be *glued* together in a coherent way so that the resulting *glued* module simulates the circuit evaluation process.

The main aim of this thesis is twofold: on the one hand to provide a general theoretical framework which would allow to unify both research lines by exhibiting a link between computational capabilities of an automata network and its dynamical behavior; on the other hand to study how the graph structure of a network can impact the dynamics of the latter. In particular, we face this task through two main research questions:

1. What is the link between dynamical and computational properties of an automata network?

2. What is the impact of the interaction graph topology in the global dynamics of an automata network?

In order to answer these questions, we start by establishing a general framework for automata networks, including what is precisely a family of automata networks and how to represent it in a concrete way. We explore a general way to combine different automata networks in order to construct a new network preserving dynamical properties from the networks used in this combination. This framework allows us to provide sufficient conditions to deduce global dynamical and computational properties for a particular family by focusing in studying local structures called *gadgets*. In addition, inspired by the intrinsic simulation framework for cellular automata, we compare the capabilities of different automata network families by developing a definition for simulation between families. More precisely, we develop a simulation scheme based on an injective assignment of states in the simulated network to a subgraph in the simulator. We call these subgraphs *blocks* and we represent each node the simulated network by one of this subgraphs in the simulator. Then, the dynamics of the simulated network is simulated (possibly several time steps represent a single time transition on the simulated network dynamics) by this latter injective coding in each block. This latter definition of simulation, does not only allow us to transfer dynamical properties from the simulated network to the simulator but also help us to deduce computational complexity results regarding the prediction problem in the simulator by studying the simulated network. This is possible because our definition of simulation is compatible with standard polynomial and logspace reductions. We go further in this context by stating the concept of universality

for automata networks, expressed as the capability to efficiently simulate families of automata networks. These families can be described by a family of circuits, each one representing an automata network in this family. We ask these circuit families to satisfy that the maximum number of gates in a circuit grows bounded by a polynomial function on the size of the represented network. In particular, we study the case of automata networks in which its graph has a maximum degree bounded by a constant (not depending on the size of the network) and we propose a stronger definition of universality related to the capability of simulating this latter family. We then, apply our framework to a concrete case of study, composed by different families of automata networks and different *update schemes*. Roughly, we consider a pool of automata network families sharing a common set of properties and we establish a partial hierarchy in the set of families given by different constraints. In parallel, we have also a hierarchy in update schemes. We observe an interesting trade-off between the constraints we can apply in a particular family and the constraints we can apply related to the update scheme we are considering. Particularly, we observe that families with more constraints equipped with general update schemes are universal, but they can lose this property if we apply constraints in the type of update scheme we are considering and complementarily, if we start with more general families, we can observe universality when family is equipped with more restricted update schemes.

Then, we tackle the second question by considering a particular family of automata network, called *freezing automata networks*. An automata network is *freezing* if there is an order on states such that the state evolution of any node is non-decreasing in any orbit. In this part of the thesis, we go beyond the deterministic definition of automata networks and we study the non-deterministic case in the context of freezing networks. We define in this context a generalization of the prediction problem called the *specification* problem which also includes some other decision problems related to the dynamics of an automata network. Then, we explore the impact of the graph topology on the dynamics by identifying key graph parameters which allow us to design efficient algorithms for solving the latter problem. We address this latter challenge from a parametric complexity approach. Finally, we study the particular version of bootstrap percolation model, consisting in a freezing version of the well-known majority automata. In this latter network, each node takes the state of the majority of its neighbors in the graph. We see this model as a disease spreading model and we study the complexity of the functional problem consisting in computing the probability of some node to get infected. We then establish that in general the problem is hard since a polynomial time algorithm to compute probability is unlikely to exist. Then, we give sufficient conditions on the graph in order to assure that probability can be computed in polynomial time.

Related work

There is a wide amount of literature for the prediction problem at sake in this thesis work [46, 61, 37, 34, 39, 38] in which the computational complexity of the latter problem have been studied for cellular automata and different automata network families such as: the freezing version of the well-known game of life called, Life without death, and threshold networks among which are majority networks and conjunctive networks.

Moreover, the notion of intrinsic universality in cellular automata have been addressed in several papers [49, 8, 60, 64, 65] and it is based on the concept of *sub-automaton* which

consists in a restriction on the set of states and the concept of *rescaling* which consists in packing cells. Roughly speaking, a cellular automaton simulates another one if some rescaling of the first one contains a rescaling of the second one as a sub-automaton. In this sense, a cellular automaton is *intrinsically universal* if for any arbitrary other automaton there exists some rescaling of the first which contains it as a sub-automaton.

Regarding freezing automata networks, several models that received a lot of attention in the literature are actually freezing automata networks, for instance: bootstrap percolation which has been studied on various graphs [1, 6, 5, 48], epidemic [25] or forest fire [4] propagation models [1], cristal growth models [75, 43] and more recently self-assembly tilings [78]. Moreover, their complexity as computational models has been studied from various standpoints: as language recognizers where they correspond to bounded change or bounded communication models [77, 56, 13], for their computational universality [66, 31, 9], as well as for various associated decision problems [33, 35, 41, 34].

A major topic of interest in automata networks theory is to determine how the network graph affects dynamical or computational properties [26, 41]. In the freezing case, it was for instance established that one-dimensional freezing cellular automata, while being Turing universal via Minsky machine simulation, have striking computational limitations when compared to bi-dimensional ones: they are NL-predictable (instead of P-complete) [66, 46, 77], can only produce computable limit fixed points starting from computable initial configurations (instead of non-computable ones starting from finite configurations) [66], and have a polynomial time decidable nilpotency problem (instead of uncomputable) [66].

Regarding specifically the complexity of the dynamics of bootstrap percolation, we present the following related work: in [41] it is studied the Stability problem which is the decision problem consisting in deciding, given a graph and an initial condition, whether an objective node reaches state 1 in some time-step, when the states evolve according to the synchronous bootstrap percolation dynamics. This problem is solvable in polynomial time, since the dynamics reaches a fixed point in a linear number of synchronous time-steps. Interestingly, in [41] is shown that in graphs of degree at most 4 the problem is in class **NC**, which is the subclass of **P** containing all problems that can be efficiently solved by a parallel machine. Moreover, in graphs of maximum degree at least 5 the Stability problem is **P**-Complete, meaning that there is no algorithm solving the problem better than simply simulating the dynamics until a fixed point is reached.

Later, as we mentioned before, in [38] the authors study the complexity of bootstrap percolation on asynchronous updating schemes. To do so, two decision problems are defined. The first one is Good-Sequence (which is called asynchronous prediction in [38]). The second one is called Asyncronous Stability, and is an asynchronous version of the stability problem. The difference between the two problems is that Good-Sequence asks for the existence of a good-sequence for a given time-span, while Asynchronous Stability asks for the state of the node once the attractor is reached. In that article, it is shown that given an initial condition, the dynamics of bootstrap percolation reaches the same fixed point for every updating scheme. That makes Asyncronous Stability equivalent to Stability.

[1] They are discrete counterparts of the family of spatial SIR/SEIR models [53] (or other variants) which are sadly famous amid the actual COVID-19 pandemic.

Contrarily, problem Good-Sequence is solvable in polynomial time when restricted to graphs maximum degree 4, but **NP**-Complete restricted to graphs of maximum degree 5.

Chapter structure

This thesis work is divided in six chapters. The first chapter contains basic definitions and results that are used to derive main results of this work. This includes some elements of computational complexity and graph theory and some related results in the dynamics of automata networks as well as the definition of some particular families of automata networks that we will study.

The second chapter develops the main framework on computational complexity and decision problems related to automata networks. It starts with a precise definition on the representation of different automata network families. Then, it continues with the development of a general simulation background. This chapter goes beyond this latter point by exploring the notion of universality in the context of the latter definition.

The third chapter presents the framework about gadgets and how the existence of local structures has an impact on the global dynamics of automata networks. In first place, a family of bounded degree automata networks, $\mathcal{G}$-networks, is presented. Then, the notion of gadget is introduced and a scheme for glueing automata networks is developed. Finally, some useful $\mathcal{G}$-networks are introduced and studied in order to illustrate the relation between universality and the richness of the dynamics of some families in terms of periods and transients.

The fourth chapter gives a framework about update schemes. First, basic definitions on update schemes are provided. Then, the notion of projection system and asynchronous extension is introduced as a way to obtain a general formalism to study asynchronous dynamics.

The fifth chapter is a case of study in which the dynamics of different families of automata networks is studied under different update schemes. First, some of the main results are stated and then chapter is divided in sections, each one devoted to the study of one particular family. Each section is divided in subsections, one of each devoted to the study of the dynamics under one particular update scheme. Then, in each of these sections, the question of whether that family equipped with that specific update scheme is universal is addressed.

Finally, the sixth chapter is devoted to the study of freezing automata networks and it is divided in two main sections: the first one consists in analyzing the effect of graph topology on the dynamics of freezing automata networks by studying a generalization of the prediction problem; the second one is focused on studying a concrete model consisting in the study of the complexity of the dynamics of a non-deterministic variant of the bootstrap percolation model.

Chapter 1

Preliminaries

1.1 Automata networks

In this section, we give the elementary definitions related to automata network theory. We start by considering some elemental results in graph theory and then, we define the concept of automata networks and related technical definitions.

1.1.1 Elements of Graph Theory

A *graph* is a pair $G = (V, E)$ where V and E are finite sets satisfying $E \subseteq V \times V$. We will call V the set of *nodes* and the set E of *edges*. We call $|V|$ the *order* of G and we usually identify this quantity by the letter n. Usually, as E and V are finite sets we will implicitly assume that there exists an ordering of the vertices in V from 1 to n (or from 0 to $n-1$). Sometimes we will denote the latter set as $[n]$. If $G = (V, E)$ and $V' \subseteq V, E' \subseteq E$ we say that G' is a *subgraph* of G. We call a graph $P = (V, E)$ of the form $V = \{v_1, \ldots, v_n\}$, $E = \{(v_1v_2), \ldots, (v_{n-1}, v_n)\}$ a *path graph*, or simply a *path*. We often refer to a path by simply denoting its sequence of vertices $\{v_1, \ldots, v_n\}$. We denote the *length* of a path by its number of edges and if $P = (V, E)$ with $V = \{v_1, \ldots, v_n\}$, we call any v_k such that $1 < k < n$ an internal node of P. In addition, if $A, B \subseteq V$ are node sets, we say that P is a A-B path if $P = (V, E)$ is a path such that $V = \{v_1, \ldots, v_n\}$ and $v_1 \in A$, $v_n \in B$ and no internal node lies in A nor in B. If $A = \{v\}$ and $B = \{w\}$ then, we call P a v-w path. Whenever $P = (V = \{v_1, \ldots, v_n\}, E = \{(v_1v_2), \ldots, (v_{n-1}, v_n)\}$ is a path we call the graph in which we add the edge $\{v_n, v_1\}$ a *cycle graph* or simply a *cycle* and we denote it C where $C = P + \{v_n, v_1\}$. Analogously, a cycle is denoted usually by a sequence of nodes and its length is also given by the amount of edges (or vertices) in the cycle. Depending of the length of C we call it a k-cycle when k is its length. A non-empty graph is called *connected* if any pair of two vertices u, v are linked by some path. Given any non-empty graph, a maximal connected subgraph is called a connected component.

We call *directed graph* a pair $G = (V, E)$ together with two functions init : $E \to V$ and ter : $E \to V$ where each edge e $\in E$ is said to be directed from init(e) to ter(e) and we write e $= (u, v)$ whenever init(e) $= u$ and ter(e) $= v$. There is also a natural extension

of the definition of paths, cycles and connectivity for directed graphs in the obvious way. We say a directed graph is strongly connected if there is a directed path between any two nodes. A strongly connected component of a directed graph $G = (V, E)$ is a maximal strongly connected subgraph.

Given a (non-directed) graph $G = (V, E)$ and two vertices u, v we say that u and v are neighbors if $(u, v) \in E$. Remark that abusing notations, an edge (u, v) is also denoted by uv. Let $v \in V$, we call $N_G(v) = \{u \in V : uv \in E\}$ (or simply $N(v)$ when the context is clear) the set of neighbors (or *neighborhood*) of v and $\delta(G)_v = |N_G(v)|$ to the *degree* of v. Observe that if $G' = (V', E')$ is a subgraph of G and $v \in V'$, we can also denote by $N_{G'}(v)$ the set of its neighbors in G' and the degree of v in G' as $\delta(G')_v = |N_{G'}(v)|$. In addition, we define the *closed neighborhood* of v as the set $N[v] = N(v) \cup \{v\}$ and we use the following notation $\Delta(G) = \max_{v \in V} \delta_v$ for the *maximum degree* of G. Additionally, given $v \in V$, we will denote by E_v its set of *incident edges*, i.e., $E_v = \{\mathrm{e} \in E : \mathrm{e} = uv\}$. We will use the letter n to denote the order of G, i.e. $n = |V|$. Also, if G is a graph whose sets of nodes and edges are not specified, we use the notation $V(G)$ and $E(G)$ for the set of vertices and the set of edges of G respectively. In the case of a directed graph $G = (V, E)$ we define for a node $v \in V$ the set of its *in-neighbors* by $N^-(v) = \{u \in V : (u, v) \in E\}$ and its *out-neighbors* as $N^+(v) = \{u \in V : (v, u) \in E\}$. We have also in this context the indegree of v given by $\delta^- = |N^-(v)|$ and its *outdegree* given by $\delta^+ = |N^+(v)|$

During the most part of of the this manuscript, and unless explicitly stated otherwise, every graph G will be assumed to be connected and undirected. We define a *class* or a *family* of graphs as a set $\mathcal{G} = \{G_n\}_{n \geq 1}$ such that $G_n = (V_n, E_n)$ is a graph of order n.

We say that G is a *tree-graph* or simply a *tree* if it does not have cycles as subgraphs. Usually, we distinguish certain node in $r \in V(G)$ that we call the root of G. Whenever G is a tree and there is a fixed vertex $r \in V(G)$ we call G a *rooted tree*. In addition, we will say that $v \in V(G)$ is a *leaf* if $\delta_v = 1$. Straightforwardly the choice of r induces a partial order in the vertices of G given by the distance (length of the unique path) between a node $v \in V(G)$ and the root r. We define the *height* of G (and we write it as $h(G)$) as the longest path between a leaf and r. We say that a node v is in the $(h(G) - k)$-th level of a tree G if the distance between v and r is k and we write $v \in \mathcal{L}_{h(G)-k}$. We call the *children* of a node $v \in \mathcal{L}_k$ all $w \in N(v)$ such that w is in level $k - 1$.

Tree decomposition, treewidth and brambles

We now introduce the graph parameter known as treewidth. This parameter indicates how "close" is a graph to a tree graph. In order to introduce this concept, we need first the following definition:

Definition 1.1 *Given a graph $G = (V, E)$, a* tree decomposition *is pair $\mathcal{D} = (T, \Lambda)$ such that T is a tree and Λ is a family of subsets of nodes $\Lambda = \{X_t \subseteq V |\ t \in V(T)\}$ called* bags *such that:*

- *Every node in G is in some X_t, i.e: $\bigcup_{t \in V(T)} X_t = V$.*

- *For every* $\mathrm{e} = uv \in E$ *there exists* $t \in V(T)$ *such that* $u, v \in X_t$.
- *For every* $u, v \in V(T)$ *if* $w \in V(T)$ *is in the* v*-*y *path in* T*, then* $X_u \cap X_v \subseteq X_w$.

We define the *width* of a tree decomposition $\mathcal{D}$ as the amount $\mathrm{width}(\mathcal{D}) = \max_{t \in V(T)} |X_t| - 1$. Given a graph $G = (V, E)$, we define its *treewidth* as the parameter $\mathrm{tw}(G) = \min_{\mathcal{D}} \mathrm{width}(\mathcal{D})$. In other words, the treewidth is the minimum width of a tree decomposition of G. Note that, if G is a connected graph such that $|E(G)| \geq 2$ then, G is a tree if and only if $\mathrm{tw}(G) = 1$.

We now introduce a dual concept related to treewidth, which is called *bramble*. Let $G = (V, E)$ be a graph and B_1, B_2 be two subgraphs of G. We say that B_1 and B_2 *touch* if they share at least a common vertex or if there is some edge $\mathrm{e} \in E$ joining them, i.e. if there exists $\mathrm{e} \in E$ such that $\mathrm{e} = (u, v)$ with $u \in B_1$ and $v \in B_2$.

Definition 1.2 *Let* $G = (V, E)$ *a graph. A* bramble *in* G *is a set* $\mathcal{B}$ *of connected subgraphs of* G *such that any* $B_1, B_2 \in \mathcal{B}$ *touch.*

Let $G = (V, E)$ a graph and $\mathcal{B}$ a bramble in G. We say that a set of nodes $S \subseteq V(G)$ is a *hitting set* for $\mathcal{B}$ if it intersects every subgraph in $\mathcal{B}$. The size of a minimum hitting set for $\mathcal{B}$ is known as its *order*.

Definition 1.3 *Let* $G = (V, E)$ *a graph and let* $\mathcal{B}$ *be a bramble on* G*. We call* $\mathcal{B}$ *a* perfect bramble *if:*

1. *Any two* $B_1, B_2 \in \mathcal{B}$ *have non-empty intersection.*
2. *For every* $v \in V$ *there are at most two subgraphs in* $\mathcal{B}$ *that contains* v.
3. *Every vertex has degree at most* 4 *in* $\bigcup_{B \in \mathcal{B}} B$.

There is a very interesting link between treewidth and perfect brambles which will be used in this work, specifically in the chapter about freezing dynamics. This latter link consists in the fact that, informally speaking, graphs having larger treewidth admit perfect brambles with sufficiently great enough order and moreover, there exist a polynomial time algorithm that finds them (see Theorem 5.3 of [55])

1.1.2 Automata networks as discrete dynamical systems

We start by stating the following basic definitions, notations and properties regarding automata networks (for more details see [62]). We use Q and V to denote finite sets representing the alphabet and the set of nodes respectively. We define $\Sigma(Q)$ as the set of all possible permutations over alphabet Q.

We call an *abstract automata network* or simply an *automata network* any function $F : Q^V \to Q^V$. Note that F induces a dynamics on Q^V and thus we can see (Q^V, F) as dynamical system. In this regard, we recall some classical definitions:

Given an initial configuration $x \in Q^V$, we define the *orbit* of x as the sequence $\mathcal{O}(x) = (F^t(x))_{t\geq 0}$. We define the set of *limit configurations* or *recurrent configurations* of F as $L(F) = \bigcap_{t\geq 0} F^t(Q^V)$. Observe that since Q is finite and F is deterministic, each configuration is eventually periodic, i.e. for each $x \in Q^V$ there exist some $\tau, p \in \mathbb{N}$ such that $F^{\tau+p}(x) = F^\tau(x)$ for all $x \in Q^V$. Note that if x is a limit configuration then, its orbit is periodic. In addition, any configuration $x \in Q^V$ eventually reaches a limit configuration in finite time. We denote the set of orbits corresponding to periodic configurations as $\mathrm{Att}(F) = \{\mathcal{O}(x) : x \in L(F)\}$ and we call it the set of *attractors* of F. We define the *global period* or simply the *period* of $\overline{x} \in \mathrm{Att}(F)$ by $p(\overline{x}) = \min\{p \in \mathbb{N} : \overline{x}(p) = \overline{x}(0)\}$. If $p(\overline{x}) = 1$ we say that $\overline{x}$ is a *fixed point* and otherwise, we say that $\overline{x}$ is a *limit cycle*.

Given any configuration $x \in Q^V$, we define its *transient length* as $\tau_F(x) = \max\{t \in \mathbb{N} : F^t(x) \notin L(F)\}$. In addition, given $x \in Q^n$ we define the restriction of x to some subset $U \subseteq V$ as the partial configuration $x|_U \in Q^{|U|}$ such that $(x|_U)_u = x_u$ for all $u \in U$.

Given a node v, its behavior $x \mapsto F(x)_v$ might depend or not on another node u. This dependencies can be captured by a graph structure which plays an important role in the theory of automata networks (see [26] for a review of known results on this aspect). This latter structure motivates the following definitions:

Definition 1.4 *Let $F : Q^V \to Q^V$ be an automata network and $G = (V, E)$ a directed graph. We say G is a* communication graph *of F if for all $v \in V$ there exist $D \subseteq N_v^-$ and some function $f_v : Q^D \to Q$ such that $F(x)_v = f_v(x|_D)$. The* interaction graph *of F is its minimal communication graph.*

Note that by minimality, for any node v and any in-neighbor u of v in the interaction graph of some F, then the next state at node v effectively depends on the actual state at node u. More precisely, there is some configuration $c \in Q^V$ and some $q \in Q$ with $q \neq c_u$ such that $F(c)_v \neq F(c')_v$ where c' is the configuration c where the state of node u is changed to q. This notion of effective dependency is sometimes taken as a definition of edges of the interaction graph (see for example [69, 63]).

From now on, for an automata network F and some communication graph G of F we use the notation $\mathcal{A} = (G, F)$. In addition, by abuse of notation, we also call $\mathcal{A}$ an abstract automata network.

1.1.3 Some important automata network families

Now we introduce the notion of automata network family. This notion is quite general and it will be specified when we address different representations for families. Let Q be a fixed alphabet. A family of automata networks is a collection of functions $\{F_s\}_{s\in\mathbb{N}}$ such that for each $s \in \mathbb{N}$, $F_s : Q^{n(s)} \to Q^{n(s)}$ is an automata network on alphabet Q. Now we are going to provide concrete examples of families that we study during this thesis work.

Observe that, as we are going to be working also with a computational complexity framework, it is necessary to be more precise in how we represent automata networks. In this regard, one possible slant is to start defining an automata network from a communication

graph.

Concrete symmetric automata networks

One of the main definition used all along this thesis work is that of *concrete symmetric automata network*. Roughly, they are non-directed labeled graph G (both on nodes and edges) that represent an automata network. They are *concrete* because the labeled graph is a natural concrete representation upon which we can formalize decision problems and develop a computational complexity analysis. They are *symmetric* in two ways: first their communication graph is non-directed, meaning that an influence of node u on node v implies an influence of node v on node u; second, the behavior of a given node is blind to the ordering of its neighbors in the communication graph, and it can only differentiate its dependence on neighbors when the labels of corresponding edges differ.

Definition 1.5 *Given a non-directed graph $G = (V, E)$, a vertex label map $\lambda : V \to (Q \times 2^Q \to Q)$ and an edge label map $\rho : E \to \Sigma(Q)$, we define the tuple $\mathcal{A} = (G, \lambda, \rho)$ and we call it a concrete symmetric automata network (CSAN) associated to the graph G. A* family of concrete symmetric automata networks *(CSAN family) $\mathcal{F}$ is given by an alphabet Q and a set of local labeling constraints $\mathcal{C} \subseteq \Lambda \times R$ where $\Lambda = \{\lambda : Q \times 2^Q \to Q\}$ is the set of possible vertex labels and $R = 2^{\Sigma(Q)}$ is the set of possible neighboring edge labels. We say that a CSAN (G, λ, ρ) belongs to $\mathcal{F}$ if for any vertex v of G with incident edges E_v it holds $(\lambda(v), \rho(E_v)) \in \mathcal{C}$.*

Note that the labeling constraints defining a CSAN family are local. In particular, the communication graph structure is a priori free. This aspect will play an important role later when building arbitrarily complex objects by composition of simple building blocks inside a CSAN family.

Let us now define the abstract automata network associated to a CSAN, by describing the semantics of labels defined above. Intuitively, labels on edges are state modifiers, and labels on nodes give a map that describes how the node changes depending on the set of states appearing in the neighborhood, after application of state modifiers. We use the following notation: given $k \geq 1$, $\sigma = (\sigma_1, \ldots, \sigma_k) \in \Sigma(Q)^k$ and $x \in Q^k$ we note $x_\sigma = \{\sigma_1(x_1), \ldots, \sigma_k(x_k)\}$.

Definition 1.6 *Given a CSAN (G, λ, ρ), its associated global map $F : Q^V \to Q^V$ is defined as follows. For all node $v \in V$ and for all $x \in Q^n$:*

$$F(x)_v = \lambda_v(x_v, (x|_{N(v)})_{\rho_v}),$$

where $N(\mathrm{i}) = \{u_1, \ldots, u_{\delta_v}\}$ is the neighborhood of v and $\rho_v = (\rho(v, u_1), \ldots, \rho(v, u_{\delta_u}))$.

Some important CSAN families Now we present some examples of families of automata networks that we study in this paper. They differ in the set of allowed labels and their degree of local symmetry.

Definition 1.7 *Let $Q = \{0, 1\}$. The family of* signed conjunctive automata networks *is the set of CSAN (G, λ, ρ) where for each node v we have $\lambda_v(q, X) = \min X = \bigwedge X$ and, for each edge* e*, ρ_{e} is either the identity map or the map $x \mapsto 1 - x$.*

The family of locally positive conjunctive networks *is the set of signed conjunctive automata networks (G, λ, ρ) where we require, in addition, that for each node v there is at least one edge* e *incident to v such that ρ_{e} is the identity.*

Finally, the familly of symmetric conjunctive networks *is the set of signed conjunctive automata networks where, for all edge* e*, ρ_{e} is the identity.*

Definition 1.8 *Let Q be a totally ordered set. The family of* min-max automata networks *over Q is the set of CSAN (G, λ, ρ) such that for each edge* e*, ρ_{e} is the identity map and, for each node v, $\lambda_v(q, X) = \max X$ or $\lambda_v(q, X) = \min X$.*

Remark 1.9 If $Q = \{0, 1\}$ then, the family of *min-max automata networks* over Q is the class of AND-OR networks, i.e., $F(x)_{\mathrm{i}} = \bigwedge_{j \in N(\mathrm{i})} x_j$ or $F(x)_{\mathrm{i}} = \bigvee_{j \in N(\mathrm{i})} x_j$

1.1.4 Bounded degree automata networks

Let us fix a natural number $\Delta > 0$. It is interesrting to consider a family of automata networks defined over a collection of communication graphs which have maximum degree bounded by Δ. Formally, a bounded degree family is a family of automata networks $\mathcal{F}_\Delta$ such that $\mathcal{A} = (G, F) \in \mathcal{F}$ is such that its communication graph G satisfies $\Delta(G) < \Delta$. We will see in the next sections that it is possible to represent $\mathcal{A}$ as a pair $(G, (\tau_v)_{v \in V(g)})$ where G is a communication graph of F of maximum degree at most Δ and $(\tau_v)_{v \in V(G)}$ is the list for all nodes of G of its local transition map F_v of the form $Q^{\mathrm{d}} \to Q$ for $\mathrm{d} \leq \Delta$. As an example, it suffices for example to consider any CSAN defined over a bounded degree graph.

1.1.5 Algebraic automata networks

In this case, the set Q is endowed with a field structure and thus the set of all possible configurations Q^n is a vector space. In this context, we define the family of algebraic automata networks over Q as the set of linear functions defined over Q^n. As a consequence of automata networks being actual linear maps, we can naturally represent them as matrices. A well known example is taking $Q = \mathbb{F}_2$ and for $n \in \mathbb{N}$ fixed, the function $F : Q^n \mapsto Q^n$ given by $F(x)_{\mathrm{i}} = \bigoplus_{\mathrm{i}=1}^{n} x_{\mathrm{i}}$ where $\oplus$ is the sum modulo 2, which is also known as the XOR function.

1.1.6 Freezing automata networks

Now, we present the family of automata networks called *freezing* automata networks. Let us consider an alphabet Q together with a partial order $\leq$. We say that an automata network $F : Q^n \mapsto Q^n$ has the *freezing property* if F is non-decreasing in $\leq$. Formally, if for each $x \in Q^n$ and for each $v \in \{1, \ldots, n\}$ we have $x_v \leq F(x)_v$. We will focus mainly, during this thesis work, in the non-deterministic version of this type of automata network which we will

introduce in the next subsection. Observe that, any freezing automata network $F : Q^n \mapsto Q^n$ has an interesting property regarding its orbits. Given an arbitrary orbit $\mathcal{O}(x)$ starting from some initial configuration $x \in Q^n$, each coordinate can change at most $|Q| - 1$ times, since once a coordinate changes its state it can never go back to a previous state (related to the order $\leq$). This property will help us in order to design efficient algorithms in order to predict the dynamical behavior of this type of automata network in the next chapters.

1.1.7 Non-deterministic automata networks

Let Q be a finite set that we will call an *alphabet*. We define a *non-deterministic automata network* over the alphabet Q as a tuple $(G = (V, E), \{F_v : Q^{N(v)} \to \mathcal{P}(Q) | v \in V\}))$ where $\mathcal{P}(Q)$ is the power set of Q. To every non-deterministic automata network we can associate a non-deterministic dynamics given by the global function $F : Q^n \to \mathcal{P}(Q^n)$ defined by $F(x) = \{x \in Q^n : x_v \in F_v(x), v \in V\}$.

Definition 1.10 *Given a a non-deterministic automata network $(G, \mathcal{F})$ we define* an orbit of a configuration $x \in Q^n$ at time t *as a sequence $(x_s)_{0 \leq s \leq t}$ such that $x_0 = x$ and $x^s \in F(x^{s-1})$. In addition, we call the set of all possible orbits at time t for a configuration x as $\mathcal{O}(x, t)$. Finally, we also define the set of all possible orbits at time t as $\mathcal{O}(\mathcal{A}, t) = \bigcup_{x \in Q^n} \mathcal{O}(x, t)$*

Observe that many properties can be extended from the deterministic case in a natural way, for instance, the freezing property: we say that a non-deterministic automata network (G, F) defined in the alphabet Q satisfies the freezing property or simply that it is freezing if there exists a partial order $\leq$ in Q such that for every $t \in \mathbb{N}$ and for every orbit $y = (x^s)_{0 \leq s \leq t} \in \mathcal{O}(\mathcal{A}, t)$ we have that $x_{\mathrm{i}}^s \leq x_{\mathrm{i}}^{s+1}$ for every $0 \leq s \leq t$ and for every $0 \leq \mathrm{i} \leq n$.

Let $y = (x^s)_{0 \leq s \leq t}$ be an orbit for a non-deterministic automata network $(G, \mathcal{F})$ and $S \subseteq V$ we define the restriction of y to S as the sequence $z \in (Q^t)^{|S|}$ such that $x_v^s = z_v^s$ for every $v \in V$ and we note it $y|_S$. In the case in which $S = \{v\}$ we simply write y_v in order to denote the restriction of y to the singleton $\{v\}$.

Definition 1.11 *Given a non-deterministic automata network $(G, \mathcal{F})$ and a set $S \subseteq V$, we define the set of S-restricted orbits as the set $\mathcal{T}(S, t) = \{z = (x_s)_{s \leq t} \in Q^{|S|} \mid \exists y \in \mathcal{O}(t) : y|_S = z\}$. When $S = \{v\}$ we simply write $\mathcal{T}(v, t)$ for $\mathcal{T}(\{v\}, t)$.*

Remark 1.12 Observe that, given a deterministic freezing automata network of global rule $F : Q^V \to Q^V$, we define the associated non-deterministic global rule F^* where each node can at each step apply F or stay unchanged, formally: $F_v^*(c) = \{F_v(c), c_v\}$. It represents the system F under an update scheme that is called *totally asynchronous update scheme*.

1.2 Elements of computational complexity

In this section, we will briefly define some basic concepts we will be using during the rest of this work. However, we refer interested reader to [3, 23] for detailed information.

We start with the concept of decision problem. Formally, a decision problem $f_L : \{0,1\}^* \to \{0,1\}$ is a function such that $f_L(x)$ indicates whether string x belongs to a set $L \subseteq \{0,1\}^*$. Set L is called a *language*. From now on, we associate a decision problem with its language. We say that a Turing machine *decides* a decision problem if, when the machine is initialized with x written in the tape, the machine always halts, and when it does it has left written in the tape $f(x)$. We say that the machine *accepts* or *rejects* depending on the bit it writes when it halts. The time-complexity of the problem, is the amount of time steps that a Turing machine requires to solve the problem in the worst case, measured in terms of the size of the input n. Analogously, the space-complexity of the problem is considered to be the number of locations that are at some point non-blank, during the execution of the Turing machine, that are required to decide a language L on some given input x, measured in terms of its size ($|x| = n$).

Classical computational complexity theory defines two main classes of decision problems, namely **P** is the class of problems solvable in polynomial-time (i.e. with time-complexity $n^{\mathcal{O}(1)}$) on a deterministic Turing machine; and **NP** are the problems that can be solved in polynomial-time on a non-deterministic Turing machine [3]. Equivalently, we can define **NP** as the problems that can be *verified in polynomial time*. Formally, a *polynomial-time verifier* V for a language L is a deterministic Turing machine such that, there is a polynomial p satisfying that for every $x \in \{0,1\}^*$ we have that $x \in L$ if and only if there exists a $u \in \{0,1\}^{p(|x|)}$ such that V accepts on input (x,u) (i.e. the concatenation of x and u). When $V(x,u)$ accepts, we say that u is a *certificate* for x.

In addition, in terms of space-complexity, there are three main classes of decision problems, namely **L** is the class of problems solvable in logarithmic space, i.e. solved by a deterministic Turing machine with space-complexity $\mathcal{O}(\log n)$ (sometimes also denoted as **DLOGSPACE**) where n is the size of the input, **NL** which is the non-deterministic version of the latter class i.e. problems that are solvable by a non-deterministic Turing machine with space-complexity $\mathcal{O}(\log n)$, and the class of problems that are solvable in polynomial space by a deterministic Turing machine, which is denoted **PSPACE**

Clearly **P** $\subseteq$ **NP** and also it is wide-believed (and probably one of the most famous conjectures in this context) that **P** $\neq$ **NP**, i.e that the inclusion is proper. In this regard, the problems in **NP** that are most likely to not be contained in **P** are called **NP**-Complete problems. Roughly, a problem in **NP** is **NP**-complete if an efficient solution for it can be transformed into an efficient solution for every other problem in **NP**. Formally, a language $L' \in$ **NP** is **NP**-hard if for every $L \in$ **NP** we have that L is polynomial time reducible to L', meaning that there exists a function f computable in polynomial time, such that x belongs to L if and only if $f(x)$ belongs to L'. In addition, if $L' \in$ **NP** then we say L' is **NP**-complete.

The canonical **NP**-complete problem is the Boolean satisfiability problem SAT. This problem receives as an input a Boolean formula $\mathcal{F}$ on n variables, and the task consists in deciding whether this formula can be satisfied, i.e. it is possible to assign truth-values to its

inputs such that the formula evaluates *true*.

Observe that, as a Turing machine can visit only one cell at time, we have that $\mathbf{P} \subseteq \mathbf{PSPACE}$ and it is also wide-believed that $\mathbf{P} \neq \mathbf{PSPACE}$. In fact, as $\mathbf{NP} \subseteq \mathbf{PSPACE}$, $\mathbf{P} = \mathbf{PSPACE}$ then, $\mathbf{P} = \mathbf{NP}$. Analogously, there is a notion of **PSPACE**-completeness proposed in similar terms. Formally, a language is L' is **PSPACE**-hard if all language $L \in \mathbf{PSPACE}$ is polynomial time reducible to L' and we say is **PSPACE**-complete if $L' \in \mathbf{PSPACE}$.

A canonical **PSPACE**-complete problem is a generalization of SAT known as *quantified Boolean formula problem* or $QBFP$ consisting in deciding, given a quantified formula of the form $Q_1 Q_2 Q_3, \cdots, Q_n \varphi$ where Q_{i} are quantifiers i.e. $Q_{\text{i}} \in \{\exists, \forall\}$. Additionally, it is well known that the problem of deciding, given a linear bounded deterministic Turing machine M, an input $x \in \{0,1\}^*$ such that $|x| = n$ and a padding of length Kn, whether M accepts x in space Kn.

1.2.1 Counting complexity

A *functional problem* for a function $f : \{0,1\}^* \to \mathbb{N}$ is the computational task consisting in computing $f(x)$ for a given input string $x \in \{0,1\}^*$. Functional problems are a generalization of decision problems, which are functions $f_L : \{0,1\}^* \to \{0,1\}$ such that the bit $f_L(x)$ indicates whether x belongs to a set $L \subseteq \{0,1\}^*$. We abuse the notation and identify a function f with the computational task consisting in computing $f(x)$.

We say that a Turing machine *solves* a functional problem if, when the machine is initialized with x written in the tape, the machine always halts, and when it does it has left written in the tape $f(x)$. For decision problems, one can define time and space complexity for computing $f(x)$.

Some complexity classes for decision problems have natural generalizations to classes of functional problems. For instance, **FP** is the set of functions computable in polynomial time by a deterministic Turing machine. In other words, **FP** is the generalization of **P** to function problems. However, it is not easy to generalize into function classes decision problems defined by non-deterministic machines, such as **NP**. A straightforward approach is to take, for a given instance x of **NP** language L, the problem consisting in computing a certificate y for x. For example, generalizing SAT to a function problem consists in computing a truth-assignment satisfying a Boolean formula given in the input. The complication is that such a generalization does not define a function but a *binary relation*, as a single instance could have several certificates. In that context, an interesting alternative is to consider functions that *count* certificates. For instance, consider #SAT as the function problem consisting in computing the number of truth-assignments satisfying a given Boolean formula.

The class #**P** is the set of functions f such that there exists a polynomial-time verifier V such that, for every $x \in \{0,1\}^*$,

$$f(x) = \#\{y \in \{0,1\}^{p(|x|)} : V(x,y) \text{ accepts}\},$$

where p is the polynomial defined for V. Informally speaking, class #**P** contains all functions that *count certificates* of **NP** problems. Directly from the definition we have that #SAT $\in$

#**P**.

Note that **FP** is contained in #**P**. Indeed, for a given function $f \in \mathbf{FP}$, we define the polynomial verifier V_f that accepts (x, y) if and only if $y \leq f(x)$, when both strings are interpreted as positive integers. Then, $f(x)$ is exactly the number of certificates for x on verifier V_f.

It is conjectured that #**P** $\neq$ **FP**, as clearly #**P** = **FP** implies **NP** = **P**. Moreover, the equality of the two classes would have much stronger (and unlikely) consequences. For instance, Toda's theorem imply that if **FP** = #**P** then the whole polynomial hierarchy collapses in **P** [3, Theorem 9.11].

The functions in #**P** that are the most likely not belong to **FP** are the #**P**-Complete functions. A function $f \in \#\mathbf{P}$ is #**P**-complete if an efficient algorithm computing f can be used to efficiently solve all problems in #**P**. For defining this notion of hardness, we need the notion of *Turing reduction*.

We say a Turing machine has *oracle access* to a function $f : \{0,1\}^* \to \{0,1\}^*$ if at any given time step, the machine can execute a subroutine that in one time-step chooses a section of the tape containing a string $x \in \{0,1\}^*$, and writes in the section of the tape the string $f(x)$. For fixed function $f : \{0,1\}^* \to \{0,1\}^*$, we call $\mathbf{FP}^f$ the class of functions $g : \{0,1\}^* \to \mathbb{N}$ that are computable in polynomial-time by a Turing machine with oracle access to f. Intuitively, suppose that we have an algorithm computing function f. Then $\mathbf{FP}^f$ is the set of functions that can be computed running the algorithm for f a polynomial number of times. In particular, when f is computable in polynomial time $\mathbf{FP}^f = \mathbf{FP}$.

A function g is *Turing reducible in polynomial time* to a function f if $g \in \mathbf{FP}^f$. Then, we say that a function $f \in \#\mathbf{P}$ is #**P**-complete if for all function $g \in \#\mathbf{P}$ we have that g is Turing reducible in polynomial time to f, i.e. $g \in \mathbf{FP}^f$. An important example of a #**P**-Complete problem is #Sat [3, Theorem 9.7]. In fact, it is natural to expect that an **NP**-Complete decision problem defines a #**P**-Complete counting version. Up to our knowledge, there are no examples of a **NP**-Complete problems whose counting version is not #**P**-Complete [58].

Interestingly, there are #**P**-Complete problems for which the corresponding decision version can be solved in polynomial time. An important example is the *Monotone-2CNF-Satisfability* (Mon-2-Sat) and its counting version #Mon-2-Sat. A n-variable Boolean formula $\mathcal{F}$ is a *monotone-2-CNF formula* if $\mathcal{F}$ can be written as

$$\mathcal{F}(z_1, \ldots, z_n) = \bigwedge_{j \in [m]} (c_1^j \vee c_2^j) \text{ with } c_j^1, c_j^2 \in \{z_1, \ldots, z_n\}.$$

In such a case we call (c_1^j, c_2^j) a *clause*. Now consider Mon-2-Sat as the problem Sat restricted to monotone-2-CNF formulas. This problem is trivial, as such a formula can be satisfied assigning all variables *true*. Now consider #Mon-2-Sat as the restriction of #Sat to monotone-2-CNF formulas. Formally, the latter problem is defined by

Problem (#Mon-2-Sat)

Input: a monotone-2-CNF-Boolean formula $\mathcal{F}$ with n variables and m clauses.

Output: $|\{x \in \{\textbf{True}, \textbf{False}\}^n : F(x) = \textbf{True}\}|$

Like #SAT, problem #MON-2-SAT belongs to #**P**. Moreover, in [76] it is shown that it remains #**P**-Complete.

1.2.2 Parametric complexity

We now present basic concepts in parameterized complexity that we will be using during this thesis work (see [23] for more details and context). A parameterized language is defined by $L \subset \{0,1\}^* \times \mathbb{N}$. Whenever we take an instance (x,k) of a parameterized problem we will call k a *parameter*. The objective behind parameterized complexity is to identify which are the key parameters in an intractable problem that make it hard.

For instance, consider the problems Vertex-Cover and Dominating-Set (i.e. finding a set of nodes that cover all edges or all the neighborhoods, respectively) that are classical examples of **NP**-Hard problems. Consider now problems k-Vertex-Cover and k-Dominating-Set as the *parameterized version* of Vertex-Cover and Dominating-Set, respectively, when the solution must be of size at most k. Then those versions can be easily solved in time $n^{\mathcal{O}(k)}$ by simply checking all possible subsets of k nodes. Hence, when k is constant, those problems can be solved in polynomial time. Contrarily, problem Coloring, another example of an **NP**-Hard problem, remains hard even when it is parameterized by the number of colors (for instance 3-Coloring is **NP**-Complete). We say that a parameterized language L is *slice-wise polynomial* if $(x,k) \in L$ is decidable in polynomial time for every fixed $k \in \mathbb{N}$. More precisely, when $(x,k) \in L$ can be decided in time $|x|^{f(k)}$ for some arbitrary function f depending only on k. The class of slice-wise polynomial parameterized languages is called XP. In our examples, k-Vertex-Cover and k-Dominating-Set are in **XP** when they are parameterized by the size of the solution.

An important subclass of **XP** is the set of parameterized languages L that are *fixed-parameter tractable*, denoted FPT. A parameterized language L is in **FPT** if there exist an algorithm deciding if $(x,k) \in L$ in time $f(k)|x|^{\mathcal{O}(1)}$ where f an arbitrary function depending only in k. It is known that **XP** is not equal to **FPT**, however showing that some problem in **XP** is not in **FPT** seems currently out of reach for many natural examples. For instance, it is known that k-Vertex-Cover is **FPT**, and is widely-believed that k-Dominating-Set is not.

As in other domains of complexity a hardness theory has been developed relying on the following notion of reduction. Given two parameterized languages L_1, L_2 we say that L_1 is FPT reducible to L_2 (and we write this as $L_1 \leq_{\text{FPT}} L_2$) if there exist some functions $r, s : \mathbb{N} \to \mathbb{N}$ and $M : \{0,1\}^* \times \mathbb{N} \to \{0,1\}^*$ such that for each instance (x,k) of L_1, M is computable in time $s(k)|x|^c$ for some constant c and $(x,k) \in L_1$ if and only if $(M(x), r(k)) \in L_2$. A hierarchy of parameterized languages has been defined, called **W**-hierarchy, that contain a countable sequence of classes of parameterized languages, namely $\mathbf{W}[1]$, $\mathbf{W}[2], \mathbf{W}[3] \ldots$ such that $\mathbf{FPT} \subseteq \mathbf{W}[1] \subseteq, \mathbf{W}[2] \subseteq \ldots \mathbf{XP}$ and it is conjectured that are inclusions are proper. We don't give the formal definition these classes and we refer to [23] for more details. For our purposes, it is enough to know that k-Dominating-Set (i.e. finding a set of k nodes that intersects the neighborhood of any node) is $\mathbf{W}[2]$-hard and that a parameterized language is

W[2]-hard if there is an **FPT**-reduction from k-Dominating-Set to it.

1.2.3 Parellel computing and NC

We now present, briefly, some elements on parallel computation and particularly on the class **NC**. We will not give many details on definitions but a general framework and idea of the definitions that are required to understand presented results. We refer interested reader to [45]. We start by defining to different parallel computation models before introducing the formal definition of the **NC** class. There is an equivalent way to define the latter class in terms of both models as we will see in this section.

PRAM model

In the context of sequential algorithm design, one of the most well-known models of computation is the random access machine or RAM. This model consists of a computation unit provided with a fixed user defined program. Particularly, it has a read-only and a write only input tapes together with an unbounded number of local memory cells $R_0, \ldots, R_n, \ldots$. Each of these memory cells is able to hold an integer of unbounded size. Computation unit is defined by simple tasks such as: moving data between memory cells, comparing information and establishing conditional branches, executing simple arithmetic operations (subtraction, addition, multiplication, division, etc), etc. A RAM program is defined by a sequence of the latter instructions which starts with a first given instruction and continues executing each consecutive instruction in the sequence until a halt instruction is reached. Usually, complexity in this model is measured in terms of the number of instructions executed and space in the form of memory cells that are accessed during execution. An important remark is that, as memory unit can handle integers of unbounded size, the model prohibits rapid generation of very large numbers (by, for example, a careful choice of the instruction set). For example, the model will prohibit numbers of superpolynomial size to be tested or generated in order to avoid an effect of distortion in the context of latter notion of time complexity. More precisely, for $t \geq \log n$ no number may exceed $\mathcal{O}(t)$ bits in t steps.

A generalization of this model to the context of parallel computing is called Parallel Random Access Machine (PRAM). This model consists of a collection of numbered RAM processors $P_0, \ldots, P_n, \ldots$, and an unbounded collection of shared memory cells $C_1, \ldots, C_n, \ldots$. Each of these processors is equipped with its own collection of local memory cells and its own index. In addition, each processor has instructions involving read/write access of shared memory cells. Instead of being in tapes, inputs and ouputs are placed in shared memory cells allowing processors to have concurrent access to this information. Additionally, instructions are executed in unit time and are synchronized over all active processors. In the literature regarding this model there are two technical issues that need to be specified in the definition of the model: the first one is related to the coordination of different processors during computation and the second one is how execution is handled when multiple processors require access to the same shared memory cells. The most common way to solve the first issue is by adding a processor P_0 which stores a number representing the larger index of a processor which is allowed to be activated. All processors having an index that is lower than this latter amount will be activated during execution. Initially only processor P_0 is active, and executes a special instruction which computes the required number of processors and stores this value

in a special register. All the other processors wait for P_0 to finish this instruction and then start execution accordingly. When the time in which processor P_0 halts is reached all active processors halt.

Second issue is typically handled in the following way: reading and writing are handled in separated (independent) cycles. PRAM instruction in shared memory is executed in three phases: first the reading phase in which all processors read information for shared memory, then computation associated with read instructions are handled and finally there is a writing phase. Then, access conflicts can be addressed in different ways:

1. Concurrent Read Concurrent Write (CRCW) model: simultaneous readings and writings are allowed and thus, some method of arbitrating simultaneous writings in shared memory is required. Different variants are proposed such as: PRIORITY in which when there is a simultaneous writing conflict only the processor with lowest index writes, COMMON in which the write succeeds only if all involved processors are writing the same value and finally there is ARBITRARY in which some processor is chosen in a non-deterministic way.
2. Concurrent Read Exclusive Write (CREW) model: simultaneous readings are allowed on a same shared memory cell but only one processor can write in that cell at the same time.
3. Exclusive Read Exclusive Write (EREW) model: no two processors can access one memory cell simultaneously.

The standard measure of complexity is given in terms of the following definition of computation:

Definition 1.13 *Let M be a PRAM. An input $x \in \{0,1\}^*$ such that $|x| = n$ is considered as a natural number n together with a sequence of bits. Each bit is stored in memory cells $C_1, \ldots, C_n$ and n is stored as an integer in C_0. An output is a string $y \in \{0,1\}^*$ stored in the same way. We say that M computes in parallel time $t(n)$ and processors $p(n)$ if for any input $x \in \{0,1\}^n$, M halts within at most $t(n)$ time steps, activates at most $p(n)$ processors during execution and presents an output $y \in \{0,1\}^*$. We say M computes sequentially if it takes $t(n)$ time steps by using 1 processor.*

Observe that, since the model prohibits rapid generation of large integers, functions $p(n)$ and $t(n)$ are sufficiently coherent to describe time and space complexity. It is also well-known that previous variants (CRCW,CREW and EREW models) are polynomially equivalent (one can simulate the execution of one particular model into other one by using a polynomial amount of resources).

We now present the definition of the class **NC** and then, we will return to this formalism in order to establish a link between this latter complexity class and the PRAM model.

NC class and uniform circuit families

We first recall the definition of a uniform circuit family. A Boolean circuit α is a labeled directed acyclic graph in which each vertex v is called a *gate* and it is labeled by some function

$g(v) : \{0,1\}^{k(v)} \to \{0,1\}^{k'(v)}$ where $k(v), k'(v) \geq 0$ are such that $k(v) = \delta_v^+$ and $k'(v) = \delta_v^-$. Usually, in the literature ,the outdegree or fanout $k'(v)$ is not restricted but the indegree or fanin is considered to be at most 2. Whenever a vertex has outdegree 0 i.e. $\delta_v^- = 0$ we say v is an *output* and if $\delta_v^+ = 0$ we say it is an *input*. A Boolean circuit having n inputs and m outputs computes a function $f : \{0,1\}^n \mapsto \{0,1\}^m$ in the following way: Let $x \in \{0,1\}^n$ be an argument for f. Each node is assigned with some value $\nu(v) \in \{0,1\}$ depending on its type. If v is an input then, it is assign by $\nu(v) = x_v$. For each gate which is not an output nor an input, we assign a value $\nu(v)$ according to the value assigned to its incoming vertices and function $g(v)$. At the end, each output v satisfies $\nu(v) = f(x)$. Observe that the order in which values are computed for incoming neighbors could play a role depending on each gate.

We define a layer or level of node v in a circuit as the length of the longest path from an input to v. The size of the circuit is denoted size(α) and is defined as the number of vertices in the circuit. In addition, the depth of a circuit is denoted as deph(α) and it is defined as the longest path from an input to an output.

A Boolean circuit family $\{\alpha_n\}$ is a collection of circuits, each α_n computing a function $f^n : \{0,1\}^n \mapsto \{0,1\}^{m(n)}$. The function computed by the family $\{\alpha_n\}$ is a function of the form $f_\alpha : \{0,1\}^* \mapsto \{0,1\}^*$, defined by $f^\alpha(x) = f^{|x|}(x)$. Let $f : \{0,1\}^* \mapsto \{0,1\}^*$ be a function. Let $\{\alpha_n\}$ a Boolean circuit family that computes some function $f_\alpha : \{0,1\}^* \mapsto \{0,1\}$. We define the language *accepted* by $\{\alpha_n\}$ as the set $L_\alpha = \{x \in \{0,1\}^* : f_\alpha(x) = 1\}$ and we say that L_α is recognized by family α_n.

We say that a family $\{\alpha_n\}$ of circuits is *logarithmic space uniform* if the transformation $1^n \mapsto \overline{\alpha_n}$ where $\overline{\alpha_n}$ is the standard encoding of α_n (see [45]) can be computed in $\mathcal{O}(\log(size(\alpha_n)))$ space on a deterministic Turing machine.

We are now ready to define the class **NC** :

Definition 1.14 *Let $k \geq 1$ be an integer. We define the class $\boldsymbol{NC}^k$ as the set of all languages L such that L is recognized by a uniform Boolean $\{\alpha_n\}$ satisfying size$(\alpha_n) = n^{\mathcal{O}(1)}$ and depth$(\alpha_n) = \mathcal{O}((\log(n)))^k)$ and we define $\boldsymbol{NC} = \bigcup_{k \geq 1} \boldsymbol{NC}^k$.*

Now we present a result that relates the PRAM model to the **NC** class. In fact, the result establishes an equivalence between circuits and PRAMs and also establishes a hierarchy called the **NC**-hierarchy.

We call a PRAM algorithm *logarithmic space uniform* if there exists a logarithmic space deterministic Turing machine which takes an input $x \in \{0,1\}^n$ and generates the program executed by each processor for correspondent input. Let $k \geq 1$. We call $\mathbf{CRCW}^k$ (respectively $\mathbf{CREW}^k$, $\mathbf{EREW}^k$) the class of problems that are solvable by a *logarithmic space uniform* CRCW (respectively $\mathbf{CREW}^k$, $\mathbf{EREW}^k$) PRAM algorithm using $\mathcal{O}((\log(n)))^k)$ time and $n^{\mathcal{O}(1)}$ processors.

Proposition 1.15 *Let $k \geq 1$, the following inclusions hold:*

$$\boldsymbol{EREW}^k \subseteq \boldsymbol{NC}^k \subseteq \boldsymbol{CREW}^k \subseteq \boldsymbol{CRCW}^k \subseteq \boldsymbol{NC}^{k+1}.$$

Finally, we would like to remark that it is clear from the latter definitions that $\mathbf{NC} \subseteq \mathbf{P}$ and it is well-believed that the inclusion is strict. This means that there are some problems that are solvable in polynomial time but are believed not to be solvable by an efficient parallel algorithm. An example of a **P**-complete problem under **NC**-reductions is the well-known *circuit value problem*. Interested reader is referenced to [45] for more details.

1.3 Toolbox

In this section, we state some useful results that we use during the thesis work. We only provide the statement of the results and we refer readers to corresponding references.

1.3.1 Automata networks dynamics

In this subsection, we present some results regarding a specific family of automata networks called threshold networks, which includes some of the CSAN families that we analyze during this work. In addition, we state some technical results on number theory that we use in order to deduce lower bounds on periods.

Threshold networks

A threshold network is an automata network on alphabet $Q = \{0, 1\}$ such that the next state of each vertex is given by the amount of neighbors that are currently in state 1. If this amount is more than some given threshold, then the node will change to state 1 and will remain in previous state otherwise. Formally, a threshold network is a tuple $\mathcal{N} = (G, \lambda)$ where G is a non-directed graph and $\lambda : V \times \{0, 1\} \times \mathbb{N}^{\{0,1\}} \mapsto \{0, 1\}$ such that $\lambda(q, m) = 1$ if $m(1) \geq \theta_v$ and $\lambda(q, m) = 0$ otherwise. Observe that threshold network is a type of network that is similar to a CSAN but more general as m is a multiset. Well known examples of threshold networks are majority networks in which $\theta_v = \lceil \frac{\delta_v}{2} \rceil$ and thus v will change to 1 when the majority of its neighbors in state 1. Other well known examples are disjunctive networks ($\theta_v = 0$) and conjunctive networks ($\theta_v = \delta_v$). We introduce a classical result on threshold network dynamics by Goles et al. [42] which bounds the transient and the period of attractors for this family.

Proposition 1.16 ([42]) *Let Θ be the family of all threshold networks. Each network $\mathcal{N} = (G, \lambda) \in \Theta$ has only attractors for period 2 or fixed points. In addition, for each configuration $x \in \{0, 1\}^{V(G)}$ its transient time $\tau_{\mathcal{N}}(x)$ satisfies $\tau(x) = |V(G)|^{\mathcal{O}(1)}$.*

Superpolynomial period attractors

We present in this section a classical result in number theory that we use to show the existence of networks admitting attractors of superpolynomial period by combining different connected components exhibiting attractors of some prime period.

Lemma 1.17 *[47] Let $m \geq 2$ and $\mathcal{P}(m) = \{p \leq m \mid p \text{ prime}\}$. If we define $\pi(m) = |\mathcal{P}(m)|$ and $\theta(m) = \sum_{p \in \mathcal{P}(m)} \log(p)$ then we have $\pi(m) \sim \frac{m}{\log(m)}$ and $\theta(m) \sim m$.*

1.3.2 Treewidth and tree decompositions

It is well known that, given an arbitrary graph G, and $k \in \mathbb{N}$, the problem of deciding if $\text{tw}(G) \leq k$ is **NP**-complete [2]. Nevertheless, if k is fixed, that is to say, it is not part of the input of the problem then, there exist efficient algorithms that allow us to compute a tree-decomposition of G. More precisely, it is shown that for every constant $k \in \mathbb{N}$ and a graph G such that $\text{tw}(G) \leq k$, there exists a log-space algorithm that computes a tree-decomposition of G [24]. In addition, in Lemma 2.2 of [10] it is shown that given any tree decomposition of a graph G, there exists a fast parallel algorithm that computes a slightly bigger width binary tree decomposition of G. More precisely, given a tree decomposition of width k, the latter algorithm computes a binary tree decomposition of width at most $3k + 2$. We outline these results in the following proposition:

Proposition 1.18 *Let $n \geq 2, k \geq 1$ and let $G = (V, E)$ with $|V| = n$ be a graph such that $tw(G) \leq k$. There exists a CREW PRAM algorithm using $\mathcal{O}(\log^2 n)$ time, $n^{\mathcal{O}(1)}$ processors and $\mathcal{O}(n)$ space that computes a binary treewidth decomposition of width at most $3k + 2$ for G.*

1.3.3 Fast parallel subroutines

We cite the following results that we use as some kind of toolbox for the proof of our main results in freezing dynamics (see Chapter 6):

Proposition 1.19 (Prefix-sum algorithm, [50]) *The following problem can be solved by a CREW PRAM machine with $p = \mathcal{O}(n)$ processors in time $\mathcal{O}(\log n)$: Given $A = \{x_1, \ldots, x_n\}$ be a finite set, $k \leq n$ and $\oplus$ be a binary associative operation in A, compute $\oplus_{\mathrm{i}=1}^{k} x_{\mathrm{i}}$*

Proposition 1.20 ([68, Theorem 5.3]) *Let $n \in \mathbb{N}$. The following problem can be solved in space $\mathcal{O}(\log n)$: given an undirected graph $G = (V, E)$ with $|V| = n$, $s, t \in V$ find a path from s to t and if there exists such a path, return the path as an output.*

Proposition 1.21 ([28, Theorem 3]) *Let $\Delta \in \mathbb{N}$. The following problem can be solved in time $\mathcal{O}(\Delta \log(\Delta + \log^* n))$ by an EREW PRAM: given a graph $G = (V, E)$ such that $\Delta(G) \leq \Delta$ finding a $\Delta + 1$ coloring of G.*

1.3.4 Arithmetic results

In our proofs we will require to perform some arithmetic computations of large integers. We finish this section by giving a proposition that compiles results regarding the complexity of arithmetic computations.

Proposition 1.22 ([1]) *Given two integers $x, y \in [n]$:*

- *There is an algorithm computing the product xy in time $\mathcal{O}(\log n \log\log n)$*
- *There is an algorithm computing the division x/y in time $\mathcal{O}(\log n \log\log^2 n)$*
- *There is an algorithm computing the factorial $x!$ in time $\mathcal{O}(n(\log n \log\log n)^2)$.*

Therefore, when $y \leq x$ the expression $\binom{x}{y} = \frac{x!}{(x-y)!y!}$ can be computed in time $\mathcal{O}(n^2)$.

In addition, at some point we would like to compute the coefficients of a polynomial $P(x)$. This is possible when we know an evaluation of the polynomial in a large enough point:

Lemma 1.23 ([76]) *Let $P(x) = \sum_{i=0}^{n} a_i x^i$ be a polynomial with integer coefficients upper bounded by a $A > 2$. Suppose that we know a pair (x_0, y_0) such that $y_0 = P(x_0)$ and $x_0 > A^2$. Then, there exists an algorithm that outputs the coefficients $a_0, \ldots, a_n$ of P in a time that is polynomial in $n(\log(x_0) + \log(y_0) + \log n)$.*

[1] See https://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations for a nice table resuming different algorithms

Chapter 2

Measuring dynamical behavior

In this chapter, a theoretical framework for studying automata networks is presented, based on a computational complexity approach. In particular, decision problems regarding questions about the dynamics of specific automata networks are studied. As it has been many times referred in the literature, one of the most tricky parts of the analysis of a decision problem, is to observe how the input is given. This of course will have a tremendous impact on its computational complexity as in some way it fixes the objects that we have available in order to solve the latter problem. In fact, it determines the amount of resources that one has to manage in order to effectively treat available information. In this regard, a natural question is how automata networks are represented. Is there something like a standard representation for a certain family of automata networks? If some extra information regarding the networks is given (the topology of the network, the properties of the local rues, etc.), is it possible to represent in a canonical way a vast group of networks?. All these issues regarding representation of automata networks are explored.

Then, once we know what is an automata network for practical purposes, the question on how different families can be classified in terms of their dynamical properties is discussed. More precisely, the question of how the dynamics of a certain family may simulate the dynamics of another one is explored. In order to address this question, a notion of simulation between standard representations of automata network families are adapted from dynamical notions of simulation presented in previous sections. In this regard, we discuss how representation plays an essential role when universality (the capacity of simulating any arbitrary automata network) is addressed. We propose two definitions of universality that are closely related to the concept of simulation. Roughly speaking, if a family is able to simulate any automata network (provided that it can be represented by a circuit that has reasonable size related to the size of the network), it is universal. In addition, if some family is able to simulate any bounded degree network, then, it is strongly universal. We show that the latter definition is in fact stronger, by exhibiting a family of bounded degree that is universal and strongly universal at the same time.

Finally, we study in detail the prediction problem. As it was introduced before in this thesis, the complexity of this problem gives some insight into how complex is the dynamics of certain automata network by measuring how difficult is to predict its dynamical behavior

with respect to the complexity of simply computing the global dynamics through its global rule. If the automata network presents some topological or algebraic properties that can be exploited in order to propose an efficient algorithm to solve prediction problem (compared to raw computation of global rule) then, its dynamics is said to be simple. Contrarily, if we have no other choice than simple simulating the global rule then, its dynamics is said to be complex. In this section, we study different variants of the prediction problem. In particular, since an observation time is needed in order to evaluate the state of the objective node, the representation of time (unary or binary) is addressed. In addition, long-term prediction, a version in which time is not part of the input, is explored. Then, we apply previous results on simulation in order to establish some sorts of heritage properties. Generally speaking, if a certain family simulates another one then, the prediction problem on the simulator family will be as hard as on the simulated family.

2.1 Automata network representations

In this section, the notion of standard representation is introduced. A standard representation of a certain family $\mathcal{F}$ of automata networks is a family $\mathcal{F}^*$ of Boolean circuits which computes, via some coding function depending on the alphabet of the family, each automata network in the usual way. Note that each circuit can be encoded in multiple ways as well it is possible that the circuit is considered to be given in its standard encoding. In addition, a representation could be virtually any kind of object which encodes in some way a circuit family simulating the original automata network family. Then, in general, a standard representation is considered to be a particular language in which any word *encodes* in an *efficient* way the encoding of some circuit computing an automata network in the family. In order to motivate an analysis on how could be a natural representation in the context of concrete examples of automata network families, we present here different representations and we show that they are related to the circuit representation.

2.1.1 Representation of some particular families

Observe that the communication graph is an interesting piece of information in the context of describing an automata network, as it contains, the structure of the global rule, expressed as the adjacency of nodes in latter network. However, as we will see in this section, constructing the communication graph of a given automata network (provided that this automata network is represented for example as a circuit) is not always an easy task. Conversely, if some representation including the information of the automata network (plus for example, information about local rules) is provided, it could be also hard to construct a circuit representation for the global rule of the automata network. We will illustrate this fact by analyzing three examples of canonical types of families: bounded degree networks, CSAN and algebraic families. The first one is characterized by a communication graph which has bounded degree, i.e. there exists a constant $\Delta \in \mathbb{N}$ (not depending on the size of the network) such that maximum degree is at most Δ. The second one was widely presented in previous sections and it roughly corresponds to the family of all set-valued functions described by labelled communication graphs. Finally, an algebraic family is essentially a family of linear maps in the case in which Q is seen as a finite field and Q^n as a linear space. We show in the latter examples that there exists a natural representation that is based on the properties of the family. For instance,

it is possible to deduce that for the bounded degree families, the bounded degree parameter provides a way to store information in an efficient way. This is because local functions do not depend in the size of the network. In addition, we have also an efficient representation for the case of CSAN since local functions are set-valued (locally every node depends only on the set of states given by the states of their neighbors) and thus they do not depend on the size of the network. Finally, we have the case of linear maps that can be represented as matrices.

The CSAN case.

Note that, when we think at the latter collection of automata networks as a CSAN family, we can directly work with a structure that can be represented efficiently. This is because, a CSAN family is a collection of labeled graphs and thus, the construction of the simulator can be done by simply reading the adjacency information of the graph and replacing each node by some specific gadget. More precisely, we have that a CSAN is a graph G together with some circuit family which represents local functions (i.e. λ and ρ). Each local function will depend only on the local configuration composed by a node and its neighbors. In addition, for a fixed CSAN family, a collection of set-valued functions and edge-labels is provided independently of the structure of each graph as they depend on the possible sets of elements in the alphabet Q (note that what makes different two networks in one family is the position of labels but local functions are chosen from the same fixed set). Thus, we can represent a CSAN family $\mathcal{F}$ by a collection of labeled graphs $\mathcal{G}$ and a circuit family Λ. We call this representation a succinct representation for CSAN $\mathcal{F}$. Since circuit family Λ depends only in the size of the alphabet and since we are considering that the alphabet is fixed, we will usually omit Λ and just write $\mathcal{G}$ as a succinct representation for $\mathcal{F}$ and we will denote a CSAN $(G, \lambda, \rho) \in \mathcal{F}$ by simply G (where of course G is a labeled graph in $\mathcal{G}$). Note that since the alphabet size is constant, a succinct representation is, in particular, a standard representation for CSAN family as an automata network family. Formally, the language $L_{\mathcal{G}}$ containing the encoding of each labeled graph in $\mathcal{G}$ together with the standard encoding of constant circuit family Λ is a standard representation for CSAN family $\mathcal{F}$.

Bounded degree case.

A briefly note in the definition of a bounded degree family is recalled: let $k \in \mathbb{N}$, $\mathcal{F}$ be an automata network family and $\mathcal{G}$ be a collection of graphs such that for each $F \in \mathcal{F}$, F has an interaction graph $G_F \in \mathcal{G}$. We say $\mathcal{F}$ is a bounded degree family of degree $k \in \mathbb{N}$ if for each F, G_F satisfies $\Delta(G_F) \leq k$. Thus, a representation for F is simply an encoding of G as an adjacency list and a list of circuits $(G, \{f_{\mathrm{i}} : \mathrm{i} \in V(G_F)\}$ representing local functions. Note that each of these circuits has constant depth as the number of variables is at most k. Let $w_F \in \{0,1\}^*$ an encoding of $(G, \{f_{\mathrm{i}} : \mathrm{i} \in V(G_F)\}$ then, we define $\Gamma(G) : \{w_F : F \in \mathcal{F}\}$ and we call it a graph representation.

Let us fix some positive constant Δ. It is natural to consider the family of automata networks whose interaction graph has a maximum degree bounded by Δ (see Remark 3.5 in page 43). We associate to this family the following representation: an automata network F is given as pair $(G, (\tau_v)_{v \in V(G)})$ where G is a communication graph of F of maximum degree at most Δ and $(\tau_v)_{v \in V(G)}$ is the list for all nodes of G of its local transition map F_v of the form $Q^{\mathrm{d}} \to Q$ for $\mathrm{d} \leq \Delta$ and represented as a plain transition table of size $|Q|^{\mathrm{d}} \log(|Q|)$.

Remark 2.1 Given any CSAN family, there is a **DLOGSPACE** algorithm that transforms a bounded degree representation of an automata network of the family into a CSAN representation: since all local maps are bounded objects, it is just a matter of making a bounded computation for each node.

The algebraic case.

When endowing the alphabet Q with a finite field structure, the set of configurations Q^n is a vector space and one can consider automata networks that are actually linear maps. In this case the natural representation is a $n \times n$ matrix. It is clearly a standard representation in the above sense since circuit encodings can be easily computed from the matrix. Moreover, as in the CSAN case, when a linear automata network is given as a bounded degree representation, it is easy to recover its associated matrix in **DLOGSPACE**.

More generally, we can consider matrix representations without field structure on the alphabet. An interesting case is that of Boolean matrices: $Q = \{0, 1\}$ is endowed with the standard Boolean algebra structure with operations $\vee, \wedge$ and matrix multiplication is defined by:

$$(AB)_{\mathrm{i},j} = \bigvee_k A_{\mathrm{i},k} \wedge B_{k,j}.$$

They are a standard representation of disjunctive networks (and by switching the role of 0 and 1 conjunctive networks), *i.e.* networks F over alphabet $\{0, 1\}$ whose local maps are of the form $F_{\mathrm{i}}(x) = \vee_{k \in N(\mathrm{i})} x_k$ (respectively $F_{\mathrm{i}}(x) = \wedge_{k \in N(\mathrm{i})} x_k$) . When their dependency graph is symmetric, disjunctive networks (resp. conjunctive networks) are a particular case of CSAN networks for which ρ_v maps are the identity and λ_v are just max (resp. min) maps. For disjunctive networks (resp. conjunctive networks) the CSAN representation and the matrix representation are **DLOGSPACE** equivalent.

2.1.2 Computing interaction graphs from representations.

One of the key differences between all the representations presented so far is in the information they give about the interaction graph of an automata network. For instance, it is straightforward to deduce the interaction graph of a linear network from its matrix representation in **DLOGSPACE**. At the other extreme, one can see that it is NP-hard to decide whether a given edge belongs to the interaction graph of an automata network given by a circuit representation: indeed, one can build in **DLOGSPACE** from any SAT formula ϕ with n variables a circuit representation of an automata network $F : \{0, 1\}^{n+1} \to \{0, 1\}^{n+1}$ with

$$F(x)_1 = \begin{cases} x_{n+1} & \text{if } \phi(x_1, \ldots, x_n) \text{ is true,} \\ 0 & \text{else.} \end{cases}$$

This F is such that node 1 depends on node $n + 1$ if and only if ϕ is satisfiable.

For automata networks with communication graphs of degree at most Δ, there is a polynomial time algorithm to compute the interaction graph from a circuit representation: for each node v, try all the possible subsets S of nodes of size at most Δ and find the largest one such that the following map

$$x \in Q^S \mapsto F_v(\phi(x))$$

effectively depends on each node of S, where $\phi(x)_w$ is x_w if $w \in S$ and some arbitrary fixed state $q \in Q$ else. Note also, that we can compute a bounded degree representation in polynomial time with the same idea.

In the CSAN case, the situation is ambivalent. On the one hand, the interaction graph can be computed in polynomial time from a CSAN representation because for any given node v the following holds: either λ_v is a constant map and then v has no dependence, or λ_v is not constant and then the neighborhood of v in the CSAN representation is exactly the neighborhood of v in the interaction graph. On the other hand, a polynomial time algorithm to compute the interaction graph from a circuit representation would give a polynomial algorithm solving Unambiguous-SAT (which is very unlikely, following Valiant-Vazirani theorem [3, Theorem 9.15]). Indeed, any "dirac" map $\delta : \{0,1\}^n \to \{0,1\}$ with $\delta(x) = 1$ if and only if $x_1 \cdots x_n = b_1 \cdots b_n$ can be seen as the local map of a CSAN network because it can be written as $\lambda(\{\rho_{\mathrm{i}}(x_{\mathrm{i}}) : 1 \leq \mathrm{i} \leq n\})$ where $\rho_{\mathrm{i}}(x_{\mathrm{i}}) = x_{\mathrm{i}}$ if $b_{\mathrm{i}} = 1$ and $\neg x_{\mathrm{i}}$ else, and λ is the map

$$S \subseteq Q \mapsto \begin{cases} 1 & \text{if } S = \{1\} \\ 0 & \text{else.} \end{cases}$$

A constant map can also be seen as the local map of some CSAN network. Therefore, given a Boolean formula ϕ with the promise that is at has at most one satisfying assignment, one can easily compute the circuit representation of some CSAN network which has some edge in its interaction graph if and only if ϕ is satisfiable.

2.1.3 Standard representation

Recall that given a fixed alphabet Q, a family of automata networks is a collection of functions $\mathcal{F} = (F_s)_{s \in \mathbb{N}}$ satisfying that for each $s \in \mathbb{N}$ there exist some $n(s) \in \mathbb{N}$ such that $F_s : Q^{n(s)} \to Q^{n(s)}$ is an automata network.

We fix for any alphabet Q an injective map $m_Q : Q \to \{0,1\}^{k_Q}$ which we extend cell-wise for each n to $m_Q : Q^n \to \{0,1\}^{k_Q n}$. Given an automata network $F : Q^n \to Q^n$, a circuit encoding of F is a Boolean circuit $C : \{0,1\}^{k_Q n} \to \{0,1\}^{k_Q n}$ such that $m_Q \circ F = C \circ m_Q$ on Q^n. We also fix a canonical way to represent circuits as words of $\{0,1\}^*$ (for instance given by a number of vertices, the list of gate types positioned at each vertex and the adjacency matrix of the graph of the circuit).

Let $\mathcal{F}$ be a family of automata networks over alphabet Q. A standard representation $\mathcal{F}^*$ for $\mathcal{F}$ is a language $L_{\mathcal{F}} \subseteq \{0,1\}^*$ together with a **DLOGSPACE** algorithm such that:

- the algorithm transforms any $w \in L_{\mathcal{F}}$ into the canonical representation of a circuit encoding $C(w)$ that codes an automata network $F_w \in \mathcal{F}$;
- for any $F \in \mathcal{F}$ there is $w \in L_{\mathcal{F}}$ with $F = F_w$.

The default general representations we will use are circuit representations, *i.e.* representations where $w \in L_{\mathcal{F}}$ is just a canonical representation of a circuit. In this case the **DLOGSPACE** algorithm is trivial (the identity map). However, we sometimes want to work with more concrete and natural representations for some families of networks: in such a case, the above definition allows any kind of coding as soon as it is easy to deduce the canonical circuit representation from it.

From now on, a family of automata networks will be given as a pair $(\mathcal{F}, \mathcal{F}^*)$ where $\mathcal{F}$ is the set of automata networks and $\mathcal{F}^*$ a standard representation. In order to give some intuition regarding this latter general definition, examples of standard representations for specific families of automata networks are given in the next section. Particularly, bounded degree case and CSAN case are explored. In addition, we discuss how and when one can easily construct the circuit representation of an automata network from one of the latter representations.

Remark 2.2 It follows from the latter discussion on representations in the context of CSAN networks that a polynomial time algorithm to compute a CSAN representation of a CSAN represented by circuit would give a polynomial time algorithm to solve Unambiguous-SAT.

The following table synthesizes the latter discussion about representation conversions. It shall be read as follows: given a family $(\mathcal{F}, \mathcal{F}^*)$ listed horizontally and a family $(\mathcal{H}, \mathcal{H}^*)$ listed vertically, the corresponding entry in the table indicates the complexity of the problem of transforming $w \in L_F$ with the promise that $F_w \in \mathcal{F} \cap \mathcal{H}$ into $w' \in L_H$ such that $F_w = H_{w'}$.

output \ input	circuit	CSAN	Δ-bounded degree	matrix
circuit	trivial	**DLOGSPACE**	**DLOGSPACE**	**DLOGSPACE**
CSAN	USAT-hard	trivial	**DLOGSPACE**	**DLOGSPACE**
Δ-bounded degree	PTIME	PTIME	trivial	**DLOGSPACE**

USAT-hard means that any PTIME algorithm would imply a PTIME algorithm for Unambiguous-SAT.

2.2 Simulation and universality

In this section, we present a background in simulation of automata networks. We start by presenting a definition that involves mainly automata networks as dynamical systems. More precisely, we consider a simulation by blocks which is a classical approach in the context of simulation in discrete dynamical systems. Then, we present a notion of simulation between families of automata networks. The latter notion considers the amount of resources required to perform the simulation such as time and space and also describes simulation in terms of the latter defined standard representation. Finally, we discuss universality and implications of this definition in terms of computational resources and dynamical properties.

2.2.1 Simulation between automata networks

We present in this subsection a definition of simulation which is based on blocks. Generally speaking, given two automata networks, one being the simulated network and one being the simulator, we represent nodes in the simulated network by block of nodes in the simulator (which induces subgraphs in the communication graph of the simulator). In addition, we encode the state of a particular node in the simulated network by a group of states in the alphabet of the simulator. Particularly, we choose this assignation to be injective, induc-

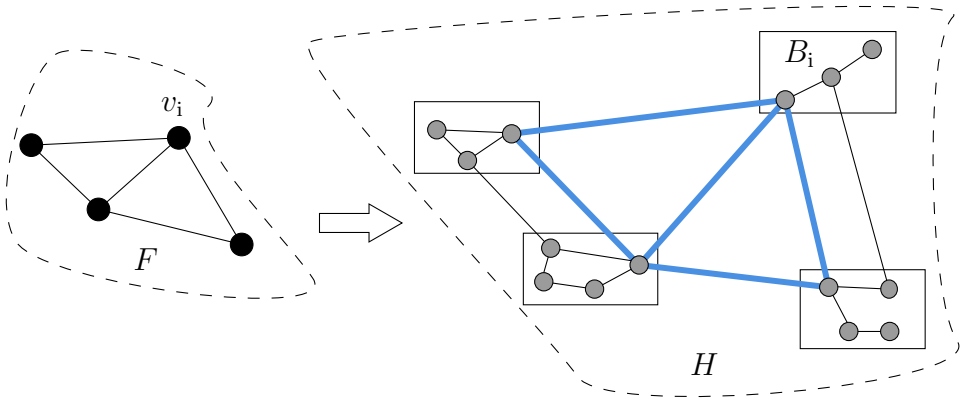


Figure 2.1: Scheme of one-to-one block simulation. In this case, network F is simulated by H. Each node in F is assigned to a block in H and state coding is injective. Observe that blocks are connected (one edge in the original graph may be represented by a path in the communication graph of H) according to connections between nodes in the original network F. This connections are represented by blue lines.

ing certain injective code map. A similar approach was broadly studied in the context of simulation between cellular automata [49, 8, 60, 64, 65]. Note that defining an injective coding map is not arbitrary. It allows us to avoid problems that could arrive in the context of transferring dynamical properties from a simulated network to the simulator such as, for example, preserving the transient lengths and the attractor periods. In addition, it makes the notion of simulation compatible with reductions in the context of comparing the computational complexity of the prediction problem between the simulator network and the simulated network.

Definition 2.3 *Let $F : Q_F^{V_F} \to Q_F^{V_F}$ and $H : Q_H^{V_H} \to Q_H^{V_H}$ be automata networks. A* block embedding *of $Q_F^{V_F}$ into $Q_H^{V_H}$ is a collection of blocks $D_{\mathrm{i}} \subseteq V_H$ for each $\mathrm{i} \in V_F$ which forms a partition of V_H together with a collection of patterns $p_{\mathrm{i},q} \in Q_H^{D_{\mathrm{i}}}$ for each $\mathrm{i} \in V_F$ and each $q \in Q_F$ such that $p_{\mathrm{i},q} = p_{\mathrm{i},q'}$ implies $q = q'$. This defines an injective map $\phi : Q_F^{V_F} \to Q_H^{V_H}$ by $\phi(x)_{D_{\mathrm{i}}} = p_{\mathrm{i},x_{\mathrm{i}}}$ for each $\mathrm{i} \in V_F$. We say that H simulates F via block embedding ϕ if there is a time constant T such that the following holds on $Q_F^{V_F}$:*

$$\phi \circ F = H^T \circ \phi.$$

See Figure 2.1 for a scheme of block simulation. In the following, when useful we represent a block embedding as the list of blocks together with the list of patterns. The size of this representation is linear in the number of nodes (for a fixed alphabet).

Remark 2.4 It is convenient in many concrete cases to define a block embedding through blocks D_{i} that are disjoint but do not cover V_H and add a context block C disjoint from the D_{i} that completes the covering of V_H. In this variant a block embedding of $Q_F^{V_F}$ into $Q_H^{V_H}$ is given by patterns $p_{\mathrm{i},q}$ and a constant context pattern $p_C \in Q_H^C$ which define an injective map $\phi : Q_F^{V_F} \to Q_H^{V_H}$ by $\phi(x)_{D_{\mathrm{i}}} = p_{\mathrm{i},x_{\mathrm{i}}}$ for each $\mathrm{i} \in V_F$ and $\phi(x)_C = p_C$. This variant is actually

just a particular instance of Definition 2.3 because we can include C in an arbitrary block $(D_\mathrm{i} \leftarrow D_\mathrm{i} \cup C)$ and define the block embedding as in Definition 2.3.

In our proofs of simulations (see next subsections), the variant blocks/context will be particularly relevant because the size of blocks will be bounded while the context will grow with the size of the considered automata. Said differently, the information about an encoded state will be very localized.

Another natural particular case of Definition 2.3 corresponding to localized information is when in each block D_i, there is a special node $v_\mathrm{i} \in D_\mathrm{i}$ such that the map $q \mapsto p_{\mathrm{i},q}(v_\mathrm{i})$ is injective. It is only possible when Q_H is larger than Q_F, but it will be the case in several examples of Boolean automata networks (see Chapter 5) . Interestingly, this local coding phenomena is forced when some automata network H simulates some Boolean automata network F: indeed, in any block D_i of H at least one node v_i must change between patterns $p_{\mathrm{i},0}$ and $p_{\mathrm{i},1}$, but the map $x \in \{0,1\} \mapsto p_{\mathrm{i},x}$ being injective, it means that $x \mapsto p_{\mathrm{i},x}(v_\mathrm{i})$ is injective too.

Remark 2.5 The simulation relation of Definition 2.3 is a pre-order on automata networks.

We now observe that simulation can also be seen in terms of orbit graphs. In this sense, we present the following lemma:

Lemma 2.6 *If H simulates F via block embedding with time constant T then the orbit graph G_F of F is a subgraph of G_H^T. In particular if F has an orbit with transient of length t and period of length p, then H has an orbit with transient of length Tt and period Tp.*

Proof. The embedding of G_F inside G_H^T is realized by definition by the block embedding of the simulation. The consequence on the length of periods and transients comes from the fact that the embedding ϕ verifies: x is in a periodic orbit if and only if $\phi(x)$ is in a periodic orbit. □

2.2.2 Simulation between families of automata networks

In this section we present a notion of simulation in terms of representations and the amount of resources that are needed in order to simulate a certain family of automata networks by some other arbitrary family. Roughly speaking, a family of automata network can simulate a particular automata network if we are able to *effectively* construct some automata network in this collection that is able to simulate the first automata network in the sense of the latter definition of simulation. More precisely, we ask the automata network which performs the simulation to do this task in reasonable time and reasonable space in a sense that we specify in the next definition.

Definition 2.7 *Let $(\mathcal{F}, \mathcal{F}^*)$ and $(\mathcal{H}, \mathcal{H}^*)$ be two families with standard representations on alphabets Q_F and Q_H respectively. Let $T, S : \mathbb{N} \to \mathbb{N}$ be two functions. We say that $\mathcal{F}^*$*

simulates $\mathcal{H}^*$ *in time* T *and space* S *if there exists a* ***DLOGSPACE*** *Turing machine* M *such that for each* $w \in L_{\mathcal{H}}$ *representing some automata network* $H_w \in \mathcal{H} : Q_H^n \to Q_H^n$*, the machine produces a pair* $M(w)$ *which consists in:*

- $w' \in L_{\mathcal{F}}$ *with* $F_{w'} : Q_F^{n_F} \to Q_F^{n_F}$*,*
- *a representation of a block embedding* $\phi : Q^{n_F} \to Q^n$*,*

such that $n_F = S(n)$ *and* $F_{w'}$ *simulates* H_w *in time* $T = T(n)$ *under block embedding* ϕ*.*

From now on, whenever $\mathcal{F}^*$ simulates $\mathcal{H}^*$ in time T and space S we write $\mathcal{H}^* \preccurlyeq_S^T \mathcal{F}^*$.

Remark 2.8 The simulation relation between families is transitive because the class **DLOGSPACE** is closed under composition and simulation between individual automata networks is also transitive. When composing simulations time and space maps S and T get multiplied.

2.3 Decision problems and automata network dynamics

In this section we introduce three variants of a classical decision problem that is closely related to the dynamical behavior of automata networks: the prediction problem. This problem consists in predicting if one node of the network will change its state at given time. We study first a short term version of the problem in which time is given in the input. In addition, we explore a stronger variant of this problem in which we ask if some node has eventually changed, provided a constant observation time rate τ. In other words, we check the system for any change on the state of a particular node every multiple of τ time steps. Finally, we conclude that these problems are coherent with our simulation definition in the sense that if some family of automata networks $(\mathcal{F}_2, \mathcal{F}_2^*)$ simulates $(\mathcal{F}_1, \mathcal{F}_1^*)$ then, if some of the latter problem is hard in some sense for $\mathcal{F}_1$ it will also be hard for $\mathcal{F'}_2$. We will precise this result in the following lines.

Let $(\mathcal{F}, \mathcal{F}^*)$ be an automata network family and let $L \in \{0,1\}^* \times \{0,1\}^*$ be a parameterized language. We say that L is parameterized by $\mathcal{F}$ if L has $\mathcal{F}^*$ encoded as parameter. We note $L_{\mathcal{F}} \subseteq \{0,1\}^*$ as the language resulting on fixing $\mathcal{F}^*$ as a constant.

In particular, we are interested in studying prediction problems. We start by defining two variants of this well-known decision problem:

Problem (Unary Prediction (PREDu))

Parameters: alphabet Q, a standard representation $\mathcal{F}^*$ of an automata network family $\mathcal{F}$

Input:

1. a word $w_F \in \mathcal{F}^*$ representing an automata network $F : Q^n \to Q^n$ with $F \in \mathcal{F}$.
2. a node $v \in V(F) = \{1, \ldots, n\}$

3. an initial configuration $x \in Q^V$.
4. a natural number t represented in unary such that $t \in \mathbf{1}$.

Question: $F^t(x)_v \neq x_v$?

Problem (Binary Prediction (PREDb))

Parameters: alphabet Q, a standard representation $\mathcal{F}^*$ of an automata network family $\mathcal{F}$

Input:

1. a word $w_F \in \mathcal{F}^*$ representing an automata network $F : Q^n \to Q^n$ with $F \in \mathcal{F}$.
2. a node $v \in V(F) = [n]$
3. an initial configuration $x \in Q^V$.
4. a natural number t represented in binary $t \in \{0,1\}^*$.

Question: $F^t(x)_v \neq x_v$?

Note that two problems are essentially the same, the only difference is the representation of time t that we call the *observation time*. We will also call node v the objective node. As it happens with other decision problems, such as integer factorization, the representation of observation time will have an impact on the computation complexity of the prediction problem. When the context is clear we will refer to both problems simply as PRED. Observe that one can always solve the prediction problem by simply computing $F^t(x)$ in order to verify if $F^t(x)_v \neq x_v$. Thus, since F is deterministic, we have that $\text{PRED}_{\text{U}\mathcal{F}} \in \mathbf{P}$ and $\text{PRED}_{\text{B}\mathcal{F}} \in \mathbf{PSPACE}$.

Finally, we show that latter problem is coherent with our definition of simulation, in the sense that we can preserve the complexity of PRED. Note that this gives us a powerful tool in order to classify concrete automata rules according to the complexity of the latter decision problem.

Lemma 2.9 *Let $(\mathcal{F}, \mathcal{F}^*)$ and $(\mathcal{H}, \mathcal{H}^*)$ be two automata network families. Let $T, S : \mathbb{N} \to \mathbb{N}$ be two polynomial functions such that $\mathcal{H}^* \preccurlyeq_S^T \mathcal{F}^*$ then,* $\text{PRED}_{\mathcal{H}^*} \leq_{\boldsymbol{L}}^T \text{PRED}_{\mathcal{F}^*}$[1]

Proof. Let (w_H, v, x, t) be an instance of $\text{PRED}_{\mathcal{H}^*}$. By definition of simulation, there exists a **DLOGSPACE** algorithm which takes w_H and produces a word $w_F \in L_{\mathcal{F}}$ with $F : Q^{n_F} \to Q^{n_F}$ and a block representation $\phi : Q^{n_F} \to Q^n$ such that $n_F = S(n)$ and F simulates H in time $T(n)$ under block embedding ϕ. Particularly, there exists a partition of blocks $D_v \subseteq V(F) = [n_F]$ for each $v \in V(H) = [n]$ and a collection of injective patterns, i.e. patterns $p_{\text{i},q} \in Q_F^{D_\text{i}}$ such that $p_{\text{i},q} = p_{\text{i},q'} \implies q = q'$. In addition, we have $\phi \circ H = F^T \circ \phi$. Let us define the configuration $y \in Q_F^{n_F}$ as $y_{D_\text{i}} = p_{\text{i},x_\text{i}}$, i.e., $\phi(x)_{D_\text{i}} = y_{D_\text{i}}$. Note that y is well-defined as the block map is injective. In addition, let us choose an arbitrary vertex $v' \in D_v$ and let us consider

[1]Here we denote $\leq_{\mathbf{L}}^T$ as a **DLOGSPACE** Turing reduction. The capital letter "T" stands for *Turing reduction* and it is not related to the simulation time function which is also denoted by T.

now the instance of $\mathrm{PRED}_{\mathcal{F}^*}$ given by $(w_F, v', y, t \times T)$. Note that for each $v' \in D_v$ the transformation $(w_H, v, x, t) \to (w_F, v', y, t \times T)$ can be done in **DLOGSPACE**$(|w_H|)$ because we can read the representation of ϕ for each block $p_{\mathrm{i},x_\mathrm{i}}$ and then output the configuration y. We claim that there exists a **DLOGSPACE**$(|w_H|)$ algorithm that decides if $(w_H, v, x, t) \in L_{\mathrm{PRED}_{\mathcal{H}^*}}$ with oracle calls to $\mathrm{PRED}_{\mathcal{F}^*}$. More precisely, as a consequence of the injectivity of block embedding, we have that $(w_H, v, x, t) \in L_{\mathrm{PRED}_{\mathcal{H}^*}}$ if and only if $(w_F, v', y, t \times T) \in L_{\mathrm{PRED}_{\mathcal{F}^*}}$ for some $v' \in D_v$. In fact, the latter algorithm just runs oracle calls of $\mathrm{PRED}_{\mathcal{F}^*}$ for $(w_F, v', y, t \times T)$ for each $v' \in D_v$ and decides if some of these instances is a YES instance and thus, if some node in the simulation block have changed its state. As block-embedding is injective, this necessary means that node v have changed its state in t time steps. Finally, all of this can be done in **DLOGSPACE** since $n_F = S(n) = n^{\mathcal{O}(1)}$ and $T = n^{\mathcal{O}(1)}$ and thus, a polynomial amount of calls to each oracle is needed. □

Finally, we would like to study the case in which time is not part of the entry of PRED. In other words, we would like to observe the long-term dynamical behavior of the objective node. However, in order to preserve complexity properties under simulation, we still need to have some sort of observation gap as a part of the entry of PRED. This will allow us to avoid giving misleading answers when the simulating network is performing one step of simulation. We recall that it could take several time steps for the simulating network in order to represent one step of the dynamics of the simulated network. In order to manage this sort of time dilatation phenomenon between simulating and simulated systems, we introduce the following decision problem:

Problem (Prediction change $\mathrm{PRED}\textsc{c}_{\mathcal{F}^*}$)

Parameters: alphabet Q, a standard representation $\mathcal{F}^*$ of an automata network family $\mathcal{F}$.

Input:

1. a word $w_F \in \mathcal{F}^*$ representing an automata network $F : Q^V \to Q^V$ with $F \in \mathcal{F}$ and $V = \{1, \ldots, n\}$.
2. a node $v \in V$
3. an initial configuration $x \in Q^V$.
4. a time gap $k \in \mathbf{1}$

Question: $\exists t \in \mathbb{N} : F^{kt}(x)_v \neq x_v$

Observe that, again, the latter decision problem can be solved in **PSPACE** by direct computation of $F^{kt}(x)$ for different values of t until the orbit reaches an attractor. For each t, it is possible to very if objective node has eventually changed or not. Now we show that the computational complexity of the latter problem is consistent under our notion of simulation.

Lemma 2.10 *Let $(\mathcal{F}, \mathcal{F}^*)$ and $(\mathcal{H}, \mathcal{H}^*)$ be two automata network families and $T, S : \mathbb{N} \to \mathbb{N}$ two polynomial functions such that $\mathcal{H}^* \preccurlyeq_S^T \mathcal{F}^*$ then, $\mathrm{PRED}\textsc{c}_{\mathcal{H}^*} \leq_{\boldsymbol{L}}^T \mathrm{PRED}\textsc{c}_{\mathcal{F}^*}$.*

Proof. The proof is analogous to short term prediction case. Let (w_H, v, x) be an instance of $\mathrm{PRED}_{\mathcal{H}^*}$. Again, by the definition of simulation, there exists a **DLOGSPACE** algorithm which takes w_H and produces a word $w_F \in L_{\mathcal{F}}$ with $F : Q^{n_F} \to Q^{n_F}$ and a block representation $\phi : Q^{n_F} \to Q^n$ such that $n_F = S(n)$ and F simulates H in time $T(n)$ under block embedding ϕ. The latter statements means, particularly, that there exists a partition of blocks $D_v \subseteq V(F) = [n_F]$ for each $v \in V(H) = [n]$ and a collection of injective patterns, i.e. patterns $p_{\mathrm{i},q} \in Q_F^{D_\mathrm{i}}$ such that $p_{\mathrm{i},q} = p_{\mathrm{i},q'} \implies q = q'$ and also that $\phi \circ H = F^T \circ \phi$. Let us define the configuration $y \in Q_F^{n_F}$ as $y_{D_\mathrm{i}} = p_{\mathrm{i},x_\mathrm{i}}$, i.e., $\phi(x)_{D_\mathrm{i}} = y_{D_\mathrm{i}}$. Note, again, that y is well-defined as the block map is injective. Now we proceed by using the same approach than before: for each $v' \in D_v$ we can produce an instance (w_F, v', y) of $\mathrm{PREDC}_{\mathcal{F}^*}$. Then, as T and S are polynomials, there exists an oracle **DLOGSPACE** machine which produces (w_F, v', y) for each $v' \in D_v$ and calls for an oracle $\mathrm{PREDC}_{\mathcal{F}^*}(w_F, v', y)$. Then, algorithm verifies whether there is a YES-instance for some v'. By definition of simulation and injectivity of block embedding function we have that this algorithm outputs 1 if and only if $(w_H, v, x) \in \mathrm{PRED}_{\mathcal{H}^*}$. □

2.4 Universal automata network families

In the next lines, we will give the definition of a universal family of automata networks. In simple words, a universal family is a set of automata networks which is able to simulate every other automata network. Of course, in this context, we precise the amount of resources that some family will need in order to simulate an arbitrary automata network by controlling functions S and T (otherwise we could take families that use an enormous amount of resources as simulators). In addition, it is important to remark again the essential role of representations. When we say that some family can simulate every other family, we have to be careful with the fact that, as we are choosing mainly circuits as a representation for automata networks, the size of a circuit which simulates certain automata network could be exponential in the size of the network. Thus, as we are going to use the circuit as a main construction block in the process of constructing a simulator, this latter observation will have an important role in the moment to express the properties of the simulator in terms of the simulated network. In order to avoid this issue, we are very specific in the notion of universal family of automata networks by adding a constraint in the size of the circuit representing an automata network in this family.

Building upon our definition of simulation, we can now define a precise notion of universality. In simple words, a universal family is one that is able to simulate every other automata network under any circuit encoding. Our definition of simulation ensures that the amount of resources needed in order to simulate is controlled so that we can deduce precise complexity results.

Consider some alphabet Q and some polynomial map $P : \mathbb{N} \to \mathbb{N}$. We denote by $\mathcal{U}_{Q,P}$ the class of all possible functions $F : Q^n \to Q^n$ for any $n \in \mathbb{N}$ that admits a circuit representation of size at most $P(n)$. We also denote $\mathcal{U}^*_{Q,P}$ the language of all possible circuit representations of size bounded by P of all functions from $\mathcal{U}_{Q,P}$. Finally for any $\Delta \geq 1$, denote by $\mathcal{B}_{Q,\Delta}$ the set of automata networks on alphabet Q with a communication graph of degree bounded by Δ and by $\mathcal{B}^*_{Q,\Delta}$ their associated bounded degree representations made of a pair (graph, local maps) as discussed above.

Definition 2.11 *A family of automata networks* $(\mathcal{F}, \mathcal{F}^*)$ *is :*

- ***universal*** *if for any alphabet* Q *and any polynomial map* P *it can simulate* $(\mathcal{U}_{Q,P}, \mathcal{U}^*_{Q,P})$ *in time* T *and space* S *where* T *and* S *are polynomial functions;*
- ***strongly universal*** *if for any alphabet* Q *and any degree* $\Delta \geq 1$ *it can simulate* $(\mathcal{B}_{Q,\Delta}, \mathcal{B}^*_{Q,\Delta})$ *in time* T *ans space* S *where* T *and* S *are linear maps.*

Remark 2.12 The link between the size of automata networks and the size of their representation is the key in the above definitions. A universal family must simulate any individual automata network F (just take P large enough so that $F \in \mathcal{U}_{Q,P}$). However it is not required to simulate in polynomial space and time a family of networks whose smallest circuit representations are super-polynomial. Actually no family admitting polynomial circuit representation could simulate such a super-polynomial family in polynomial time and space. In particular the family $\mathcal{B}_{Q,\Delta}$ cannot.

At this point it is clear, by transitivity of simulations, that if some $\mathcal{B}_{Q,\Delta}$ happens to be universal then, any strongly universal family is also universal. It turns out that $\mathcal{B}_{\{0,1\},3}$ is universal. We will however delay the proof until Chapter 3 where we prove a more precise result which happens to be very useful to get universality result in concrete families.

Now, we introduce an important corollary of universality regarding complexity. Roughly speaking, a universal family exhibits all the complexity in terms of dynamical behavior and computational complexity of prediction problems. Concerning computational complexity, let us introduce the following definition.

Definition 2.13 *We say that an automata network family* $\mathcal{F}$ *is computationally complex if the following conditions hold:*

1. $\text{PREDU}_{\mathcal{F}}$ *is* ***P****-hard.*
2. $\text{PREDB}_{\mathcal{F}}$ *is* ***PSPACE****-hard.*
3. $\text{PREDC}_{\mathcal{F}}$ *is* ***PSPACE****-hard.*

As a direct consequence of universality, we have the following result where there is no difference between strong or standard universality.

Corollary 2.14 *Let* $\mathcal{F}$ *be a (strongly) universal automata network family. Then,* $\mathcal{F}$ *computationally complex.*

Proof. We show that there exists a bounded degree automata network family which is dynamically complex. Then, any universal family will also satisfy each of latter properties as a direct consequence of the simulation. In fact, we cite a classical result in simulation of Turing Machines by elementary cellular automata [57] in which it is shown that it is possible to simulate an arbitrary Turing Machine with n states and m symbols in space $\mathcal{O}(n+m)$ and in twice linear time. This is accomplished by adding two auxiliary states which decou-

ple interactions and movements of the machine in two different time steps, thus, if machine performs t-steps of computation, the simulator will need $2t$-steps. As a direct consequence of this, we have that it is possible to define an automata network family which simulates in linear time and space every linear bounded space Turing Machine and thus, results on complexity of the prediction problem and on transient length and attractor period follow as a direct consequence.

□

In addition, reader interested in simulation results which does not involve cellular automata but automata networks having a more general graph structure, we provide following references in which authors have presented alternative simulation schemes by using very well-known Boolean network families as threshold networks:

1. The family Θ of threshold networks over the binary alphabet $\{0,1\}$ satisfies that PREDU_Θ is **P**-hard [37].
2. The family Θ of threshold networks over an alphabet $Q = \{0,1\}$ (equipped with block sequential update schemes, see Chapter 4) satisfies that PREDC_Θ is **PSPACE**-hard [40]

We now turn to the dynamical consequences of universality. By definition simulations are particular embeddings of orbit graphs into larger ones, but the parameters of the simulation can involve some distortions.

Definition 2.15 *Fix a map $\rho : \mathbb{N} \to \mathbb{N}$, we say that the orbit graph G_F of F with n nodes is* ρ-succinct *if F can be represented by circuits of size at most $\rho(n)$. We say that the orbit graph G_H of H with m nodes* embeds *G_F with distortion $\varepsilon : \mathbb{N} \to \mathbb{N}$ if $m \leq \varepsilon(n)$ and there is $T \leq \varepsilon(n)$ such that G_F is a subgraph of G_{H^T}.*

Remark 2.16 The embedding of orbit graphs with distortion obviously modify the relation between the number of nodes of the automata netwroks and the length of paths or cycles in the orbit graph. In particular, with polynomial distortion ε, if F has n nodes and a cyclic orbit of length 2^n (hence exponential in the number of nodes) then in H it gives a cyclic orbit of size $O(\varepsilon(n)2^n)$ for up to $\varepsilon(n)$ nodes, which does not guarantee an exponential length in the number of nodes in general, but just a super-polynomial one ($n \mapsto 2^{n^\alpha}$ for some $0 < \alpha < 1$).

To fix ideas, we give examples of orbit graphs of bounded degree automata networks with large components corresponding to periodic orbits or transients.

Proposition 2.17 *There is an alphabet Q such that for any $n \geq 1$ there is an automata network $F_n \in \mathcal{B}_{Q,2}$ whose orbit graph G_{F_n} has the following properties:*

- *It contains a cycle C of length at least 2^n.*
- *There is a complete binary tree T with 2^n leaves connected to some $v_1 \in C$, i.e. for all*

$v \in T$ *there is a path from* v *to* v_1.

- *There is a node* $v_2 \in C$ *with a directed path of length* 2^n *pointing towards* v_2.

Proof. First on a component of states $\{0,1,2\} \subseteq Q$ the large cycle C is obtained by the following "odometer" behavior of F_n: if $x_n \in \{0,1,2\}$ then $F_n(x)_n = x_n + 1 \bmod 3$, and if both $x_\mathrm{i}, x_{\mathrm{i}+1} \in \{0,1,2\}$ for $1 \leq \mathrm{i} < n$ then

$$F_n(x)_\mathrm{i} = \begin{cases} 0 & \text{if } x_\mathrm{i} = 2, \\ x_\mathrm{i} + 1 & \text{else if } x_{\mathrm{i}+1} = 2, \\ x_\mathrm{i} & \text{else.} \end{cases}$$

C is realized on $\{0,1,2\}^n$ as follows. For $x \in \{0,1,2\}^n$ we denote by S_i the sequence $(F^t(x)_\mathrm{i})_{t \geq 0}$ for any $1 \leq \mathrm{i} \leq n$. Clearly S_n is periodic of period 012. S_{n-1} is ultimately periodic of period 200111 (of length 6) and by a straighforward induction we get that S_1 is ultimately periodic of period $20^{3 \cdot 2^{n-2}-1}1^{3 \cdot 2^{n-2}}$ which is of length $3 \cdot 2^{n-1}$.

For the tree T, just add states $\{a,b\} \subseteq Q$ with the following behavior: if $x_1 \in \{a,b\}$ then $F_n(x)_1 = 0$ and if $x_\mathrm{i} \in \{a,b\}$ and $x_{\mathrm{i}-1} = 0$ then $F_n(x)_\mathrm{i} = 0$ for $1 < \mathrm{i} \leq n$. In any other case, we set $F(x)_\mathrm{i} = x_\mathrm{i}$ for $x \in \{0,1,2,a,b\}^n$ and $1 \leq \mathrm{i} \leq n$.

Finally, using similar mechanisms as above on additional states $c_0, c_1, c_2 \in Q$, F_n runs another odometer whose behavior is isomorphic to the behavior of F_n on $\{0,1,2\}^n$ through $\mathrm{i} \mapsto c_\mathrm{i}$, but with the following exception: when $x_1 = c_2$ we set $F_n(x)_1 = 0$ and then state 0 propagates from node 1 to node n as in the construction of tree T. We thus get a transient behavior of length more than $3 \cdot 2^{n-1}$ which yields to configuration 0^n, which itself (belongs or) yields to cycle C. □

Now we can now state that any universal family must be dynamically rich in a precise sense.

Theorem 2.18 *Let* $\mathcal{F}$ *be an automata networks family.*

- *If* $\mathcal{F}$ *is universal then, for any polynomial map* ρ*, there is a polynomial distortion* ε *such that, any* ρ*-succinct orbit graph can be embedded into some* $F \in \mathcal{F}$ *with distortion* ε*. In particular* $\mathcal{F}$ *contains networks with super-polynomial periods and transients.*
- *If* $\mathcal{F}$ *is strongly universal then it embeds the orbit graph of any bounded-degree automata network with linear distortion. In particular it contains networks with exponential periods and transients.*

Proof. This is a direct consequence of Lemma 2.6, Definition 2.7 and Proposition 2.17 above. □

In a directed graph, we say that a node v belongs to a strongly connected component if there is a directed path from v to v.

Corollary 2.19 *Any universal family $\mathcal{F}$ satisfies the following: there is a constant α with $0 < \alpha \leq 1$ such that for any $m > 0$ there is a network $F \in \mathcal{F}$ with $n \geq m$ nodes such that there exists some node v belonging to a strongly connected component of the interaction graph of F and a periodic configuration x such that the trace at v of the orbit of x is of period at least 2^{n^α}.*

Proof. Consider a Boolean network F with n nodes numbered by $\{1, \dots, n\}$ that does the following on configuration x: it interprets $x_1, \dots, x_n$ as a number k written in base 2 where x_1 is the most significant bit and define $F(x)$ so that $F(x)_{x_1,\dots,x_n}$ represents number $k + 1 \bmod 2^n$ with node 1 holding the most significant bit. F is such that node 1 has trace of exponential period in some periodic orbit and it is in a strongly connected component since it depends on itself. Note that F has a circuit representation which is polynomial in n, and take $F' \in \mathcal{F}$ of polynomial size in n that simulates F in polynomial time. Taking the notations of Definition 2.3, we have that each node $v \in D_{\mathrm{i}}$ for each block D_{i} is such that the map $q \in \{0, 1\} \mapsto p_{\mathrm{i},q}(v)$ is either constant or bijective (because F has a Boolean alphabet, see Remark 2.4). In the last case, the value of the node $v \in D_{\mathrm{i}}$ completely codes the value of the corresponding node i in F. Take any $v \in B_1$ that has this coding property. Since node 1 depends on itself in F, there must be a path from v to some coding node $v' \in D_1$. Then, again, as node 1 depends on itself in F, there must be a path from v' to some coding node in D_1. Iterating this reasoning we must find a cycle, and in particular we have a coding node in D_1 which belongs to some strongly connected component of the interaction graph of F'. Since this node is coding the values taken by node 1 of F and since the simulation is in polynomial time and space, we deduce the super-polynomial lower bound on the period of its trace for a well-chosen periodic configuration. □

Chapter 3

Gadget complexity: from local to global behavior

3.1 Putting pieces together: glueing automata networks

In this section we define an operation that allows us to "glue" two different abstract automata networks on a common part in order to create another one which, preserve some dynamical properties in the sense that it allows to glue pseudo-orbits of each network to obtain a pseudo-orbit of the glued network. One might find useful to think about the common part of the two networks as a dowel attaching two pieces of wood: each individual network is a piece of wood with the dowel inserted in it, and the result of the glueing is the attachment of the two pieces with a single dowel (see Figure 3.1). The idea is that, in this scheme, each node in the glued network will dynamically behave as same as a node in the first network or as a node in the second network.

Definition 3.1 *Consider* $F_1 : Q^{V_1} \to Q^{V_1}$ *and* $F_2 : Q^{V_2} \to Q^{V_2}$ *two automata networks with* V_1 *disjoint from* V_2, C *a set disjoint from* $V_1 \cup V_2$, $\varphi_1 : C \to V_1$ *and* $\varphi_2 : C \to V_2$ *two injective maps and* C_1, C_2 *a partition of* C *in two sets. We define*

$$V' = C \cup (V_1 \setminus \varphi_1(C)) \cup (V_2 \setminus \varphi_2(C))$$

and the map $\alpha : V' \to V_1 \cup V_2$ *by*

$$\alpha(v) = \begin{cases} v & \text{if } v \notin C \\ \varphi_{\mathrm{i}}(v) & \text{if } v \in C_{\mathrm{i}}, \text{ for } \mathrm{i} = 1, 2. \end{cases}$$

We then define the glueing *of* F_1 *and* F_2 *over* C *as the automata network* $F' : Q^{V'} \to Q^{V'}$ *where*

$$F'_v = \begin{cases} (F_1)_{\alpha(v)} \circ \rho_1 & \text{if } \alpha(v) \in V_1, \\ (F_2)_{\alpha(v)} \circ \rho_2 & \text{if } \alpha(v) \in V_2, \end{cases}$$

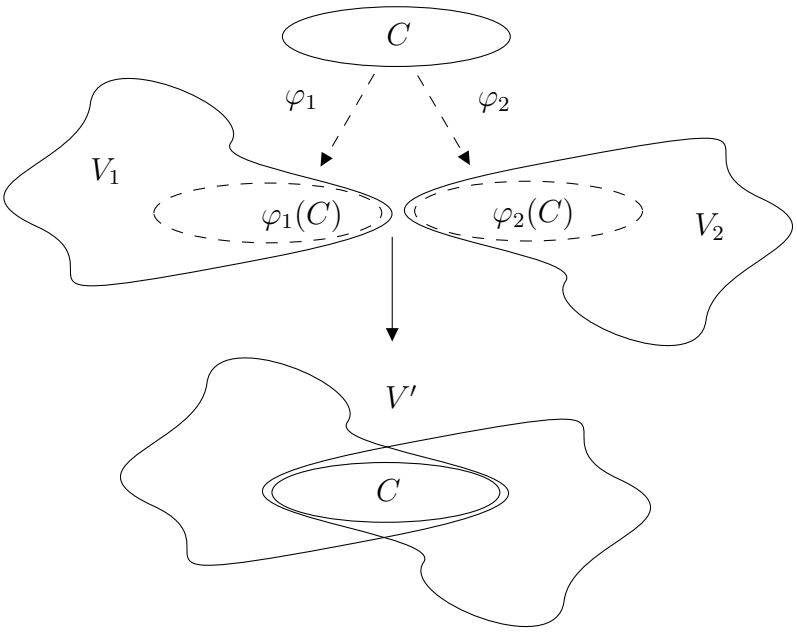


Figure 3.1: General scheme of a glueing.

where $\rho_i : Q^{V'} \to Q^{V_i}$ *is defined by*

$$\rho_i(x)_v = \begin{cases} x_{\varphi_i^{-1}(v)} & \text{if } v \in \varphi_i(C), \\ x_v & \text{else.} \end{cases}$$

When necessary, we will use the notation $F' = F_1 \overset{\phi_1}{\underset{C_1}{\oplus}}\overset{\phi_2}{_{C_2}} F_2$ to underline the dependency of the glueing operation on its parameters.

Given an automata network $F : Q^V \to Q^V$ and a set $X \subseteq V$, we say that a sequence $(x^t)_{0 \leq t \leq T}$ of configurations from Q^V is a X-pseudo-orbit if it respects F as in a normal orbit, except on X where it can be arbitrary, formally: $x_v^{t+1} = F(x^t)_v$ for all $v \in V \setminus X$ and all $0 \leq t < T$. The motivation for Definition 3.1 comes from the following lemma.

Lemma 3.2 (Pseudo-orbits glueing) *Taking the notations of Definition 3.1, let* $X \subseteq V_1 \setminus \varphi_1(C)$ *and* $Y \subseteq V_2 \setminus \varphi_2(C)$ *be two (possibly empty) sets. If* $(x^t)_{0 \leq t \leq T}$ *is a* $X \cup \varphi_1(C_2)$*-pseudo-orbit for* F_1 *and if* $(y^t)_{0 \leq t \leq T}$ *is a* $Y \cup \varphi_2(C_1)$*-pseudo-orbit for* F_2 *and if they verify for all* $0 \leq t \leq T$ *that:*

$$\forall v \in C, x^t_{\varphi_1(v)} = y^t_{\varphi_2(v)}, \tag{3.1}$$

then the sequence $(z^t)_{0 \leq t \leq T}$ *of configurations of* $Q^{V'}$ *is a* $X \cup Y$*-pseudo-orbit of* F'*, where*

$$z_v^t = \begin{cases} x^t_{\alpha(v)} & \text{if } \alpha(v) \in V_1, \\ y^t_{\alpha(v)} & \text{if } \alpha(v) \in V_2. \end{cases}$$

Proof. Take any $v \in V' \setminus (X \cup Y)$. Suppose first that $\alpha(v) \in V_1$. By definition of F', we have $F'(z^t)_v = (F_1)_{\alpha(v)} \circ \rho_1(z^t)$ but $\rho_1(z^t) = x^t$ (using the Equation 3.1 in the hypothesis) so

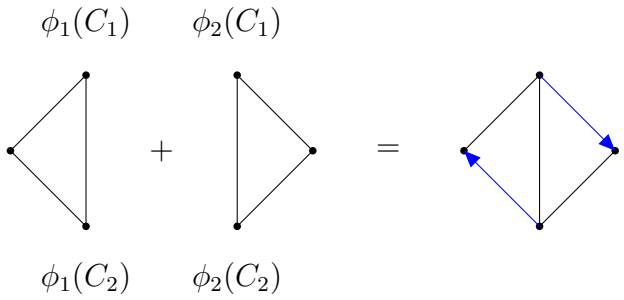


Figure 3.2: Symmetry breaking in interaction graph after a glueing operation. Arrows indicate influence of a node (source) on another (target), edges without arrow indicates bi-directional influence. Here C consists in two nodes only.

$F'(z^t)_v = F_1(x^t)_{\alpha(v)}$. Since (x^t) is a $X \cup \varphi_1(C_2)$-pseudo-orbit and since $\alpha(v) \notin X \cup \varphi_1(C_2)$, we have

$$F_1(x^t)_{\alpha(v)} = x^{t+1}_{\alpha(v)} = z^{t+1}_v.$$

We conclude that $z^{t+1}_v = F'(z^t)_v$. By a similar reasoning, we obtain the same conclusion if $\alpha(v) \in V_2$. We deduce that (z^t) is a $X \cup Y$-pseudo-orbit of F'. □

In the case of a CSAN family where the transition rules are determined by a labeled non-directed graph, the result of a glueing operation has no reason to belong to the family because the symmetry of the interaction graph might be broken (see Figure 3.2). The following lemma gives a sufficient condition in graph theoretical terms for glueing within a CSAN family. Intuitively, it consists in asking that, in each graph, all the connections of one half of the dowel to the rest of the graph goes through the other half of the dowel. Here the wooden dowel metaphor is particularly relevant: when considering a single piece of wood with the dowel inserted in it, one half of the dowel is "inside" (touches the piece of wood), the other half is "outside" (not touching the piece of wood); then, when the two pieces are attached, each position in the wood assembly is locally either like in one piece of wood with the dowel inserted in it or like in the other one with the dowel inserted in it.

Lemma 3.3 (Glueing for CSAN) *Let (G_1, λ_1, ρ_1) and (G_2, λ_2, ρ_2) be two CSAN from the same CSAN family $\mathcal{F}$ where G_1 and G_2 are disjoint and F_1 and F_2 are the associated global maps. Taking again the notations of Definition 3.1, if the following conditions hold:*

- *the labeled graphs induced by $\varphi_1(C)$ and $\varphi_2(C)$ in G_1 and G_2 are the same (using the identification $\varphi_1(v) = \varphi_2(v)$);*
- *$N_{G_1}(\varphi_1(C_2)) \subseteq \varphi_1(C)$; and*
- *$N_{G_2}(\varphi_2(C_1)) \subseteq \varphi_2(C)$;*

then the glueing $F' = F_1 \overset{\phi_1}{\underset{C_1}{\oplus}} \overset{\phi_2}{\underset{C_2}{}} F_2$ can be defined as the CSAN on graph $G' = (V', E')$ where V' is as in Definition 3.1 and each node $v \in V'$ has the same label and same labeled neighborhood as either a node of (G_1, λ_1, ρ_1) or a node of (G_2, λ_2, ρ_2). In particular F' belongs to $\mathcal{F}$.

Proof. Let us define $\beta_{\mathrm{i}} : V_{\mathrm{i}} \to V'$ by

$$\beta_{\mathrm{i}}(v) = \begin{cases} \phi_{\mathrm{i}}^{-1}(v) & \text{if } v \in \phi_{\mathrm{i}}(C), \\ v & \text{else.} \end{cases}$$

Fix $\mathrm{i} = 1$ or 2. According to Definition 3.1, if $v \in V'$ is such that $\alpha(v) \in V_{\mathrm{i}}$ then $F'_v = (F_{\mathrm{i}})_{\alpha(v)} \circ \rho_{\mathrm{i}}$. By definition of CSAN, this means that for any $x \in Q^{V'}$ we have $F'_v(x) = \psi_{\mathrm{i},\alpha(v)}(x_{|\beta(N_{G_{\mathrm{i}}}(\alpha(v)))})$ where $\psi_{\mathrm{i},\alpha(v)}$ is a map depending only on the labeled neighborhood of $\alpha(v)$ in G_{i} as in Definition 1.5. So the dependencies of v in F' are in one-to-one correspondence through β with the neighborhood of $\alpha(v)$ in G_{i}. The key observation is that the symmetry of dependencies is preserved, formally for any $v' \in \beta(N_{G_{\mathrm{i}}}(\alpha(v)))$:

- either $\alpha(v') \in V_{\mathrm{i}}$ in which case the dependency of v' on v (in map $\psi_{\mathrm{i},\alpha(v')}$) is the same as the dependency of v on v' (in map $\psi_{\mathrm{i},\alpha(v)}$), and both are determined by the undirected labeled edge $\{\alpha(v), \alpha(v')\}$ of G_{i};
- or $\alpha(v') \notin V_{\mathrm{i}}$ and in this case necessarily $v \in C_{\mathrm{i}}$ and $v' \in C_{3-\mathrm{i}}$ (because $N_{G_{\mathrm{i}}}(V_{\mathrm{i}} \setminus \varphi_{\mathrm{i}}(C)) \cap \varphi_{\mathrm{i}}(C) \subseteq \varphi_{\mathrm{i}}(C_{\mathrm{i}})$ from the hypothesis), so the dependency of v' on v is the same as the dependency of v on v' because the labeled graphs induced by $\phi_1(C)$ and $\phi_2(C)$ in G_1 and G_2 are the same.

Concretely, F' is a CSAN that can be defined on graph $G' = (V', E'_1 \cup E'_2 \cup E(C))$ with

$$E'_{\mathrm{i}} = E(V_{\mathrm{i}} \setminus \varphi_{\mathrm{i}}(C)) \cup \{(u, v_{\mathrm{i}}) : u \in V(C_{\mathrm{i}}), v_{\mathrm{i}} \in (V_{\mathrm{i}} \setminus \varphi_{\mathrm{i}}(C)), (\varphi_{\mathrm{i}}(u), v_{\mathrm{i}}) \in E_{\mathrm{i}}\},$$

and labels as follows:

- on $E(C)$ as in both G_1 and G_2 (which corresponds to the image of maps ϕ_1 and ϕ_2 on C);
- on $E(V_{\mathrm{i}} \setminus \varphi_{\mathrm{i}}(C))$ as in G_{i};
- for each $u \in V(C_{\mathrm{i}}), v_{\mathrm{i}} \in (V_{\mathrm{i}} \setminus \varphi_{\mathrm{i}}(C))$ such that $(\varphi_{\mathrm{i}}(u), v_{\mathrm{i}}) \in E_{\mathrm{i}}$, edge (u, v_{i}) has same label as $(\varphi_{\mathrm{i}}(u), v_{\mathrm{i}})$;

Since any CSAN families (Definition 1.5) is entirely based on local constraints on labels (vertex label plus set of labels of the incident edges), we deduce that F' is in $\mathcal{F}$.

□

3.2 Computing on automata networks

3.2.1 $\mathcal{G}$-networks

In this subsection, we formalize the idea of building large automata networks from small building blocks. By building blocks, we mean any finite maps with some number of inputs and some number of outputs that all share the same alphabet. It can be helpful to think about Boolean automata networks built from Boolean gates. However, in our formalism, fan in and fan out are not free (they are allowed if available as a building block). These details are important since neither information duplication, neither synchronization are granted in our context of embedding computations into dynamical systems.

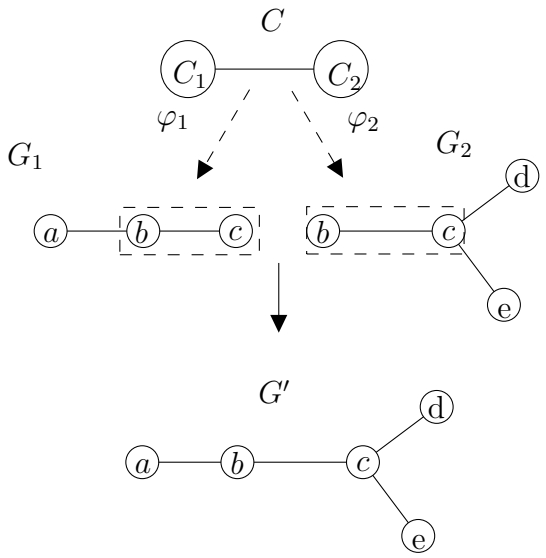


Figure 3.3: Example of a glueing of two compatibles CSAN. The labeling in nodes of G_1, G_2 and G' shows equalities between local λ maps of these three CSAN.

Let Q be a fixed alphabet and $\mathcal{G}$ be any set of maps of type $g : Q^{\mathrm{i}(g)} \to Q^{o(g)}$ for some $\mathrm{i}(g), o(g) \in \mathbb{N}$. We say that g is *reducible* if it can be written as a disjoint union of two gates, and *irreducible* otherwise. In other terms, if G is the (bipartite) dependency graph of g describing on which inputs effectively depends each output, then g is irreducible if G is weakly connected.

Intuitively, a $\mathcal{G}$-network is an automata network obtained by wiring outputs to inputs of a number of gates from $\mathcal{G}$. Generally speaking, each gate has some input and output wires. Each of these wires is equipped with some internal state and each output wire computes its next state according to the local function associated to its gate. In addition, we assume that the total amount of output and input wires is the same, so each output wire is also the input wire of some gate. To simplify some later results, we add the technical condition that no output of a gate can be wired to one of its inputs (no self-loop condition).

Definition 3.4 *A* $\mathcal{G}$-network *is an automata network* $F : Q^V \to Q^V$ *with set of nodes* V *associated to a collection of gates* $g_1, \ldots, g_n \in \mathcal{G}$ *with the following properties. Let*

$$I = \{(j,k) : 1 \leq j \leq n \text{ and } 1 \leq k \leq \mathrm{i}(g_j)\} \text{ and}$$
$$O = \{(j,k) : 1 \leq j \leq n \text{ and } 1 \leq k \leq o(g_j)\}$$

be respectively the sets of inputs and outputs of the collection of gates $(g_j)_{1 \leq j \leq n}$*. We require* $|V| = |I| = |O|$ *and the existence of two bijective maps* $\alpha : I \to V$ *and* $\beta : V \to O$ *with the condition that there is no* $(j,k) \in I$ *such that* $\beta(\alpha(j,k)) = (j,k')$ *for some* k' *(no self-loop condition). For* $v \in V$ *with* $\beta(v) = (j,k)$*, let* $I_v = \{\alpha(j,1), \ldots, \alpha(j, \mathrm{i}(g_j))\}$ *and denote by* g_v *the map:* $x \in Q^{I_v} \mapsto g_j(\tilde{x})_k$ *where* $\tilde{x} \in Q^{\mathrm{i}(g_j)}$ *is defined by* $\tilde{x}_k = x_{\alpha(j,k)}$*. Then* F *is defined as follows:*

$$F(x)_v = g_v(x_{I_v}).$$

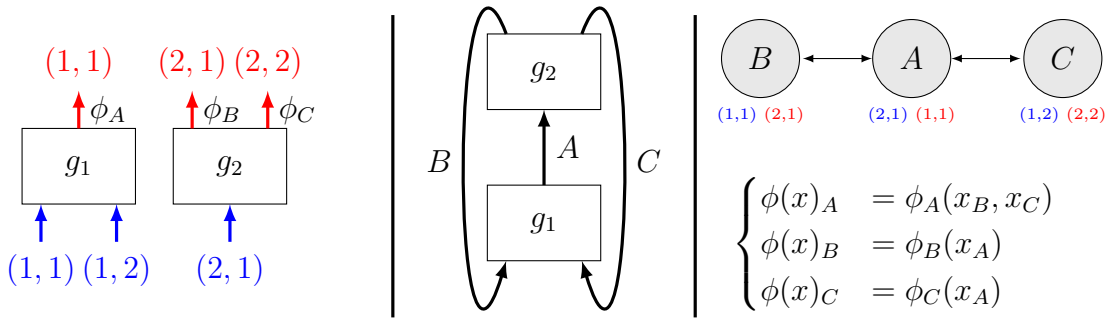


Figure 3.4: (Left panel) A set of maps $\mathcal{G}$ over alphabet Q. (Central panel) An intuitive representation of input/output connections to make a $\mathcal{G}$-network. (Right panel) The corresponding formal $\mathcal{G}$-network $\phi : Q^3 \to Q^3$ together with the global map associated to it. The bijections α and β from Definition 3.4 are represented in blue and red (respectively).

Remark 3.5 Once $\mathcal{G}$ is fixed, there is a bound on the degree of dependency graphs of all $\mathcal{G}$-networks. Thus, it is convenient to represent $\mathcal{G}$-networks by the standard representation of bounded degree automata networks (as a pair of a graph and a list of local update maps). Another representation choice following strictly Definition 3.4 consists in giving a list of gates $g_1, \dots, g_k \in \mathcal{G}$, fixing $V = \{1, \dots, n\}$ and give the two bijective maps $\alpha : I \to V$ and $\beta : V \to O$ describing the connections between gates (maps are given as a simple list of pairs source/image). One can check that these two representations are **DLOGSPACE** equivalent when the gates of $\mathcal{G}$ are irreducible: we can construct the interaction graph and the local maps from the list of gates and maps α and β in **DLOGSPACE** (the incoming neighborhood of a node v, I_v, and its local map g_v are easy to compute as detailed in Definition 3.4); reciprocally, given the interaction graph G and the list of local maps (g_v), one can recover in **DLOGSPACE** the list of gates and their connections as follows:

- for v from 1 to n do:
  - gather the (finite) incoming neighborhood $N^-(v)$ of v then the (finite) outgoing neighborhood $N^+(N^-(v))$ and iterate this process until it converges (in finite time) to a set I_v of inputs and O_v of outputs with $v \in O_v$;
  - check that all $v' \in O_v$ are such that $v' \geq v$ otherwise jump to next v in the loop (this guaranties that each gate is generated only once);
  - since the considered gates are irreducible, I_v and O_v actually correspond to input and output sets of a gate $g \in \mathcal{G}$ that we can recover by finite checks from the local maps of nodes in O_v;
  - output gate g and the pairs source/image to describe α and β for nodes in I_v and O_v respectively.

For example, in Figure 3.4, right panel, the latter algorithm executes the following instruc-

tions:

- **For** $v = A$:
  - Compute: $N^-(\{A\}) = \{B, C\} \mapsto N^+(\{B, C\}) = \{A\} \mapsto N^+(\{A\}) = \{B, C\}$.
  - Define $I_A = \{B, C\}$ and $O_A = \{A\}$.
  - Check that for all $v' \in O_A$ we have $v' \geq v$.
  - Define g_1 with two inputs labeled as $\{B, C\}$ and one output labeled by $\{A\}$ where $g_1 \equiv \phi_A$.
- **For** $v = B$:
  - Compute: $N^-(\{B\}) = \{A\} \mapsto N^+(\{A\}) = \{B, C\} \mapsto N^+(\{B, C\}) = \{A\}$.
  - Define $I_B = \{A\}$ and $O_B = \{B, C\}$.
  - Check that for all $v' \in O_A$ we have $v' \geq v$.
  - Define g_2 with one input labeled as $\{A\}$ and two outputs labeled by $\{B, C\}$ where $g_2 \equiv \phi_B$.
- **For** $v = C$:
  - Compute: $N^-(\{C\}) = \{A\} \mapsto N^+(\{A\}) = \{B, C\} \mapsto N^+(\{B, C\}) = \{A\}$.
  - Define $I_C = \{A\}$ and $O_C = \{B, C\}$.
  - Check that for all $v' \in O_A$ we have $v' \geq v$. $\mapsto$ **SKIP** because B was visited before.
- **Return** (g_1, g_2) and its labels.

In the sequel we denote $\Gamma(\mathcal{G})$ the family of all posible $\mathcal{G}$-networks associated to their bounded degree representation.

3.2.2 $\mathcal{G}$-gadgets and simulation of $\mathcal{G}$-networks

Now we give a precise meaning to the intuitively simple fact that, if a family of automata networks can coherently simulate a set of small building blocks (gates from $\mathcal{G}$), it should be able to simulate any automata network that can be built out of them ($\mathcal{G}$-networks).

The key idea here is that gates from $\mathcal{G}$ will be represented by networks of the family called $\mathcal{G}$-gadgets, and the wiring between gates to obtain a $\mathcal{G}$-network will translate into glueing between $\mathcal{G}$-gadgets. Following this idea there are two main conditions for the family to simulate any $\mathcal{G}$-network:

- The glueing of gadgets should be freely composable inside the family to allow the building of any $\mathcal{G}$-network.
- The gadgets corresponding to gates from $\mathcal{G}$ should correctly and coherently simulate the functional relation between inputs and outputs given by their corresponding gate.

For clarity, we separate these conditions in two definitions. Concerning glueing, as we want to mimic the wiring of gates which connects inputs to outputs, several copies of particular subgraphs will be identified in each gadget, some of them corresponding to input, and the other ones to outputs. Then, the only glueing operations we will use are those where some output of a gadget A are glued on input dowels of a gadget B and some inputs of A are glued on output of B.

In order to accomplish that, we introduce in the following section a general framework on glueing gadgets.

3.2.3 Gadget glueing

Now we focus in developing a definition for gadget glueing. Recall first that Definition 3.1 relies on the identification of a common dowel in the two networks to be glued. Here, as we want to mimic the wiring of gates which connects inputs to outputs, several copies of an interface dowel will be identified in each gadget, some of them corresponding to input, and the other ones to outputs. In this context, the only glueing operations we will use are those where some output dowels of a gadget A are glued on input dowels of a gadget B and some input dowels of A are glued on output dowels of B. Then, the global dowel used to formally apply Definition 3.1 is a disjoint union of the selected input/output interface dowels (see Figure 3.5).

Definition 3.6 (Glueing interface and gadgets) *Let $C = C_{\mathrm{i}} \cup C_o$ be a fixed set partitioned into two sets. A* gadget *with glueing interface $C = C_{\mathrm{i}} \cup C_o$ is an automata network $F : Q^{V_F} \to Q^{V_F}$ together with two collections of injective maps $\phi^{\mathrm{i}}_{F,k} : C \to V_F$ for $k \in I(F)$ and $\phi^{o}_{F,k} : C \to V_F$ for $k \in O(F)$ whose images in V_F are pairwise disjoint and where $I(F)$ and $O(F)$ are disjoint sets which might be empty.*

Given two disjoint gadgets $(F, (\phi^{\mathrm{i}}_{F,k}), (\phi^{o}_{F,k}))$ and $(G, (\phi^{\mathrm{i}}_{G,k}), (\phi^{o}_{G,k}))$ with same alphabet and interface $C = C_{\mathrm{i}} \cup C_o$, a gadget glueing *is a glueing of the form $H = F \overset{\phi_F}{C_F} \oplus \overset{\phi_G}{C_G} G$ defined as follows:*

- *a choice of a set A of inputs from F and outputs from G given by injective maps $\sigma_F : A \to I(F)$ and $\sigma_G : A \to O(G)$;*
- *a choice of a set B of outputs from F and inputs from G given by injective maps $\tau_F : B \to O(F)$ and $\tau_G : B \to I(G)$ (the set B is disjoint from A);*
- *C_F is a disjoint union of $|A|$ copies of C_{i}, and $|B|$ copies of C_o: $C_F = A \times C_{\mathrm{i}} \cup B \times C_o$;*
- *C_G is a disjoint union of $|A|$ copies of C_o, and $|B|$ copies of C_{i}: $C_G = A \times C_o \cup B \times C_{\mathrm{i}}$;*
- *$\phi_F : C_F \cup C_G \to V_F$ is such that $\phi_F(a, c) = \phi^{\mathrm{i}}_{F,\sigma_F(a)}(c)$ for $a \in A$ and $c \in C$, and $\phi_F(b, c) = \phi^{o}_{F,\tau_F(b)}(c)$ for $b \in B$ and $c \in C$; and*
- *$\phi_G : C_F \cup C_G \to V_G$ is such that $\phi_G(a, c) = \phi^{o}_{G,\sigma_G(a)}(c)$ for $a \in A$ and $c \in C$, and $\phi_G(b, c) = \phi^{\mathrm{i}}_{G,\tau_G(b)}(c)$ for $b \in B$ and $c \in C$.*

The resulting network H is a gadget with same alphabet and same interface with $I(H) = I(F) \setminus \sigma_F(A) \cup I(G) \setminus \tau_G(B)$ and $O(H) = O(F) \setminus \tau_F(B) \cup O(F) \setminus \sigma_G(A)$ and $\phi^{\mathrm{i}}_{H,k}$ is $\phi^{\mathrm{i}}_{F,k}$ when $k \in I(F)$ and $\phi^{\mathrm{i}}_{G,k}$ when $k \in I(G)$, and $\phi^{o}_{H,k}$ is $\phi^{o}_{F,k}$ when $k \in O(F)$ and $\phi^{o}_{G,k}$ when $k \in O(G)$.

Given a set of gadgets X with same alphabet and interface, its closure by gadget glueing is the closure of X by the following operations:

- *add a disjoint copy of some gadget from the current set;*
- *add the disjoint union of two gadgets from the current set; and*

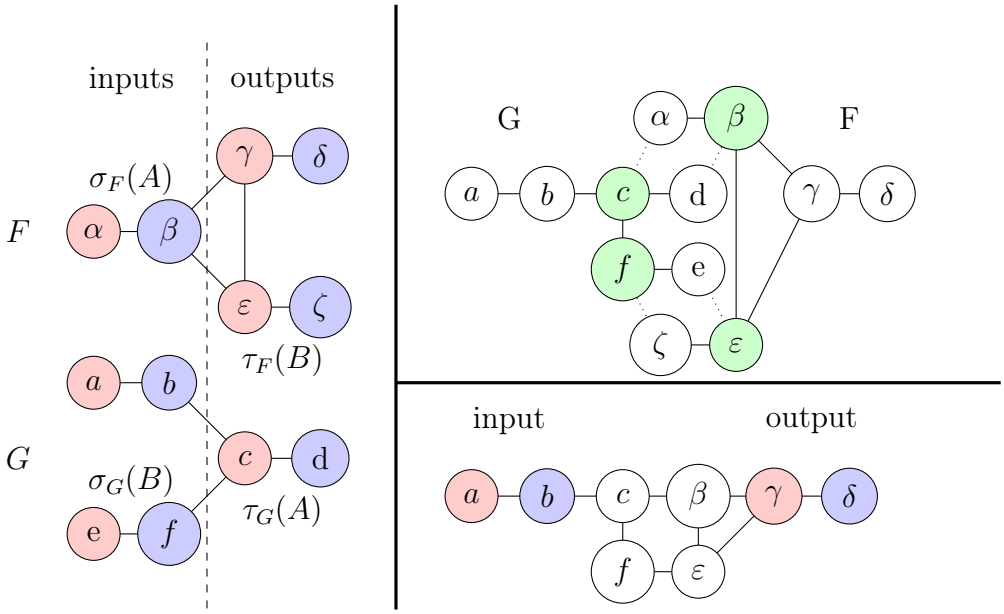


Figure 3.5: Gadget glueing as in Definition 3.6. (Left panel) Two gadgets with interface $C = C_{\mathrm{i}} \cup C_o$ where C_{i} part in each copy of the interface dowel is in red and C_o part in blue. The gadget glueing is done with input $\sigma_F(A)$ on output $\sigma_G(A)$ (here A is a singleton) and output $\tau_F(B)$ on input $\tau_G(B)$ (B is also a singleton). (Top right panel) A representation of the global glueing process where nodes in green are those in the copy of C_F in F or in the copy C_G in G; dotted links show the bijection between the embeddings of $C = C_F \cup C_G$ into V_F and V_G via maps ϕ_F and ϕ_G. (Bottom right panel) The resulting gadget with the same interface $C = C_{\mathrm{i}} \cup C_o$ as the two initial gadgets.

- *add a gadget glueing of two gadgets from the current set.*

Remark 3.7 The representation of the result of a gadget glueing can be easily computed from the two gadgets F and G and the choices of inputs/outputs given by maps σ_F, σ_G, τ_F and τ_G: precisely, by definition of glueing (Definition 3.1) the local map of each node of the result automata network is either a local map of F (when in $V_F \setminus \phi_F(C_G)$ or in C_F) or a local map of G (when in $V_G \setminus \phi_G(C_F)$ or in C_G). Note also that the closure by gadget glueing of a finite set of gadgets X is always a set of automata networks of bounded degree.

Lemma 3.3 gives sufficient conditions on a set of gadgets to have its closure by gadget glueing contained in a CSAN family.

Lemma 3.8 *Fix some alphabet Q and some glueing interface $C = C_{\mathrm{i}} \cup C_o$ and some CSAN family $\mathcal{F}$. Let (G_n, λ_n, ρ_n) for $n \in S$ be a set of CSAN belonging to $\mathcal{F}$ with associated global maps F_n. Let $\phi^{\mathrm{i}}_{F_n,k}$ for $k \in I(F_n)$ and $\phi^{o}_{F_n,k}$ for $k \in O(F_n)$ be maps as in Definition 3.6 so that $(F_n, (\phi^{\mathrm{i}}_{F_n,k}), (\phi^{o}_{F_n,k}))$ is a gadget with interface $C = C_{\mathrm{i}} \cup C_o$. Denote by X the set of such gadgets. If the following conditions hold:*

- *the labeled graphs induced by $\phi^{\mathrm{i}}_{F_n,k}(C)$ and by $\phi^{o}_{F_n,k}(C)$ in G_n are all the same for all n*

and k with the identification of vertices given by the $\phi^*_{*,*}$ maps;

- *$N_{G_n}(\phi^{\mathrm{i}}_{F_n,k}(C_o)) \subseteq \phi^{\mathrm{i}}_{F_n,k}(C)$ for all $n \in S$ and all $k \in I(F_n)$; and*
- *$N_{G_n}(\phi^{o}_{F_n,k}(C_{\mathrm{i}})) \subseteq \phi^{o}_{F_n,k}(C)$ for all $n \in S$ and all $k \in O(F_n)$;*

then the closure by gadget glueing of X is included in $\mathcal{F}$.

Proof. Consider first the gadget glueing H of two gadgets F_n and $F_{n'}$ from X. Following Definition 3.6, the global dowel $C_{F_n} \cup C_{F_{n'}}$ used in such a glueing is a disjoint union of copies of C, and its embedding ϕ_{F_n} in G_n (resp. $\phi_{F_{n'}}$ in $G_{n'}$) is a disjoint union of maps $\phi^*_{F_n,*}$ (resp. $\phi^*_{F_{n'},*}$). Therefore the three conditions of Lemma 3.3 follow from the three conditions of the hypothesis on gadgets from X and we deduce that H belongs to family $\mathcal{F}$. Moreover, it is clear that gadget H then also verifies the three conditions from the hypothesis, and adding a copy of any gadget to the set also verifies the conditions. We deduce that the closure by gadget glueing of X is included in $\mathcal{F}$.

□

The second key aspect to have a coherent set X of $\mathcal{G}$-gadgets is of dynamical nature: there must exist a collection of pseudo-orbits on each gadget satisfying suitable conditions to permit application of Lemma 3.2 for any gadget glueing in the closure of X; moreover, these pseudo-orbits must simulate via an appropriate coding the input/output relations of each gate $g \in \mathcal{G}$ in the corresponding gadget. To obtain this, we rely on a standard set of traces on the glueing interface that must be respected on any copy of it in any gadget. Generally speaking, the interface dowels play a similar role that wires in the case of the construction of $\mathcal{G}$-networks.

Definition 3.9 (Coherent $\mathcal{G}$-gadgets) *Let $\mathcal{G}$ be any set of finite maps over alphabet Q and let $\mathcal{F}$ be any set of abstract automata networks over alphabet $Q_{\mathcal{F}}$. We say $\mathcal{F}$ has* coherent $\mathcal{G}$-gadgets *if there exist:*

- *a unique glueing interface $C = C_{\mathrm{i}} \cup C_o$;*
- *a set X of gadgets $(F_g, (\phi^{\mathrm{i}}_{g,k})_{1\leq k\leq \mathrm{i}(g)}, (\phi^{o}_{g,k})_{1\leq k\leq o(g)})$ for each $g \in \mathcal{G}$ where $F_g : Q_{\mathcal{F}}^{V_g} \to Q_{\mathcal{F}}^{V_g} \in \mathcal{F}$ and sets V_g and C are pairwise disjoint, and the closure of X by gadget glueing is contained in $\mathcal{F}$,*
- state configurations $s_q \in Q_{\mathcal{F}}^{C}$ *for each $q \in Q$ such that $q \mapsto s_q$ is an injective map;*
- context configurations $c_g \in Q_{\mathcal{F}}^{\hat{V}_g}$ *for each $g \in \mathcal{G}$ where $\hat{V}_g = V_g \setminus \big(\cup_k \phi^{\mathrm{i}}_{g,k}(C) \cup_k \phi^{o}_{g,k}(C)\big)$;*
- *a time constant T,*
- *a* standard trace $\tau_{q,q'} \in (Q_{\mathcal{F}}^{C})^{\{0,\dots,T\}}$ *for each pair $q, q' \in Q$ such that $\tau_{q,q'}(0) = s_q$ and $\tau_{q,q'}(T) = s_{q'}$; and*
- *for each $g \in \mathcal{G}$ and for any uples of states $q_{\mathrm{i},1}, \dots, q_{\mathrm{i},\mathrm{i}(g)} \in Q$ and $q_{o,1}, \dots, q_{o,o(g)} \in Q$ and $q'_{\mathrm{i},1}, \dots, q'_{\mathrm{i},\mathrm{i}(g)} \in Q$ and $q'_{o,1}, \dots, q'_{o,o(g)} \in Q$ such that $g(q_{\mathrm{i},1}, \dots, q_{\mathrm{i},\mathrm{i}(g)}) = (q'_{o,1}, \dots, q'_{o,o(g)})$, a P_g-pseudo-orbit $(x^t)_{0\leq t\leq T}$ of F_g with $P_g = \bigcup_{1\leq k\leq \mathrm{i}(g)} \phi^{\mathrm{i}}_{g,k}(C_o) \cup \bigcup_{1\leq k\leq o(g)} \phi^{o}_{g,k}(C_{\mathrm{i}})$ and with:*
  - *for each $1 \leq k \leq \mathrm{i}(g)$, the trace $t \mapsto x^t_{\phi^{\mathrm{i}}_{g,k}(C)}$ is exactly $\tau_{q_{\mathrm{i},k}, q'_{\mathrm{i},k}}$;*

– *for each* $1 \leq k \leq o(g)$, *the trace* $t \mapsto x^t_{\phi^o_{g,k}(C)}$ *is exactly* $\tau_{q_{o,k},q'_{o,k}}$; *and*

– $x^0_{\hat{V}_g} = x^T_{\hat{V}_g} = c_g$.

Lemma 3.10 *Let* $\mathcal{G}$ *be a set of irreducible gates. If an abstract automata network family* $\mathcal{F}$ *has coherent* $\mathcal{G}$*-gadgets then it contains a subfamily of bounded degree networks with the canonical bounded degree representation* $(\mathcal{F}_0, \mathcal{F}_0^*)$ *that simulates* $\Gamma(\mathcal{G})$ *in time* T *and space* S *where* T *is a constant map and* S *is bounded by a linear map.*

Proof. We take the notations of Definition 3.9. To any $\mathcal{G}$-network F with set of nodes V given as in Definition 3.4 by a list of gates $g_1, \ldots, g_k \in \mathcal{G}$ and maps α and β (see Remark 3.5) we associate an automata network from $\mathcal{F}$ as follows. First, let $(F_{g_\mathrm{i}})_{1\leq \mathrm{i}\leq k}$ be the gadgets corresponding to gates g_i and suppose they are all disjoint (by taking disjoint copies when necessary). Then, start from the gadget $F_1 = F_{g_1}$. For any $1 \leq \mathrm{i} < k$ we define $F_{\mathrm{i}+1}$ as the gadget glueing of F_i and $F_{g_{\mathrm{i}+1}}$ on the input/outputs as prescribed by maps α and β. More precisely, the gadget glueing selects the set of inputs (j,k) with $1 \leq j \leq \mathrm{i}$ and $1 \leq k \leq \mathrm{i}(g_j)$ such that $\beta(\alpha(j,k)) = (\mathrm{i}+1, k')$ for some $1 \leq k' \leq o(g_{\mathrm{i}+1})$ and glue them on their corresponding output $(\mathrm{i}+1, k')$ of $g_{\mathrm{i}+1}$ (precisely, through maps σ_{F_i} and $\sigma_{F_{g_{\mathrm{i}+1}}}$ of domain $A_{\mathrm{i}+1}$ playing the role of maps σ_F and σ_G of Definition 3.6). Symmetrically, selects the inputs $(\mathrm{i}+1, k)$ with $1 \leq k \leq \mathrm{i}(g_{\mathrm{i}+1})$ such that $\beta(\alpha(\mathrm{i}+1, k)) = (j, k')$ for some $1 \leq j \leq \mathrm{i}$ and $1 \leq k' \leq o(g_j)$ and glue their corresponding output (j, k') (precisely, through maps $\tau_{F_{g_{\mathrm{i}+1}}}$ and τ_{F_i} of domain $B_{\mathrm{i}+1}$ playing the role of maps τ_G and τ_F from Definition 3.6). If both of these sets of inputs/outputs are empty, the gadget glueing is replaced by a simple disjoint union.

The final gadget F_k has no input and no output, and a representation of it as a pair graph and local maps can be constructed in **DLOGSPACE**. Indeed, the local map of each of its nodes is independent of the glueing sequence above and completely determined by the gadget F_{g_j} it belongs to and whether the node is inside some input or some output dowel or not (see Reamrk 3.7).

It now remains to show that the automata network F_k simulates F. To fix notations, let V_k be the set of nodes of F_k. For each $v \in V$, define $D_v \subseteq V_k$ as the copy of the dowel that corresponds to node v of F, *i.e.* that was produced in the gadget glueing of F_i with $F_{g_{\mathrm{i}+1}}$ for i such that $\beta(v) = (\mathrm{i}+1, k')$ for some $1 \leq k' \leq o(g_{\mathrm{i}+1})$ (or symmetrically $\alpha(\mathrm{i}+1, k) = v$ for some $1 \leq k \leq \mathrm{i}(g_j)$). More precisely, if $a \in A_{\mathrm{i}+1}$ is such that $\sigma_{F_{g_{\mathrm{i}+1}}}(a) = (\mathrm{i}+1, k')$ then $D_v = \{a\} \times C$ (symmetrically if $b \in B_{\mathrm{i}+1}$ is such that $\tau_{F_{g_{\mathrm{i}+1}}}(b) = (\mathrm{i}+1, k)$ then $D_v = \{b\} \times C$). Also denote by $\rho_v : D_v \to C$ the map such that $\rho_v(a, c) = c$ for all $c \in C$ (symmetrically, $\rho_v(b, c) = c$). With these notations, we have

$$V_k = \bigcup_{v \in V} D_v \cup \bigcup_{1 \leq \mathrm{i} \leq k} \hat{V}_{g_\mathrm{i}}.$$

Let us define the block embedding $\phi : Q^V \to Q_{\mathcal{F}}^{V_k}$ as follows:

$$\phi(x)(v') = \begin{cases} s_{x_v}(\rho_v(v')) & \text{if } v' \in D_v, \\ c_{g_\mathrm{i}}(v') & \text{if } v' \in \hat{V}_{g_\mathrm{i}}, \end{cases}$$

for any $x \in Q^V$ and any $v' \in V_k$, where s_q for $q \in Q$ are the state configurations and c_g for $g \in \mathcal{G}$ are the context configurations granted by Definition 3.9. Note that ϕ is injective because the map $q \mapsto s_q$ is injective. By inductive applications of Lemma 3.2, the P_{g_i}-pseudo-orbits of each F_{g_i} from Definition 3.9 can be glued together to form valid orbits of F_k that start from any configuration $\phi(x)$ with $x \in Q^V$ and end after T steps in a configuration $\phi(y)$ for some $y \in Q^V$ which verifies $y = F(x)$. In other words, we have the following equality on Q^V:

$$\phi \circ F = F_k^T \circ \phi.$$

Note that T is a constant and that the size of V_k is at most linear in the size of V.

□

Remark 3.11 Note that in Lemma 3.10 above, the block embedding that is constructed can be viewed as a collection of blocks of bounded size that encode all the information plus a context (see Remark 2.4).

In the case of CSAN families and using Remark 2.1 we have a simpler formulation of the Lemma.

Corollary 3.12 *If $\mathcal{G}$ in a set of irreducible gates and $\mathcal{F}$ a CSAN family which has coherent $\mathcal{G}$-gadgets then $\mathcal{F}$ simulates $\Gamma(\mathcal{G})$ in time T and space S where T is a constant map and S is bounded by a linear map.*

3.3 Some useful families of $\mathcal{G}$-networks: $\mathcal{G}_m$-networks and $\mathcal{G}_{m,2}$-networks

It is folklore knowledge that monotonic Boolean networks (with AND/OR local maps) can simulate any other network. Let $\mathrm{i}, o \in \{1, 2\}$ be two numbers. We define the functions $\mathrm{OR}^{\mathrm{i},o}, \mathrm{AND}^{\mathrm{i},o} : \{0,1\}^{\mathrm{i}} \to \{0,1\}^o$ where $\mathrm{OR}^{\mathrm{i},o}(x)_k = \max(x)$ and $\mathrm{AND}^{\mathrm{i},o}(x)_k = \min(x)$ for $1 \leq k \leq o$ and $x \in \{0,1\}^{\mathrm{i}}$. Observe that in the case in which $\mathrm{i} = o = 1$ we have $\mathrm{AND}(x) = \mathrm{OR}(x) = \mathrm{Id}(x) = x$ and also in the case $\mathrm{i} = 1$ and $o = 2$ we have that $\mathrm{AND}(x) = \mathrm{OR}(x) = (x, x)$. We call that latter gate a *copy* gate an we denote it as COPY. We define the set $\mathcal{G}_m = \{\mathrm{AND}^{\mathrm{i},o}, \mathrm{OR}^{\mathrm{i},o}\}_{\mathrm{i},o \in \{1,2\}}$. From now on, and when the context is clear, we will note $\mathrm{AND}^{2,1}$ and $\mathrm{OR}^{2,1}$ as AND and OR. In addition, we define the set $\mathcal{G}_{m,2}$ in which we fix $\mathrm{i} = o = 2$. Here we make the latter statement regarding the simulation capabilities of monotonic Boolean networks precise, within our formalism: $\mathcal{G}_m$-networks are universal. Note that there is more work than the classical circuit transformations involving monotonic gates [29] because we need to obtain a simulation of any automata network via block embedding. In particular we need to build monotonic circuitry that is synchronized and reusable (*i.e.* that can be reinitialized to a standard state before starting a computation on a new input). Moreover, our definitions require a production of $\mathcal{G}_m$-networks in **DLOGSPACE**. The main ingredient of the proof of the following theorem is an efficient circuit transformation due to Greenlaw, Hoover and Ruzzo in [45, Theorems 6.2.3 to 6.2.5]

Theorem 3.13 *The family $\Gamma(\mathcal{G}_m)$ of all $\mathcal{G}_m$-networks is strongly universal.*

Proof. Let Q an arbitrary alphabet and $F : Q^n \to Q^n$ an arbitrary automata network on alphabet Q such that the communication graph of F has maximum degree Δ. Let $C : \{0,1\}^n \to \{0,1\}^n$ be a constant depth circuit representing F. Let us assume that C has only OR, AND and NOT gates. We can also assume that C is synchronous because, as its depth does not depend on the size of the circuit, one can always add fanin one and fanout one OR gates in order to modify layer structure. We are going to use a very similar transformation to the one proposed in [45, Theorems 6.2.3] in order to efficiently construct an automata network in $\Gamma(\mathcal{G}_m)$. In fact, we are going to duplicate the original circuit by considering the coding $x \in \{0,1\} \to (x, 1-x) \in \{0,1\}^2$. Roughly speaking, each gate will have a positive part (which is essentially a copy) and a negative part which produces the negation of the original output by using De Morgan's laws. More precisely, we are going to replace each gate in the network by the gadgets shown in Figure 3.6. The main idea is that one can represent the function $x \wedge y$ by the coding: $(x \wedge y, \overline{x} \vee \overline{y})$ and $x \vee y$ by the coding: $(x \vee y, \overline{x} \wedge \overline{y})$. In addition, each time there is a NOT gate, we replace it by a fanin 1 fanout 1 OR gadget and we connect positive outputs to negative inputs in the next layer and negative outputs to positive inputs as it is shown in Figure 3.7. We call C^* the circuit built by the latter transformations. Observe that C^* is such that it holds on $\{0,1\}^{\mathrm{i}}$:

$$\phi \circ C = C^* \circ \phi$$

where $\phi : \{0,1\}^n \to \{0,1\}^{2n}$ is defined for any n by $\phi(x)_{2j} = x_j$ and $\phi(x)_{2j+1} = \neg x_j$.

Now consider the coding map $m_Q : Q \to \{0,1\}^k$ and let $n = k|V|$. Build from C^* the $\mathcal{G}_m$-network $F^* : \{0,1\}^{V^+} \to \{0,1\}^{V^+}$ that corresponds to it (gate by gate) and where the output j is wired to input j for all $1 \leq j \leq 2n$. Define a block embedding of Q^V into $\{0,1\}^{V^+}$ as follows (see Remark 2.4):

- for each $v \in V$ let D_v be the set of input nodes in F^* that code v (via m_Q and then double railed logic);
- let $C = V^+ \setminus \bigcup_v D_v$ be the remaining context block;
- let $p_{v,q} \in \{0,1\}^{D_v}$ be the pattern coding node v in state q;
- let $p_C = 0^C$ be the context pattern; and
- let $\phi : Q^V \to \{0,1\}^{V^+}$ be the associated block embedding map.

We claim that F^* simulates F via block embedding ϕ with time constant equal to the depth of C^* plus 1. Indeed, F^* can be seen as a directed cycle of N layers where layer $L_{\mathrm{i}+1 \bmod N}$ only depends on layer i. The block embedding is such that for any configuration $x \in Q^V$, $\phi(x)$ is 0 on each layer except the layer containing the inputs. On configurations where a single layer L_{i} is non-zero, F^* will produce a configuration where the only non-zero layer is $L_{\mathrm{i}+1 \bmod N}$. From there, it follows by construction of F^* that $\phi \circ F(x) = (F^*)^N \circ \phi(x)$ for all $x \in Q^V$.

The fact that the construction can be obtained in **DLOGSPACE** follows from the same reasoning used to show in [45, Theorem 6.2.3]. In fact, the authors show that the reduction is actually better since they show it is $\mathbf{NC}^1$. □

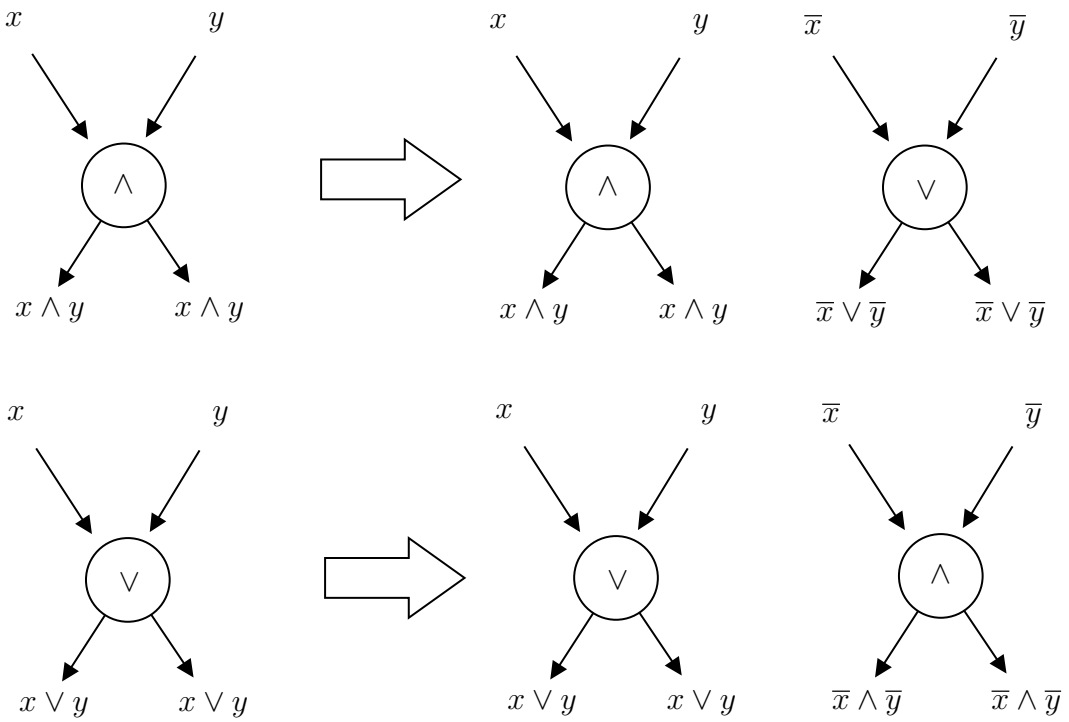


Figure 3.6: AND and OR gadgets for simulating AND/OR gates with fanin and fanout 2. For other values of fanin and fanout gadgets are the same but considering different number of inputs/outputs.

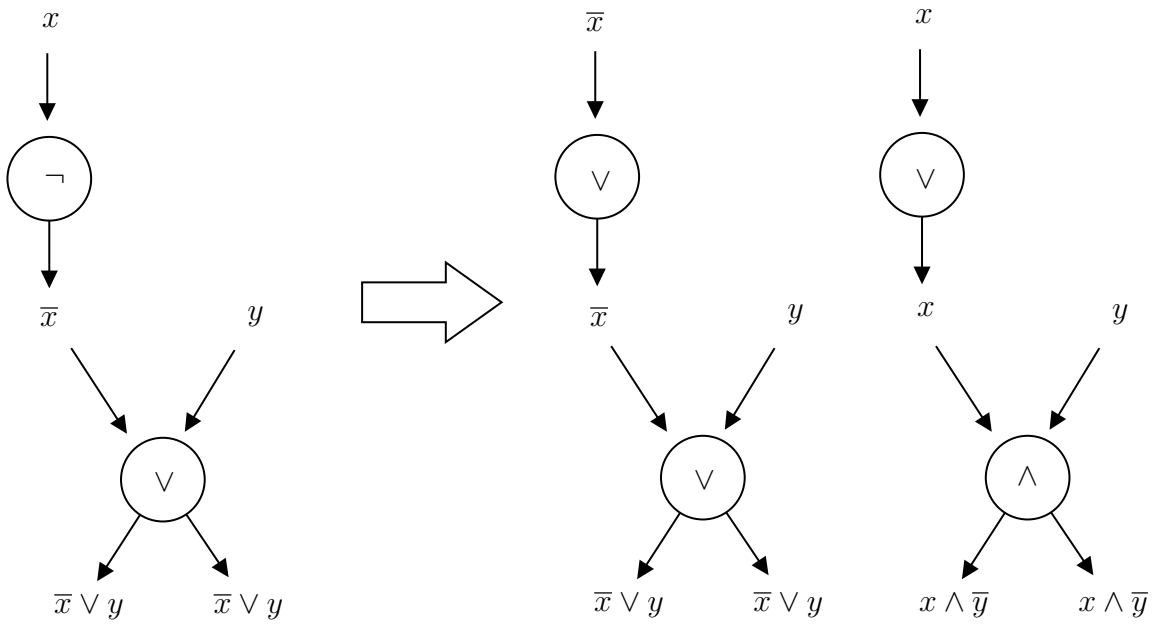


Figure 3.7: NOT gadget wiring for circuit simulation using gates from $\mathcal{G}_m$. In this case a NOT gate is connected to an OR gate in the original circuit. Copies of the NOT gate in the circuit performing simulation are connected to the copies of the OR gate switched: positive part is connected to negative part of the OR gate and viceversa.

Theorem 3.14 *The family $\Gamma(\mathcal{G}_{m,2})$ simulates in constant time and linear space the family $\Gamma(\mathcal{G}_m)$, i.e. there exist a constant function $T : \mathbb{N} \to \mathbb{N}$ and a linear function $S : \mathbb{N} \to \mathbb{N}$ such that $\Gamma(\mathcal{G}_m) \preccurlyeq_S^T \Gamma(\mathcal{G}_{m,2})$*

Proof. Let $F : Q^V \to Q^V$ be an arbitrary $\mathcal{G}_m$-network coded by its standard representation defined by a list of gates $g_1, \ldots, g_n$ and two functions α an β mapping inputs to nodes in F and nodes in F to outputs respectively. We are going to construct in **DLOGSPACE** a $\mathcal{G}_{m,2}$-network F^* that simulates F in time $T = \mathcal{O}(1)$ and space $S = \mathcal{O}(|V|)$. To this end, we are going to replace each gate g_k by a small gadget. More precisely, we are going to introduce the following coding function: $x \in \{0,1\} \to (x, x, 0) \in \{0,1\}^3$. We are going to define gadgets for each gate. Let us take $k \in \{1, \ldots, n\}$ and call g_k^* the corresponding gadget associated to g_k. Let us that suppose g_k is an OR gate and that it has fanin 2 and fanout 1 then, we define $g^* : \{0,1\}^6 \to \{0,1\}^6$ as a function that for each input of the form $(x, x, y, y, 0, 0)$ produces the output $g^*((x, x, y, y, 0, 0)) = (x \vee y, x \vee y, 0, 0, 0, 0)$. The case fanin 1 and fanout 1 is given by $g^*((x, x, 0, 0, 0, 0)) = (x, x, 0, 0, 0, 0)$, the case fanin 2 and fanout 2 is given by $g^*((x, x, y, y, 0, 0)) = (x \vee y, x \vee y, x \vee y, x \vee y, 0, 0)$ and finally case fanin 1 and fanout 2 is given by the same latter function but on input $(x, x, 0, 0, 0, 0)$. The AND case is completely analogous. We are going to implement the previous functions as small (constant depth) synchronized circuits that we call block gadgets. More precisely, we are going to identify functions g^* with its correspondent block gadget. The detail on the construction of these circuits that define latter functions are provided in Figures 3.8, 3.9 and 3.10.

Once we have defined the structure of block gadgets, we have to manage connections between them and also manage the fixed 0 inputs that we have added in addition to the zeros that are produced by the blocks as outputs. In order to do that, let us assume that gates g_{i} and g_j are connected. Note from the discussion on coding above that AND/OR gadgets have between 2 and 4 inputs and outputs fixed to 0. In particular, as it is shown in Figures 3.8, 3.9 and 3.10, all the block gadgets have the same amount of zeros in the input and in the output with the exception of the fanin 1 fanout 2 gates and the fanin 2 fanout 1 gates. However, since $\mathcal{G}$-networks are closed systems (the amount of inputs must be the same that the amount of outputs) we have that, for each fanin 1 fanout 2 gate, there must be a fanin 2 fanout 1 gate and vice versa (otherwise there would be more input than outputs or more outputs than inputs). In other words, there is a bijection between the set of fanin 1 fanout 2 gates and the set of fanin 2 fanout 1. Observe that fanin 2 fanout 1 gates consume 2 zeros in input but produce 4 zeros in output while fanin 1 fanout 2 gates consume 4 in input and produce 2 zeros in output (see Figures 3.8 and 3.9). So, between g_{i}^* and g_j^* we have to distinct two cases: a) if both gates have the same number of inputs and outputs, connections are managed in the obvious way i.e., outputs corresponding to the computation performed by original gates are assigned between g_{i}^* and g_j^* and each gate uses the same zeros that they produce to feed its inputs. b) if g_{i}^* or g_j^* have more inputs than outputs or vice versa, we have to manage the extra zeros (needed or produced). Without loss of generality, we assume that g_{i}^* is fanin 2 fanout 1. Then, by the latter observation, g_{i}^* is in bijection with another gate g_k and thus, a gadget block g_k^* with fanin 1 and fanout 2. We simply connect extra zeros produced by g_{i}^* to block g_k^* and we do the same that we did in the previous case in order to manage connections.

Note that F^* can be built in **DLOGSPACE** since it suffices to read the standard representation of F and produce associated block gadgets which have constant size. In addition we have that previous encoding $g \to g^*$ induces a block map $\phi : \{0,1\}^V \to \{0,1\}^{V^+}$ where $|V^+| = \mathcal{O}(|V|)$ and that $\phi \circ F = F^{*T} \circ \phi$ where $T = 6$ is the size of each gadget block in F^*. We conclude that $F^* \in \Gamma(\mathcal{G}_{m,2})$ simulates F in space $|V^+| = \mathcal{O}(|V|)$ and time $T = 6$ and thus, $\Gamma(\mathcal{G}_m) \preccurlyeq_S^T \Gamma(\mathcal{G}_{m,2})$ where T is constant and S is a linear function.

□

Corollary 3.15 *The family $\Gamma(\mathcal{G}_{m,2})$ is strongly universal.*

Proof. The result comes directly from Theorem 3.13 ($\Gamma(\mathcal{G}_m)$ is strongly universal) and Theorem 3.14 ($\Gamma(\mathcal{G}_m) \preccurlyeq_S^T \Gamma(\mathcal{G}_{m,2})$ where T is constant and S is a linear function). □

Now we show a result on universality of $\Gamma(\mathcal{G}_m)$ which is essentially a consequence of [45, Theorem 6.2.5]. The latter result starts with alternated monotonic circuit which has only fanin 2 and fanout 2 gates (previous results in the same reference show that one can always reduce to this case starting from an arbitrary circuit) and gives an $\mathbf{NC}^1$ construction of a synchronous alternating circuit preserving the previous property on the fanin and fanout of each gate. The main remark in this context is that the construction uses quadratic space in the number of gates of the circuit given in input, so we cannot show strong universality but only universality with this approach.

Theorem 3.16 *The family $\Gamma(\mathcal{G}_m)$ of all $\mathcal{G}_m$-networks is universal.*

Proof. Let $F : Q^k \to Q^k$ be some arbitrary network with a circuit representation $C : \{0,1\}^n \to \{0,1\}^n$ such that $n = k^{\mathcal{O}(1)}$. By [45, Theorem 6.2.5] we can assume that there exists a circuit $C' : \{0,1\}^{n'} \to \{0,1\}^{n'}$ where $n' = \mathcal{O}(n^2)$ such that C' is synchronous alternated and monotonic. In addition, every gate in C' has fanin and fanout 2. We remark that the latter reference does not only provide the standard encoding of C' but also gives us a **DLOGSPACE** algorithm (it is actually $\mathbf{NC}^1$) which takes the standard representation of $C : \{0,1\}^n \to \{0,1\}^n$ and produces C'. We are going to slightly modify the latter algorithm in order to construct not only a circuit but a $\mathcal{G}_m$-network. In fact, the only critical point is to manage the identification between outputs and inputs. This is not direct from the result by Ruzzo et al. as their algorithm involves duplication of inputs and also adding constant inputs. In order to manage this, it suffices to modify their construction in order to mark original, copies and constant inputs. Then, as $\mathcal{G}_m$ includes COPY gates and also AND/OR gates with fanout 1, one can always produce copies of certain input if we need more, or erase extra copies of outputs or constants produced at output by adding a small circuit of $\mathcal{O}(\log(n))$ depth consisting in a several fanin 2 fanout 1 gates forming a tree. Same idea applies for constant inputs. Formally, at the end of the algorithm, the **DLOGSPACE** algorithm can read extra information regarding copies and constant inputs, and then can construct a $\mathcal{O}(\log(n))$ depth circuit that produces a coherent encoding for inputs and outputs. This latter construction defines a $\mathcal{G}_m$-network $G : \{0,1\}^{n''} \to \{0,1\}^{n''}$ and an encoding $\phi : Q^n \to \{0,1\}^{n''}$ where

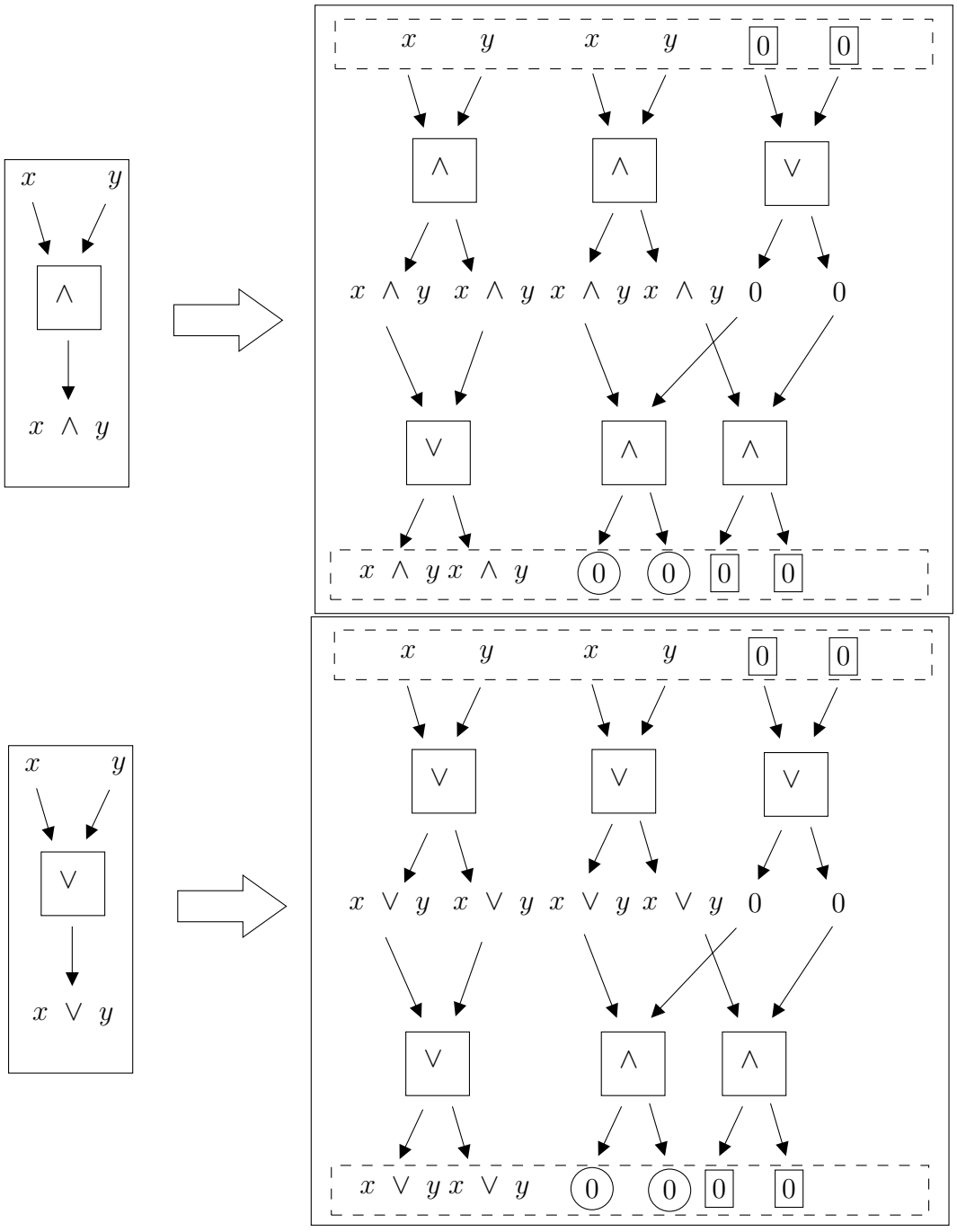


Figure 3.8: Block gadgets for simulating fanin 2 fanout 1 AND/OR gates using only gates in $\mathcal{G}_{m,2}$. Squared zeros represent the amount of zeros that can be used as inputs for the same block. Circled zeros correspond to extra zeros that need to be received from a Fanin 1 Fanout 2 gate.

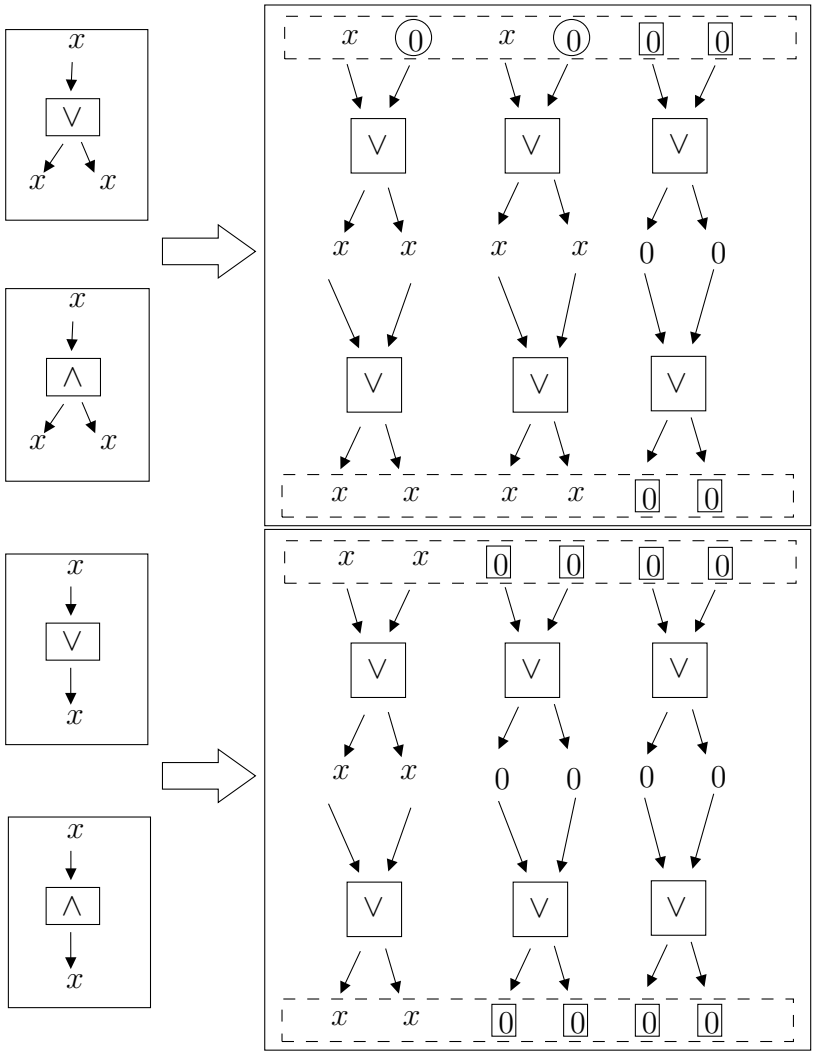


Figure 3.9: Block gadgets for simulating fanin 1 fanout 2 and fanin 1 fanout 1 AND/OR gates using only gates in $\mathcal{G}_{m,2}$. Squared zeros represent the amount of zeros that can be used as inputs for the same block. Circled zeros correspond to extra zeros that need to be received from a fanin 2 fanout 1 gate.

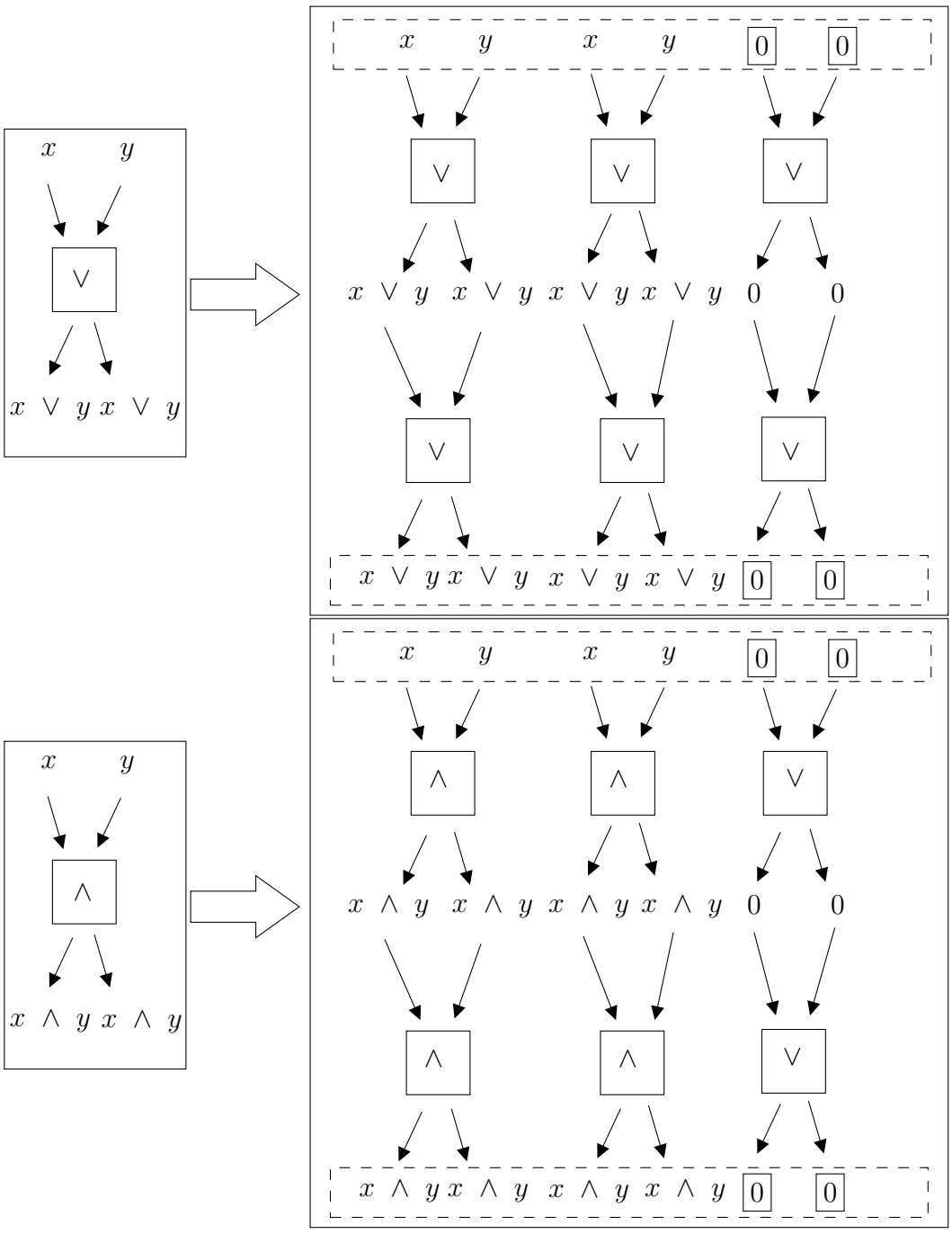


Figure 3.10: Block gadgets for simulating fanin 2 fanout 2 AND/OR gates using only gates in $\mathcal{G}_{m,2}$. Squared zeros represent the amount of zeros that can be used as inputs for the same block.

$n'' = \mathcal{O}(n^2)$ such that $\phi \circ F = G^T \circ \phi$ where $T = \mathcal{O}(\text{depth(C')} + \log(n))$. Thus, $\mathcal{G}_m$ is universal. □

We present now the following direct corollary.

Corollary 3.17 *Let $\mathcal{F}$ be a strongly universal automata network family. Then, $\mathcal{F}$ is universal.*

Proof. It suffices to exhibit a $\mathcal{G}$-network family ($\mathcal{G}$-networks are bounded degree networks) which is strongly universal and universal at the same time. By Theorem 3.16 we take $\mathcal{G} = \mathcal{G}_m$ and thus, the corollary holds. □

Corollary 3.18 *Let $\mathcal{G}$ be either $\mathcal{G}_m$ or $\mathcal{G}_{m,2}$. Any family $\mathcal{F}$ that has coherent $\mathcal{G}$-gadgets contains a subfamily of bounded degree networks with bounded degree representation which is (strongly) universal. Any CSAN family with coherent $\mathcal{G}$-gadgets is (strongly) universal.*

3.3.1 Closure and synchronous closure

Although monotonic gates are sometimes easier to realize in concrete dynamical systems, which make the above results useful, there is nothing special about them to achieve universality: any set of gates that are expressive enough for Boolean functions yields the same universality result. Given a set of maps $\mathcal{G}$ over alphabet Q, we define its *closure* $\overline{\mathcal{G}}$ as the set of maps that are computed by circuits that can be built using only gates from $\mathcal{G}$. More precisely, $\overline{\mathcal{G}}$ is the closure of $\mathcal{G}$ by composition, *i.e.* forming from maps $g_1 : Q^{I_1} \to Q^{O_1}$ and $g_2 : Q^{I_2} \to Q^{O_2}$ (with I_1, I_2, O_1, O_2 disjoint) a composition g by plugging a subset of outputs $O \subseteq O_2$ of g_1 into a subset of inputs $I \subseteq I_2$ of g_2, thus obtaining $g : Q^{I_1 \cup I_2 \setminus I} \to Q^{O_1 \setminus O \cup O_2}$ with

$$g(x)_o = \begin{cases} g_1(x_{I_1})_o & \text{if } o \in O_1 \setminus O, \\ g_2(y)_o & \text{if } o \in O_2, \end{cases}$$

where $y_j = x_j$ for $j \in I_2 \setminus I$ and $y_j = g_1(x_{I_1})_{\pi(j)}$, where $\pi : I \to O$ is the chosen bijection between I and O (the wiring of outputs of g_1 to inputs of g_2). A composition is *synchronous* if either $I = \emptyset$ or $I = I_2$. We then define the *synchronous closure* $\overline{\mathcal{G}}^2$ as the closure by synchronous composition. The synchronous composition correspond to synchronous circuits with gates in $\mathcal{G}$. A $\mathcal{G}$-circuit is a sequence of compositions starting from elements of $\mathcal{G}$. It is synchronous if the compositions are synchronous. The *depth of a $\mathcal{G}$-circuit* is the maximal length of a path from an input to an output. In the case of a synchronous circuits, all such paths are of equal length.

Proposition 3.19 *Fix some alphabet Q and consider two finite sets of maps $\mathcal{G}$ and $\mathcal{G}'$ over alphabet Q such that:*

- *either $\mathcal{G}$ contains the identity map $Q \to Q$ and is such that $\overline{\mathcal{G}}$ contains $\mathcal{G}'$;*

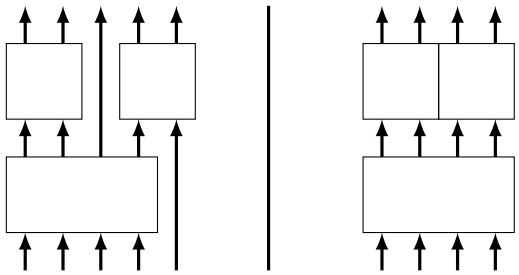


Figure 3.11: (Left panel) Non-synchronous composition and (Right panel) synchronous composition.

- *or there is an integer k such that $\overline{\mathcal{G}}^2_k$, the set of elements of $\overline{\mathcal{G}}^2$ that can be realized by a circuit of depth k contains $\mathcal{G}'$.*

Then, any family $\mathcal{F}$ that has coherent $\mathcal{G}$-gadgets has coherent $\mathcal{G}'$-gadgets.

Proof. Suppose first that the first item holds. Since $\overline{\mathcal{G}}$ contains $\mathcal{G}'$ there must exist a circuit made of gates from $\mathcal{G}$ that produces any given element $g \in \mathcal{G}'$. Then one wants to apply gadget glueing on gadgets from $\mathcal{G}$ to mimic the composition and thus obtain a gadget corresponding to g. However this does not work as simply because propagation delay is a priori not respected at each gate in the circuit composition yielding g since layers in the circuit could be connected in a way such that the information arrives at different times to each layer. However, since $\mathcal{G}$ contains the identity map, there is a corresponding gadget in the family that actually implements a delay line, allowing us to synchronize the evaluation of the circuit. This additional gadget solves the problem: it is straightforward to transform by padding with identity gates all circuits with gates in $\mathcal{G}$ into synchronous ones. Moreover, by padding again, we can assume that the finite set of such circuits computing elements of $\mathcal{G}_m$ are all of same depth.Then, this set of circuits can be trasformed into coherent $\mathcal{G}_m$-gadgets by iterating gadget glueing and using Lemma 3.2.

If the second item holds the situation is actually simpler because the synchronous closure contains only synchronous circuits of gates from $\mathcal{G}_{m,2}$. So we can directly translate the circuits producing the maps of $\mathcal{G}_{m,2}$ into gadgets via gadget glueing by Lemma 3.2 as in the previous case. Moreover, as a consequence of the hypothesis, we have that each circuit in the set of circuits having gates in $\mathcal{G}'$ can be considered to have the same depth. Thus, we get gadgets that share the same time constant.

□

3.3.2 Super-polynomial periods without universality

A universal family must exhibit super-polynomial periods, however universality is far from necessary to have this dynamical feature. In this subsection we define the family of wire networks to illustrate this.

To this end, we need the following classical result about the growth of Chebyshev function and prime number theorem.

Lemma 3.20 [47] *Let $m \geq 2$ and $\mathcal{P}(m) = \{p \leq m \mid p \text{ prime}\}$. If we define $\pi(m) = |\mathcal{P}(m)|$ and $\theta(m) = \sum_{p \in \mathcal{P}(m)} \log(p)$ then we have $\pi(m) \sim \frac{m}{\log(m)}$ and $\theta(m) \sim m$.*

By using the Lemma 3.20 we can construct automata networks with non-polynomial limit cycles simply by making disjoint union of rotations (*i.e.* network whose interaction graph is a cycle graph that just rotate the configuration at each step). Indeed, it is sufficient to consider rotations on cycle graphs whose length are successive prime numbers. It turns out that these automata networks are exactly $\mathcal{G}_w$-networks where $\mathcal{G}_w$ is a single 'wire gate': $\mathcal{G}_w = \{\mathrm{id}_B\}$ where id_B is the identity map over $\{0,1\}$.

Formally, according to Definition 3.4 (see Figure 3.4), for any $\mathcal{G}_w$-network $F : Q^V \to Q^V$ there exist a partition $V = C_1 \cup C_2 \ldots \cup C_k$ where $C_\mathrm{i} = \{u_1^\mathrm{i} \ldots, u_{l_\mathrm{i}}^\mathrm{i}\}$ with $l_\mathrm{i} \geq 2$ for each $\mathrm{i} = 1, \ldots, k$ and $F(x)_{u_{s+1 \bmod l_\mathrm{i}}^\mathrm{i}} = x_{u_s^\mathrm{i}}$ for any $x \in Q^V$ and $0 \leq s \leq l_\mathrm{i}$.

Theorem 3.21 *Any family $\mathcal{F}$ that has coherent $\mathcal{G}_w$-gadgets has superpolynomial limit cycles, more precisely: there is some $\alpha > 0$ such that for infinitely many $n \in \mathbb{N}$, there exists a network $F_n \in \mathcal{F}$ with $O(n)$ nodes and a periodic orbit of size $\Omega(\exp(n^\alpha))$.*

Proof. Taking the notations of Lemma 3.20, define for any n the $\mathcal{G}_w$-network G_n made of disjoint union of cycle graphs of each prime length less than n. G_n has size at most $n\pi(n)$ and if we consider a configuration x which is in state 1 at exactly one node in each of the $\pi(n)$ disjoint cycle graphs, it is clear that the orbit of x is periodic of period $\exp\theta(n)$. Therefore, from Lemma 3.20, for any n, G_n is a cycle graph of size $m \leq n\pi(n)$ with a periodic orbit of size $\theta(n) \in \Omega(\exp(\sqrt{m \log m}))$. By hypothesis there are linear maps T and S such that for any n, there is F_n that simulates G_n (by Lemma 3.10), therefore F_n also has a super-polynomial limit cycle cycle by Lemma 2.6. □

3.3.3 Conjunctive networks and $\mathcal{G}_{conj}$-networks

Let $G = (V, E)$ be any directed graph. The conjunctive network associated to G is the automata network $F_G : \{0,1\}^V \to \{0,1\}^V$ given by $F(x)_\mathrm{i} = \wedge_{j \in N^-(\mathrm{i})} x_j$ where $N^-(\mathrm{i})$ denotes the incoming neighborhood of i. Conjunctive networks are thus completely determined by the interaction graph and a circuit representation can be deduced from this graph in **DLOGSPACE**. We define the family $\mathcal{F}_{conj}$ as the set of conjunctive networks together with the standard representation $\mathcal{F}^*_{conj}$ which are just directed graphs encoded as finite words in a canonical way.

Remark 3.22 We can of course do the same with disjunctive networks. Any conjunctive network F_G on graph G is conjugated to the disjunctive network F'_G on the same graph by the negation map $\rho : \{0,1\}^V \to \{0,1\}^V$ defined by $\rho(x)_\mathrm{i} = 1 - x_\mathrm{i}$, formally $\rho \circ F_G = F'_G \circ \rho$. In particular, this means that the families of conjunctive and disjunctive networks simulate each other. In the sequel we will only state results for conjunctive networks while they hold for disjunctive networks as well.

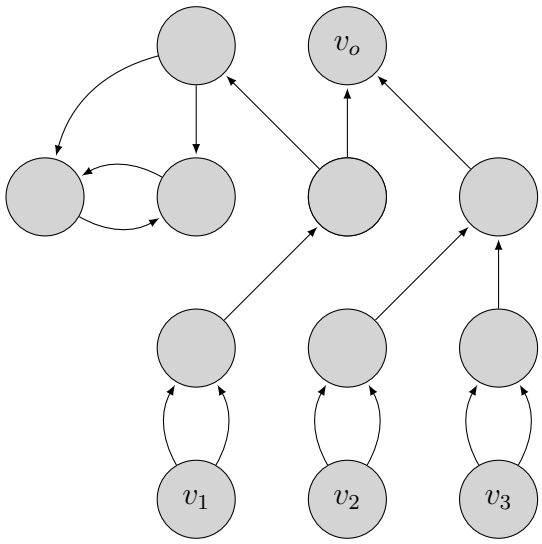


Figure 3.12: Fanin gadget of degree 3. For any configuration x, $F^3(x)_{v_o} = x_{v_1} \wedge x_{v_2} \wedge x_{v_3}$.

We define now the set $\mathcal{G}_{conj} = \{\text{AND}, \text{COPY}\}$. Observe that $\mathcal{G}_{conj}$-networks are nothing else but conjunctive networks with the following degree constraints: each node has either in-degree 1 and out-degree 2, or in-degree 2 and out-degree 1. The following theorem shows that, up to simulation, these constraints are harmless.

Theorem 3.23 *The family of $\mathcal{G}_{conj}$-networks simulates the family $(\mathcal{F}_{conj}, \mathcal{F}^*_{conj})$ of conjunctive networks in linear time and polynomial space.*

Proof. Let F be an arbitrary conjunctive network on graph $G = (V, E)$ with n nodes. Its maximal in/out degree is at most n. For each node of indegree $\mathrm{i} \leq n$ we can make a tree-like $\mathcal{G}_{conj}$-gadget with i inputs and 1 output that computes the conjunction of its i inputs in exactly n steps: more precisely, we can build a sub-network of size $O(n)$ with i identified 'input' nodes of fanin 1 and one identified output node of fanout 1 such that for any $t \in \mathbb{N}$ the state of the output node at time $t + n$ is the conjunction of the states of the input nodes at time t (the only sensible aspect is to maintain synchronization in the gadget, see Figure 3.12). We do the same for copying the output of a gate i times and dealing with arbitrary fanout. Then we replace each node of F by a meta node made of the two gadgets to deal with fanin/fanout and connect everything together according to graph G (note that fanin/fanout is granted to be 1 in the gadgets so connections respect the degree constraints). We obtain in **DLOGSPACE** a $\mathcal{G}_{conj}$-network of size polynomial in n that simulates F in linear time. □

Remark 3.24 The family of conjunctive networks can produce super-polynomial periods but it is not universal. There are several ways to show this. It is for instance impossible to produce super-polynomial transients within the family [19, Theorem 3.20] so Corollary 2.14 conclude. One could also use Corollary 2.19 since a node in a strongly connected component of a conjunctive network must have a trace period of at most the size of the component (actually much more is known about periods in conjunctive networks through the concept of

loop number or cyclicity, see [19]).

3.3.4 Super-polynomial transients without universality

Let us consider in this section the alphabet $Q = \{0, 1, 2\}$. We are going to define a set $\mathcal{G}_t$ such that $\mathcal{G}_t$-networks exhibit super-polynomial transients but are not universal. To help intuition, $\mathcal{G}_t$-networks can be though as standard conjunctive networks on $\{0, 1\}$ that can in some circumstances produce state 2 which is a spreading state (a node switches to state 2 if one of its incoming neighbors is in state 2). The extra state 2 will serve to mark super-polynomial transients, but it cannot escape a strongly connected component once it appears in it. As we will see, $\mathcal{G}_t$-networks are therefore too limited in their ability to produce large periodic behavior inside strongly connected components.

$\mathcal{G}_t$ is made of the following maps:

$$\mathrm{AND}_{\{0,1\}} : (x, y) \mapsto \begin{cases} 2 & \text{if } 2 \in \{x, y\} \\ x \wedge y & \text{else.} \end{cases}$$

$$\mathrm{AND}_2 : (x, y) \mapsto \begin{cases} 2 & \text{if } 2 \in \{x, y\} \text{ or } x = y = 1 \\ 0 & \text{else.} \end{cases}$$

$$\Lambda : (x, y) \mapsto \begin{cases} 2 & \text{if } 2 \in \{x, y\} \\ x & \text{else.} \end{cases}$$

$$\mathrm{Id} : x \mapsto x.$$

$$\Upsilon : x \mapsto (x, x).$$

$\mathcal{G}_t$-networks can produce non-polynomial cycles by disjoint union of rotations of prime lengths as in Theorem 3.21, but they can also wait for a global synchronization of all rotations and freeze the result of the test for this synchronization condition inside a small feedback loop attached to a "controlled AND map".

More precisely, as shown in Figure 3.13 we can use in the context of any $\mathcal{G}_t$-network a small module $T(x)$ made of five nodes with the following property: if the Λ node of the module is in state 0 in some initial configuration, then it stays in state 0 as long as node x is not in state 1, and when $x = 1$ at some time step t then, from step $t + 2$, the Λ node is in state 2 at least one step every two steps. This module is the key to control transient behavior.

Moreover, map $\mathrm{AND}_{\{0,1\}}$ behaves like standard Boolean AND map when its inputs are in $\{0, 1\}$. More generally, by combining such maps in a tree-like fashion, one can build modules $A(x_1, \ldots, x_k)$ for any number k of inputs with a special output node which has the following property for some time delay $\Delta \in O(\log(k))$: the output node at time $t + \Delta$ is in state 1 if and only if all nodes x_{i} (with $1 \leq \mathrm{i} \leq k$) are in state 1 at time t.

Combining these two ingredients, we can build upon the construction of Theorem 3.21 to obtain non-polynomial transients in any family having coherent $\mathcal{G}_t$-gadgets.

Theorem 3.25 *Any family $\mathcal{F}$ that has coherent $\mathcal{G}_t$-gadgets has superpolynomial transients,*

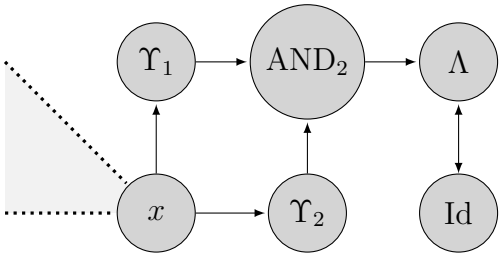


Figure 3.13: Freezing the result of a test in a $\mathcal{G}_t$-network. The module $T(x)$ is made of the nodes marked Υ, AND_2, Λ and Id. Observe that each node represent some output of its corresponding label(for more details on $\mathcal{G}$-networks see Definition 3.4). Each gate has one output with the exception of the gate Υ which is represented by two nodes. The module $T(x)$ reads the value of node x belonging to an arbitrary $\mathcal{G}_t$-network (represented in light gray inside dotted lines). The output Λ is fed back to its control input via the Id node (self-loops are forbidden in $\mathcal{G}_t$-networks). Note that x as well as the rest of the network is not influenced by the behavior of the gates of the module $T(x)$.

more precisely: there is some $\alpha > 0$ such that for any $n \in \mathbb{N}$, there exists a network $F_n \in \mathcal{F}$ with $O(n)$ nodes and a configuration x such that $F_n^t(x)$ is not in an attractor of F_n with $t \in \Omega(\exp(n^\alpha))$.

Proof. Like in Theorem 3.21, they key of the proof is to show that there is a $\mathcal{G}_t$-network with transient length as in the theorem statement, then the property immediately holds for networks of the family $\mathcal{F}$ by Lemma 3.10 and Lemma 2.6.

For any $n > 0$ we construct a $\mathcal{G}_t$-network G_n made of two parts:

- The "bottom" part of G_n uses a polynomial set of nodes B_n and consists in a disjoint union of circuits for each prime length less than n as in Theorem 3.21, but where for each prime p, the circuit of length p has a node v_p which implements a copy gate COPY, thus not only sending its value to the next node in the circuit, but also outputting it to the "top" part of G_n.
- The "top" part of G_n is made of a module $A(x_1, \ldots, x_k)$ connected to all nodes v_p as inputs and whose output is connected to a test module $T(x)$ as in Figure 3.13.

Note that the size of G_n is polynomial in n. With this construction we have the following property as soon as the modules $A(x_1, \ldots, x_k)$ and $T(x)$ are initialized to state 0 everywhere: as long as nodes v_p are not simultaneously in state 1 then the output of the test module $T(x)$ stays in state 0; moreover, if at some time t nodes v_p are simultaneously in state t, then after time $t + O(\log(t))$, the node Λ of module $T(x)$ is in state 2 one step every two steps. This means that $t + O(\log(t))$ is a lower bound on the transient of the considered orbit. To conclude the proof it is sufficient to consider the initial configuration where all nodes are in state 0 except the successor of node v_p in each circuit of prime length p, which are in state 1. In this case it is clear that the first time t at which all nodes v_p are in state 1 is the product of prime numbers less than n. As in Theorem 3.21, we conclude thanks to Lemma 3.20. □

As said above, $\mathcal{G}_t$-networks are limited in their ability to produce large periods. More

precisely, as shown by the following lemma, their behavior is close enough to conjunctive networks so that it can be analyzed as the superposition of the propagation/creation of state 2 above the behavior of a classical Boolean conjunctive network. To any $\mathcal{G}_t$-network F we associate the Boolean conjunctive network F^* with alphabet $\{0,1\}$ as follows: nodes with local map $\mathrm{AND}_{\{0,1\}}$ or AND_2 are simply transformed into nodes with Boolean conjunctive local maps on the same neighbors, nodes with local maps Υ or Id are left unchanged (only their alphabet changes), and nodes with map $\Lambda(x,y)$ are transformed into a node with only x as incoming neighborhood.

Lemma 3.26 *Let F be a $\mathcal{G}_t$-network with node set V and F^* its associated Boolean conjunctive network. Consider any $x \in \{0,1,2\}^V$ and any $x^* \in \{0,1\}^V$ such that the following holds:*

$$\forall v \in V : x_v \in \{0,1\} \Rightarrow x^*_v = x_v,$$

then the same holds after one step of each network:

$$\forall v \in V : F(x)_v \in \{0,1\} \Rightarrow F^*(x^*)_v = F(x)_v.$$

Proof. It is sufficient to check that if $F(x)_v \neq 2$, it means that all its incoming neighbors are in $\{0,1\}$ so x and x^* are equal on these incoming neighbors, and that it only depends on neighbor a in the case of a local map $\Lambda(a,b)$. In any case, we deduce $F^*(x^*)_v = F(x)_v$ by definition of F^*. □

Theorem 3.27 *The family of $\mathcal{G}_t$-networks is not universal.*

Proof. Consider a Boolean conjunctive automata network F, a configuration x with periodic orbit under F and some node v such that there is a walk of length L from v to v. We claim that $x_v = F^L(x)_v$ so the trace at node v in x is periodic of period less than L. Indeed, in a conjunctive network state 0 is spreading so clearly if $x_v = 0$ then $F^L(x)_v = 0$ and, more generally, $F^{kL}(x)_v = 0$ for any $k \geq 1$. On the contrary, if $x_v = 1$ then we cannot have $F^L(x)_v = 0$ because then $F^{Pk}(x)_v = 0$ with P the period of x which would imply $x_v = 0$.

With the same reasoning, if we consider any $\mathcal{G}_t$-network F, any configuration x with periodic orbit and some node v such that there is a walk of length L from v to v, then it holds:

$$x_v = 2 \Leftrightarrow F^L(x)_v = 2.$$

We deduce thanks to Lemma 3.26 that for any configuration x with periodic orbit of some $\mathcal{G}_t$-network F with n nodes, and for any node v belonging to some strongly connected component, the period of the trace at v starting from x is less than n^2: it is a periodic pattern of presence of state 2 of length less than n superposed on a periodic trace on $\{0,1\}$ of length less than n. We conclude that the family of $\mathcal{G}_t$-networks cannot be universal thanks to Theorem 2.19. □

Chapter 4

Describing asynchronous dynamics: a deterministic approach

In this chapter, we study the dynamics of automata networks under different update schemes. Recall that, generally speaking, automata networks dynamics is defined in terms of a global rule $F : Q^n \to Q^n$. If we see now F as a network, i.e. a collection of entities that are somehow related, we can interpret the evolution of this dynamics as defined by a scheme in which each of these entities is updated at the same time. This dynamics is usually called *parallel* or *synchronous*. As we have pointed out previously, this approach is considered usually unrealistic for some applications. Probably the most iconic case (to consider) in this context is that of applications in gene regulatory networks. Within this domain, the synchronous dynamics seems to be inaccurate since it can be interpreted as a type of dynamics in which the expression of all genes takes place simultaneously. Considering the latter observation, there are some classical types of breaking the synchronous update scheme or introducing asynchronicity to the dynamics of a fixed automata network. As we work mainly with CSAN families for all the applications considered in Chapter 5, we define update schemes exclusively for this type of automata networks. However, all our definitions can be extended to the general case. Particularly, we explore in the first section of this chapter three classes of update schemes which are widely referenced in the literature and also studied from the point of view of how they impact the original synchronous dynamics. These classes are: periodic update schemes, local clocks update schemes and block sequential update schemes. We start by establishing a strict hierarchy between these three classes and studying some examples. We discuss also the limits of periodic dynamics compared to other types of update schemes. Then, we present a general framework which allows us to see the dynamics defined by means of update schemes as a projection of the dynamics of another automata network which is deterministic and operates over a bigger alphabet, which contains the original alphabet as a component. This approach does not only include periodic update schemes, and thus, local clocks and block sequential update schemes, but also considers some of more general update schemes that we will define in this section.

4.1 Update schemes

We recall that given an automata network $F : Q^V \to Q^V$, we can define a dynamical system over the alphabet Q by the subsequent iterations of F . This is the most natural way to define a dynamics from an automata network and it is usually said that in this case that the dynamics follows a *paralell* update scheme. This name is because, if we see any coordinate i of F as a node in G and its associated local function F_i given by λ and ρ, then, it updates its state at each time step. In other words, all the nodes execute their local function synchronously. Nevertheless one can also induce a dynamical system over Q by considering other ways of updating each node in the network in which not all the nodes are updated at the same time. Observe that, generally speaking, this latter notion demands some sort of temporal order in the nodes that will indicate which nodes have to be updated at a particular time step. In this sense, we present a more general definition that uses the concept of an *update scheme*.

Definition 4.1 *Let $F : Q^V \to Q^V$ an automata network. An* update scheme *is a sequence $\mu : \mathbb{N} \to 2^V$. Given an update scheme μ and a natural number $k \in \mathbb{N}$ we call an* intermediate step *of the dynamics given by μ the function $F^{\mu(k)}$ defined by $F^{\mu(k)}(x)_\mathrm{i} = \begin{cases} F(x)_\mathrm{i} & \text{if } \mathrm{i} \in \mu(k), \\ x_\mathrm{i} & \text{otherwise.} \end{cases}$.*

We define an orbit given by μ starting from some $x \in Q^V$ as the sequence $\mathcal{O}_{\mu,F}(x) = (x, F^{\mu(0)}(x), F^{\mu(1)}(F^{\mu(0)}(x)), F^{\mu(2)}(F^{\mu(1)}(F^{\mu(0)}(x))), \ldots)$

In this thesis work, we focus in a particular subclass of update schemes named *periodic update schemes*, i.e. the function μ is periodic, as the finite related sequence of sets that defines which set of nodes is updated at each intermediate time step. In the next sections, we are interested in studying two different subclasses of periodic update schemes: the "block sequential" ones and the "local clocks" ones. We define properly these subclasses in the next subsection.

4.1.1 Periodic update schemes

One of the most studied types of update schemes are the *periodic update schemes*. This class contains the parallel update scheme, in which all the nodes of the networks are updated at the same time, the sequential update schemes in which only one node its updated in each time step following a given order (see Figure 4.1) and also the block sequential update schemes in which, given an ordered partition of the node set V, the nodes are updated sequentially following the order of the sets in the partition but in parallel with respect to the other nodes that belong to the same set. In addition, we explore a new class of update schemes which contains the block sequential ones that it is called *local clocks*. In this class each node contains an internal clock that periodically controls when this node is going to update. More precisely, each node v is updated once every p_v steps but the frequency of update p_v might depend on the node.

Definition 4.2 *We say that an update scheme μ is a* periodic update scheme *if there exists $p \in \mathbb{N}$ such that $\forall t \in \mathbb{N}$, $\mu(t + p) = \mu(t)$. Moreover, we say that μ is:*

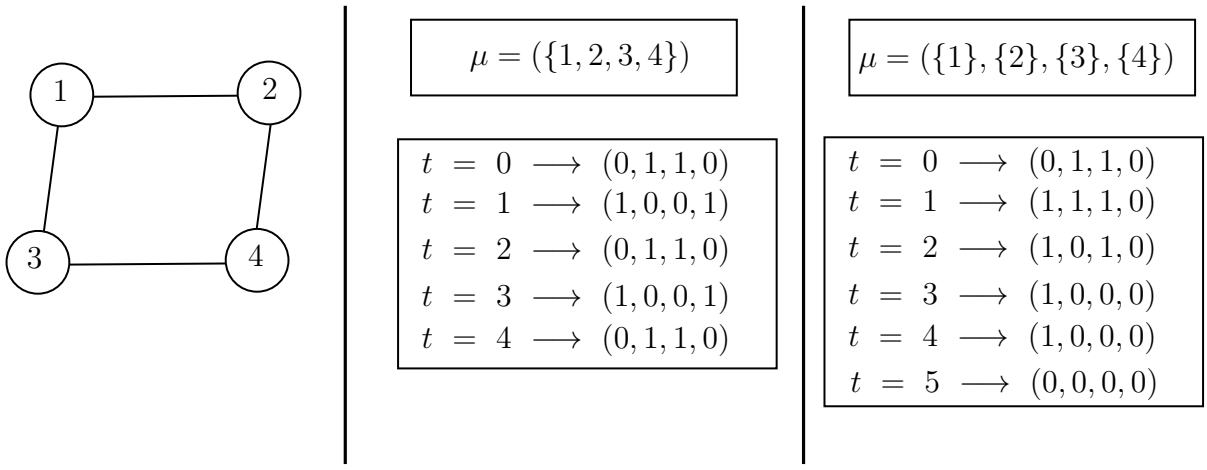


Figure 4.1: Synchronous update scheme and sequential update scheme for the same conjunctive automata network. (Left panel) communication graph of the network. Local function is given by the minimum (AND function) over the set of states of neighbors for each node. (Central panel) Synchronous or parallel update scheme and the associated dynamics of configuration $(0,1,1,0)$. In this case μ has period 1 and all nodes are updates simultaneously. Observe that dynamics exhibits an attractor of period 2 (Right panel) Sequential update scheme and the associated dynamics of configuration $(0,1,1,0)$. In this case function μ has period $n = 4$ and only one node is updated at each time step. Dynamics reach a fixed point given by $\vec{0}$.

- *a* block sequential scheme *if there are subsets (called blocks)* $B_0, \ldots, B_{p-1} \subseteq V$ *forming an ordered partition of* V *such that* $\forall t \in \mathbb{N}$, $\mu(t) = B_{t \bmod p}$,
- *a* local clocks scheme *if for each* $v \in V$ *there is a local period* $p_v \in \mathbb{N}$ *and a local shift* $\tau_v \in \{0, \ldots, p_v - 1\}$ *such that* $v \in \mu(t) \iff t = \tau_v \bmod p_v$.

For a concrete example on how these update schemes work, see Figure 4.2 in which different dynamics for a simple conjunctive network under block sequential and local clocks update schemes are shown.

Observe that, block sequential and local clocks schemes are clearly periodic schemes. Moreover, any block sequential scheme given by $B_0, \ldots, B_{p-1} \subseteq V$ is a local clocks scheme given by $p_v = p$ and $\tau_v = \mathrm{i} \iff v \in B_\mathrm{i}$ for all $v \in V$. As already said, block sequential schemes can thus be seen as local clocks schemes where all nodes share the same update frequency. General periodic update schemes allows different time intervals between two consecutive updates of a node, which local clocks schemes obviously cannot do (see Figure 4.3). We will see later the tremendous consequences that such subtle differences in time intervals between updates at each node can have. For now let us just make the formal observation that the inclusions between these families of update schedules are strict when focusing on the sets of maps μ.

Remark 4.3 A so-called *block-parallel* scheme has also been considered more recently [20] which is defined by a set of lists of nodes $L_\mathrm{i} = (v_{\mathrm{i},j})_{0 \leq j < p_\mathrm{i}}$ (for $1 \leq \mathrm{i} \leq k$) forming a partition (*i.e.* such that $v_{\mathrm{i},j}$ are all distinct over j and $\cup_{\mathrm{i},j} v_{\mathrm{i},j} = V$) to which it is associated the map μ

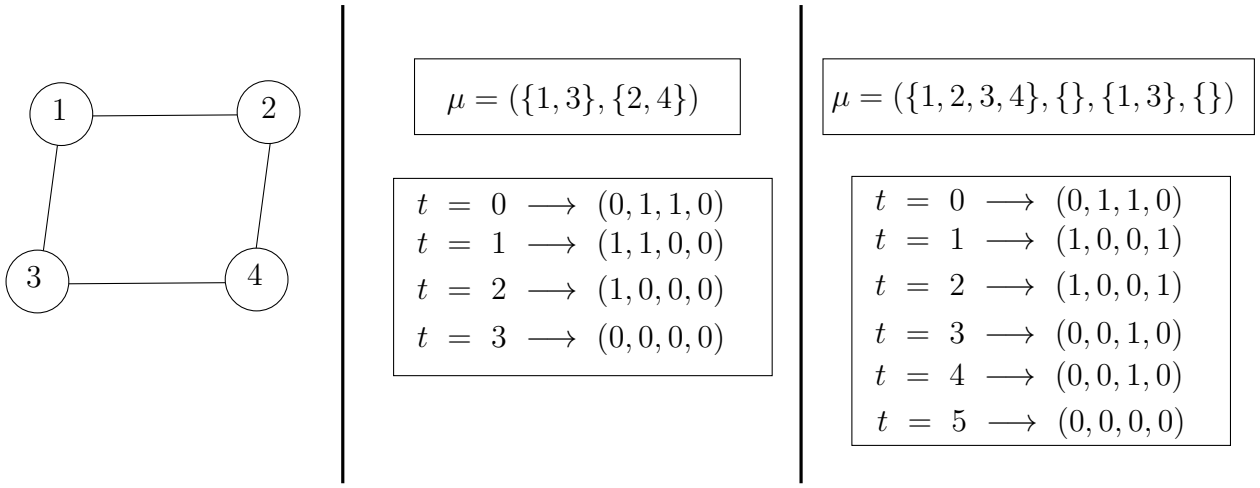


Figure 4.2: A block sequential and a local clocks update schemes over a simple conjunctive network. (Left panel) communication graph of the network. Local functions are given by the minimum (AND) over the states of the neighbors of each node. (Central panel) block sequential update scheme and the associated dynamics of configuration $(0,1,1,0)$. In this case function μ is defined by two blocks: $\{1,3\}$ and $\{2,4\}$. Dynamics reach a fixed point after 3 time steps. (Right panel) local clocks update scheme and the associated dynamics of configuration $(0,1,1,0)$. In this case each node has an internal clock with different local periods. Nodes 1 and 3 are updated every two steps ($p_1 = p_3 = 2$) and nodes 2 and 4 are updated every 4 time steps (i.e. $p_2 = p_4 = 4$). The shift parameters is 0 for all nodes $\tau_1 = \tau_2 = \tau_3 = \tau_4 = 0$. The dynamics reaches a fixed point after 5 time steps.

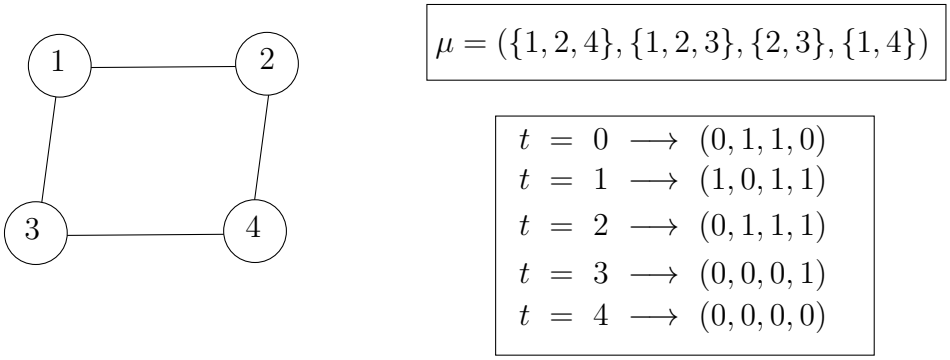
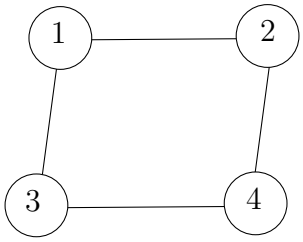


Figure 4.3: A general periodic update scheme over a conjunctive network. In this case μ has period 4 and the underlying dynamics reaches a fixed point after 4 time steps. Observe that there is no restriction on how many times a node is updated. For example, 1 is updated 3 times every 4 time steps but 4 is updated only twice every 4 time steps.



$$\mu = (\{1,2,4\},\{1,3,4\})$$

$$\begin{array}{rcl} t = 0 & \longrightarrow & (0,1,1,0) \\ t = 1 & \longrightarrow & (1,0,1,1) \\ t = 2 & \longrightarrow & (0,0,1,0) \\ t = 3 & \longrightarrow & (0,0,0,0) \end{array}$$

Figure 4.4: A block parallel update scheme defined over a conjunctive network. Updating list is given by $L = \{(1),(2,3),(4)\}$. Observe that a constant amount of nodes (equal to the length of L, i.e., 3) is updated at each time step. Dynamics reaches a fixed point after 3 time-steps.

such that $v_{\mathrm{i},j} \in \mu(n) \iff j = n \bmod p_{\mathrm{i}}$. It is a particular case of our definition of local clocks scheme above with the additional constraints that the size of the set $\mu(n)$ of updated nodes is constant with n (see Figure 4.4 for an example). We note that, conversely, any local clocks scheme on a given network can be simulated by a block-parallel scheme by artificially adding disconnected nodes that do nothing but satisfy the constraint of $\mu(n)$ being of constant size.

4.1.2 Projection systems

We recall that Q is a finite set that we call alphabet. Now we present a dynamical formalism that allows us to include all the update schemes presented above and possibly other ones into one formalism. We remark that in all periodic schemes, a given node can take the decision to update or not by simply keeping track of the current value of time modulo the period. Thus, one could see the system as a non-deterministic system in which possibilities are given by all the possible ways to update nodes respecting a periodic update scheme. However, a key observation in this context is that, when we add the knowledge of time modulo the period at each node as a new component of states, the whole system becomes deterministic. In fact, we can recover the original dynamics of some automata network with alphabet Q under a periodic update scheme by projecting a specific deterministic automata network with alphabet $Q' \times Q$ onto Q.

Let $F : Q^V \to Q^V$ be an automata network. We recall that its *asynchronous* version is an automata network that at every time-step (non-deterministically) choose if a node i should be updated or if it will be stay in the same state. More precisely, a *asynchronous* version of F is a non-deterministic function $F^* : Q^V \to \mathcal{P}(Q^V)$ such that $F^*(x) = \{y \in Q^V : \exists U \subset V, y_v = F(x)_v$ if $v \in U$ and $y_v = x_v$ otherwise.$\}$. Equivalently, we can see F^* as a function $F^* : Q^V \to (\mathcal{P}(Q))^V$ such that $F^*(x)_{\mathrm{i}} = \{x_{\mathrm{i}}, F(x)_{\mathrm{i}}\}$. Note that, analogously to the deterministic case one can define an orbit starting from x of F^* as a sequence of states $\mathcal{O}_{F^*}(x) = x^0 = x, x^1, x^2, \ldots, x^t, \ldots, \in Q^n$ such that $x^s_{\mathrm{i}} \in F(x^{s-1})_{\mathrm{i}}$ for $\mathrm{i} \in V$ and $s \geq 1$. Note also that, given $x \in Q^V$ and an orbit $\mathcal{O}_{F^*}(x)$ we can see $\mathcal{O}_{F^*}(x)$ as a particular realization of a certain update scheme μ. More precisely, there exists an update scheme μ (which is defined in the obvious way i.e. by updating the corresponding nodes in every time step according to points in $\mathcal{O}_{F^*}(x)$) such that for every $x^s \in \mathcal{O}_{F^*}(x)$ we have $x^s = (\mathcal{O}_{F,\mu}(x))^s$. In addition,

we have that for each update scheme μ there exist an orbit of F^* which coincides with its dynamics in every time step. Thus, we want to work with F^* in order to globally study a whole class of update schemes. However, we would like to continue working in a deterministic framework in order to keep things simple (in particular the notion of simulation that we define later). In order to achieve this task, we introduce the following notion of asynchronous extension which is a way to produce the dynamics of different update schemes through projection.

Definition 4.4 *Let Q be a finite alphabet and $Q' = Q \times R$ where R is finite. Let $F : Q^V \to Q^V$ and $F' : Q'^V \to Q'^V$ two automata networks. We say that F is a* projection system *of F' and that F' is an* asynchronous extension *of F if:*

$$\forall x \in Q'^V,\ \overline{\pi}(F'(x)) \in F^*(\overline{\pi}(x)), \tag{4.1}$$

where $\overline{\pi}$ is the node-wise extension of the projection $\pi : Q' \to Q$ such that $\pi(q, r) = q$ for all $q' = (q, r) \in Q'$.

We show hereunder that the dynamics associated to any of the previously presented periodic update schemes can be described as an asynchronous extension over a product alphabet. To simplify notations we sometimes identify $(A \times B \times \cdots)^V$ with $A^V \times B^V \times \cdots$.

First we introduce the block sequential extensions. Generally speaking, given an ordered partition $\{B_1, \dots B_b\}$ defining a block sequential update scheme, we define a product space in which the second coordinate contains an internal clock which counts the evolution of time modulo b. For each node, its corresponding block is coded in the initial configuration and it is updated every b time steps.

Definition 4.5 (block sequential extension) *Let $F : Q^V \to Q^V$ be an automata network. Let $b \leq n$ and let $Q' = Q \times \{0, \dots, b-1\}$. We define the* block sequential extension *of F with b blocks as the automata network $F' : (Q')^V \to (Q')^V$ such that for all $x = (x_Q, x_b) \in Q'^V$ and all $v \in V$:*

$$F'(x)_v = \begin{cases} (F(x_Q)_v, (x_b)_v - 1 \mod b) & \text{if } (x_b)_v = 0, \\ ((x_Q)_v, (x_b)_v - 1 \mod b) & \text{otherwise.} \end{cases}$$

Now we introduce the definition of the local clocks extensions. In this case, as same as the case of block sequential extensions, the second coordinate stores the dynamics of an internal clock which periodically registers the evolution of time. However, in local clocks update scheme, each node can have its own local period, so clocks are note synchronized as in block sequential extension. As a consequence, we have to define a third coordinate which contains the information about the local period of each node. This component is constant and each local period is bounded by some parameter c called *clock delay*.

Definition 4.6 (local clocks extension) *Let $F : Q^V \to Q^V$ be an automata network. Let $c \in \mathbb{N}$ and let $Q' = Q \times \{0, \dots, c-1\} \times \{1, \dots, c\}$. We define the* local clocks *extension of F with clock delay c as the automata network $F' : (Q')^V \to (Q')^V$ such that for all $x =$*

$(x_Q, x_c, x_m) \in Q'^V$ *and all* $v \in V$*:*

$$F'(x)_v = \begin{cases} (F(x_Q)_v, (\psi_{(x_m)_v}[(x_c)_v]) + 1 \mod (x_m)_v, (x_m)_v) & \text{if } (x_c)_v = 0, \\ ((x_Q)_v, (\psi_{(x_m)_v}[(x_c)_v]) + 1 \mod (x_m)_v, (x_m)_v) & \text{else.} \end{cases}$$

$$\psi_m(r) : \{0, \ldots, c-1\} \to \{0, \ldots, c-1\} \text{ is such that } \psi_m(r) = \begin{cases} r & \text{if } r \leq m-1, \\ m-1 & \text{otherwise.} \end{cases}$$

Observe that the function ψ_m cleans in one step any initially condition that is not coherent for the local clock scheme (for example if some node starts with an initial value in its second coordinate that is greater than its local period). Finally, we present the periodic extension. The main difference between the latter definitions is that updates can be defined arbitrarily over a fixed time interval which is defined through a period parameter p. In addition, we say it is periodic because what happens in this time window must be repeated every p times steps. Thus, we consider an internal clock coordinate (as same as the previous cases) together with a third constant coordinate which serves as a marker for the set of time steps (over each p steps time interval) in which each node is updated.

Definition 4.7 (periodic extension) *Let* $F : Q^V \to Q^V$ *be an automata network. Let* $p \in \mathbb{N}$ *and let* $Q' = Q \times \{0, \ldots, p-1\} \times 2^{\{0,\ldots,p-1\}}$*. We define the* periodic extension *of* F *with period length* p *as the automata network* $F' : (Q')^V \to (Q')^V$ *such that for all* $x = (x_Q, x_p, x_s) \in Q'^V$ *and all* $v \in V$*:*

$$F'(x)_v = \begin{cases} (F(x_Q)_v, (x_p)_v + 1 \mod p, (x_s)_v) & \text{if } (x_p)_v \in (x_s)_v, \\ ((x_Q)_v, (x_p)_v + 1 \mod p, (x_s)_v) & \text{otherwise.} \end{cases}$$

This approach by asynchronous extensions can also capture non-periodic update schemes. For instance, an update scheme for Boolean networks called firing memory has been recently studied [32, 39, 67]. This update scheme, uses internal clocks at each node and, in addition, it makes the delay mechanism given by these clocks depend on the state of each node in the current configuration. Roughly speaking, each node is updated at the same time but it has an internal clock which works in a similar way as local clocks update scheme, with the exception that, this clock depends on the current state of each node in the following way: if certain node is in state 0 it will update its state following the original local rule (in fact, nodes in state 0 are forced to have their internal clocks in 0). Complementarily, if a node is in state 1 and it is intended to change its state to 0 (for example in a conjunctive network this may happen if some other neighbor is in state 0) it will verify its internal clock. If its clock has not reach 0 then, it continues to the next value (it decreases its value) and its current state value is not updated (it remains in state 1). However, in any case (if node is in state 1 or in state 0), if node is intended to change its state to 1 then, its internal clock is restored to its maximum value independently of its current value (see Figure 4.5 for a concrete example).

Formally, given a boolean network $F : \{0,1\}^n \mapsto \{0,1\}^n$ it is possible to define a new state space by associating to each coordinate i of a configuration $x \in \{0,1\}^n$ an internal clock, given by some integer $y_i \in \{0, \ldots, \tau_i\}$ for some integers $\tau = (\tau_1, \ldots, \tau_n)$ such that $\tau_i \geq 1$. In

this product space, we will define the following dynamics: First, for each $(x_\mathrm{i}, y_\mathrm{i})$, we compute, by using the original rule, the amount $x_\mathrm{i}^{'} = F(x)_\mathrm{i}$. Thus, the state $(x_\mathrm{i}, y_\mathrm{i})$ will be updated to $(x_\mathrm{i}^*, y_\mathrm{i}^*)$ according to the following rules:

$$x_\mathrm{i}^* = \begin{cases} x_\mathrm{i}^{'} & \text{if } y_\mathrm{i} < 2, \\ 1 & \text{if } y_\mathrm{i} \geq 2, \end{cases}$$

and,

$$y_\mathrm{i}^* = \begin{cases} 0 & \text{if } x_\mathrm{i}^{'} = 0 \wedge y_\mathrm{i} < 2, \\ y_\mathrm{i} - 1 & \text{if } x_\mathrm{i}^{'} = 0 \wedge y_\mathrm{i} \geq 2, \\ \tau_\mathrm{i}. & \text{if } x_\mathrm{i}^{'} = 1. \end{cases}$$

Firing memory schemes can be captured as an asynchronous extension in such a way that the resulting asynchronous extension is also a CSAN (see Lemma 4.9). We give hereunder the precise definition of a firing memory extension:

Definition 4.8 (Firing memory extension) *Let $Q = \{0, 1\}$ and let $F : Q^V \to Q^V$ be an automata network. Let $\tau \in \mathbb{N}$ and let $Q' = Q \times \{0, \dots, \tau\} \times \{0, \dots, \tau\}$. We define a firing memory extension of F with maximum delay τ as the automata network $F' : (Q')^V \to (Q')^V$ such that for all $x = (x_Q, x_\tau, x_m) \in Q'^n$ we have that*

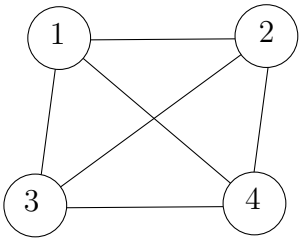
$$F'(x)_v = \begin{cases} (1, (x_m)_v, (x_m)_v) & \text{if } (x_\tau)_v > 1 \wedge f_v(x_Q|_{N(v)}) = 1, \\ (1, (\psi_{(x_m)_v}[(x_c)_v]) - 1, (x_m)_v) & \text{if } (x_\tau)_v > 1 \wedge f_v(x_Q|_{N(v)}) = 0, \\ (1, (x_m)_v, (x_m)_v) & \text{if } (x_\tau)_v \leq 1 \wedge f_v(x_Q|_{N(\mathrm{i})}) = 1, \\ (0, \max\{(\psi_{(x_m)_v}[(x_c)_v]) - 1, 0\}, (x_m)_v) & \text{if } (x_\tau)_v \leq 1 \wedge f_v(x_Q|_{N(v)}) = 0, \end{cases}$$

where $f_v : Q^{N(v)} \to Q$ is such that $F'_v \equiv f_v$ and $\psi_m(r) : \{0, \dots, \tau\} \to \{0, \dots, \tau\}$ is such that $\psi_m[r] = \begin{cases} r & r \leq m - 1 \\ m - 1 & \textit{otherwise} \end{cases}$

The previous definitions are formalized for every automata network. We now focus on CSAN families where the extensions are also CSAN as show in the following lemma.

Lemma 4.9 *Let F be a CSAN, then any block sequential extension (resp. local clocks extension, resp. periodic extension, resp. firing memory extension) of F is a CSAN. Moreover, for any CSAN family $\mathcal{F}$ and any fixed b (resp. c, resp. p, resp τ), the set of block sequential extensions (resp. local clock extensions, resp. periodic, resp. firing memory extenstions) with b blocks (resp. with clock delay c, resp. with period p, resp. with maximum delay) of networks of $\mathcal{F}$ is again a CSAN family.*

Proof. Observe first that for block sequential extensions (resp. local clocks extension, resp. periodic extension, resp. firing memory extension) the definition of F' considers an alphabet $Q' = Q \times R$ such that the action of F' on the R component is purely local (the new value of the R component of a node evolves as a function of the old value of this R component)



$$
\begin{array}{rcl}
t = 0 & \longrightarrow & (0,1,2,3) \\
t = 1 & \longrightarrow & (3,0,1,2) \\
t = 2 & \longrightarrow & (2,3,0,1) \\
t = 3 & \longrightarrow & (1,2,3,0) \\
t = 4 & \longrightarrow & (0,1,2,3)
\end{array}
$$

Figure 4.5: A firing memory update scheme over a conjunctive network. Each local function is given by the minimum over the states of neighbors. The delay component (second component in Definition 4.8) only is shown. Maximum delay of the network is $\tau = 4$. Dynamics describes an attractor of period 4 in which 0 circulates over the different nodes of the network. Note that at any time, there is exactly one node in state 0, the other being in state 1 with different delay values.

and the value of the R component at a node determines alone if the Q component should be updated according to F or left unchanged. Therefore clearly F' is a CSAN if F is.

In the context of a CSAN family $\mathcal{F}$, the CSAN definition of F' involves only local constraints coming from $F \in \mathcal{F}$ and the action on the R-component is the same at each node. So the second part of the lemma is clear. □

To sum up, our formalism allows to treat variations in the update scheme as a change in the CSAN family considered. Given a CSAN family $\mathcal{F}$ and integers b, c, p, τ, we introduce the following notations:

- $\mathcal{F}^{\text{BLOCK},b}$ is the CSAN family of all block sequential extensions of networks from $\mathcal{F}$ with b blocks,
- $\mathcal{F}^{\text{CLOCK},c}$ is the CSAN family of all local clocks sequential extensions of networks from $\mathcal{F}$ with clock delay c,
- $\mathcal{F}^{\text{PER},p}$ is the CSAN family of all periodic extensions of networks from $\mathcal{F}$ with period p.
- $\mathcal{F}^{\text{FIR},\tau}$ is the CSAN family of all firing memory extensions of networks from $\mathcal{F}$ with maximum delay τ.

Chapter 5

Concrete symmetric automata networks: a case study

In this chapter, we focus on studying concrete symmetric automata network (CSAN) families. We use previous theoretical framework on complexity of automata networks families in order to classify different CSAN families according to their dynamical behavior under different update schemes. More precisely, we focus on two different families of CSAN: signed conjunctive networks and min-max networks. We distinguish three main subfamilies inside signed conjunctive networks:

- symmetric conjunctive networks, which are regular conjunctive networks (in which all edge labels are the identity function);
- locally positive signed conjunctive networks, in which we allow negative edges (labeled by the switch function: $\text{Switch}(x) = 1 - x$) but with a local constraint forcing the existence of at least one positive edge in each neighborhood (one edge labeled by the identity function); and
- general signed conjunctive networks, in which there could be negative edges without any constraint (possibly all edges can be negative).

In addition, we consider the latter presented four update schemes: block sequential, local clocks, general periodic update scheme and firing memory. We classify previous families according to their dynamical behavior and simulation capabilities by using the framework presented in Chapters 3 and 4. Before we enter into the detail, we present in Table 5.1 a summary of the main results obtained. Observe that in each row of Table 5.1, we show how the dynamical behavior of some CSAN families changes as we change the update scheme. In particular, the most simple ones, such as conjunctive and locally positive networks exhibit a relatively simple dynamical behavior (they have bounded period attractors). Contrarily, the last two families have strong universality even for block sequential update schemes. In addition, we would like to remark that there is not only a hierarchy for update schemes (block sequential update schemes are a particular case of local clocks and both are a particular cases of periodic update schemes) but that network families are also somehow related as conjunctive networks are a sub-family of all the other ones. As a consequence of this, we have the results in the latter column for firing memory as a consequence of the result on universality for

Family/Update scheme	Block sequential	Local clocks	General periodic	Firing memory
Conjunctive networks	BPA	BPA	**SPA**	**SU**
Locally positive	BPA	**SU**	SU	SU
Signed conjunctive networks	**SU**	SU	SU	SU
Min-max networks	**SU**	SU	SU	SU

Table 5.1: Summary of the main results on complexity of the dynamics of the network families studied in the current chapter, depending on different update schemes. BPA = Bounded period attractors. SPA = Superpolynomial attractors. SU = Strong universality. Black fonts indicate the emergence of complex behavior such as long period attractors or universality.

conjunctive networks with firing memory.

Additionally, one can also observe that there is some sort of "diagonal emergence" of strong universality in Table 5.1 consisting in the fact that it seems to exists a trade-off between the complexity in the definition of some network families and the complexity of the corresponding update schemes. In other words, simple families seem to need more complex update schemes in order to be universal and as we pass to more complex rules one can observe this property for simpler update schemes.

5.1 Symmetric conjunctive networks

As said previously, symmetric conjunctive networks form a particular CSAN family on alphabet $\{0,1\}$ where all edges are labeled by the identity map and all nodes have the same 'conjunctive' local map

$$\lambda(q,X) = \begin{cases} 0 & \text{if } 0 \in X, \\ 1 & \text{else.} \end{cases}$$

First, observe that conjunctive networks are a particular case of symmetric threshold networks and thus, it follows from the classical results of [42] that they always converge to some fixed point or limit cycles of length two. Therefore they are not dynamically complex with parallel update schedule.

5.1.1 Local clocks update scheme

In this subsection, we study local clocks extenstions. Let us call $\mathcal{F}^{\text{CLOCK},c}_{\text{SYM-CONJ}}$ the familly of conjunctive networks under local clocks schemes of clock period c for some fixed $c > 0$. We recall that in this case, the original alphabet $Q = \{0,1\}$ of symmetric conjunctive networks is modified by adding a component which manages the clocks mechanism i.e. the alphabet has the form: $Q_c = Q \times \{0,\dots,c\} \times \{0,\dots,c\}$ for some parameter c. We show in this section that, interestingly, local clocks update schemes on conjunctive networks are not able to produce superpolynomial cycles and only have attractors of bounded period.

Lemma 5.1 *Let us fix any $c > 0$ and consider the family $\mathcal{F}^{\text{CLOCK},c}_{\text{SYM-CONJ}}$ of all symmetric conjunctive networks under local clocks update scheme with clock period c. Fix $n > 0$ and let F :*

$Q_c^n \to Q_c^n \in \mathcal{F}^{\text{CLOCK},c}_{\text{SYM-CONJ}}$. *For any configuration* $(x^Q, x^c, x^m) \in Q_c^n$ *we have that the period of the attractor reached from* (x^Q, x^c, x^m) *is at most* $2\,\mathrm{lcm}\{x_v^m : v \in \{1, \dots, n\}\}$. *Moreover, for each attractor* $(\overline{x}^Q, \overline{x}^c, \overline{x}^m) \in Att(F)$, *the set* $S(\overline{x}^Q, \overline{x}^c, \overline{x}^m) = \{v \in V : \pi(F((\overline{x}^Q, \overline{x}^c, \overline{x}^m)))_v \neq \overline{x}_v^Q\}$ *where* $\pi : Q_c^n \mapsto Q$ *is the projection on the alphabet* Q, *induces a bipartite subgraph.*

Proof. Let $(x^Q, x^c, x^m) \in Q_c^n$ be a configuration and $(\overline{x}^Q, \overline{x}^c, \overline{x}^m) \in \mathrm{Att}(F)$ be an attractor that is reachable from (x^Q, x^c, x^m) and that is not a fixed point, i.e. $p(\overline{x}) \geq 2$. In order to simplify the notation, we are going to denote $\overline{x}^Q$ by $\overline{x}$. Let $\mathrm{i} \in \{0, \dots, n-1\}$ be a coordinate such that $\overline{x}_\mathrm{i}$ changes its state, i.e there exists some $t \in \mathbb{N}$ such that $\overline{x}(0)_\mathrm{i} \neq \overline{x}(t)_\mathrm{i}$. Without loss of generality we assume that $\overline{x}(0)_\mathrm{i} = 1$ and $\overline{x}^c(0)_\mathrm{i} = 0$ and $\overline{x}(1)_\mathrm{i} = 0$. Consider t_1 as the first time step such that the coordinate i changes its state from 0 to 1, which means that, t_1 is the first time step such that $\overline{x}(t_1) = 0$ and $\overline{x}(t_1 + 1) = 1$. Observe that $t_1 = sx_\mathrm{i}^m$ for some $s \geq 1$. Note that, for all $j \in N(\mathrm{i})$, $\overline{x}(t_1)_j = 1$. Moreover, we have that for all $j \in N(\mathrm{i})$: and that $\overline{x}(s)_j = 1$ for all $s \in [1, t_1]$ (see Figure 5.1.1). This is because, since the interaction graph is symmetric, j cannot be in state 0 during interval $[1, t_1]$ otherwise both i and j would stay in state 0 forever, thus contradicting the hypothesis on i. We deduce that $x_j^m \geq t_1 = sx_\mathrm{i}^m$ (otherwise j would update its state and become 0 on interval $[1, t_1]$). Now consider t_0 to be the first time step in which the node i changes its state from 1 to 0, i.e. $\overline{x}(t_0) = 1$ and $\overline{x}(t_0 + 1) = 0$. Observe that, $t_1 < t_0 = t_1 + s^* x_\mathrm{i}^m$ for some $s^* \geq 1$. In addition, there must exist some neighbor $k \in N(\mathrm{i})$ satisfying that $\overline{x}(t_0)_k = 0$, otherwise i cannot change to 0 (it requires at least one neighbor in 0 in order to change its state from 1 to 0). Observe that node k satisfies $x_k^m \leq x_\mathrm{i}^m$ because it needs to update to 0 before node i. More precisely, by the definition of t_0 we have that i is fixed in state between t_1 and t_0 (see Figure 5.1.1). Additionally, we have that $\overline{x}_k(t_1) = 1$, $\overline{x}_k(t_0) = 0$ and also we have that $\overline{x}(t_0 - x_\mathrm{i}^m)_k = 1$ (otherwise it contradicts the minimality of t_0). Finally, since i remains in state 0 on the interval $[t_0, t_0 + x_\mathrm{i}^m]$ then, k cannot be updated in the same interval. Thus, $x_k^m \leq x_\mathrm{i}^m$. Moreover, the latter observations imply that i and k are synchronized, i.e. $(\overline{x}^c(0))_k = (\overline{x}^c(0))_\mathrm{i}$, $x_k^m = x_\mathrm{i}^m$, $t_1 = x_\mathrm{i}^m$ and $t_0 = t_1 + x_\mathrm{i}^m$. Note that also, we have that for all t, $\overline{x}(t)_\mathrm{i} = 1 - \overline{x}(t)_k$. We have shown that the period of any node v is at most $2x_v^m$, so we deduce $p(\overline{x}) \leq 2\,\mathrm{lcm}\{x_v^m : v \in \{1, \dots, n\}\}$.

At this point, we know that i must have at least one neighbor that is not constant in $\overline{x}$ and that it is synchronized. Let us assume that there is a non constant neighbor ℓ of i that satisfies $x_\ell^m > x_\mathrm{i}^m$. On the other hand, we have that ℓ is in state 1 on the interval $[0, t_1]$ (see Figure 5.1.1) because otherwise i cannot switch to state 1 at the time step t_1. Observe that, by hypothesis, ℓ cannot change its state on intervals of the form $[rx_\mathrm{i}^m, (r+1)x_\mathrm{i}^m)$ for $r \in \mathbb{N}$ even since i is in state 0 on those intervals (otherwise i cannot switch back to 1 because it would have a neighbor in 0). However, for r even, i is in state 1 on intervals of the form $[rx_\mathrm{i}^m, (r+1)x_\mathrm{i}^m)$. Suppose that ℓ changes its value for the first time on an interval of the form $[r^* x_\mathrm{i}^m, (r^*+1)x_\mathrm{i}^m)$ for some $r^* \in \mathbb{N}$ odd, i.e. $\overline{x}(s)_\ell = 0$ for some $s \in [r^* x_\mathrm{i}^m, (r^*+1)x_\mathrm{i}^m)$. Observe now that $\overline{x}((r^* + 2)x_\mathrm{i}^m) = 1$ since i must return to state 1 but ℓ cannot change its state in $[(r^*+1)x_\mathrm{i}^m, (r^*+2)x_\mathrm{i}^m)$ because i is in state 0. Then, we must have $x_\ell^m \leq x_\mathrm{i}^m$, which contradicts the hypothesis. We conclude that every non constant neighbor of i is synchronized. Repeating the same argument now for any non constant neighbor of i we have that all the nodes in the connected component containing i have local delay x_i^m. Iterating this same technique now for each i in the network, we deduce that $\overline{x}$ is such that $p(\overline{x}) \leq 2\,\mathrm{lcm}\{x_v^m : v \in \{1, \dots, n\}\}$ since locally, each connected component containing some node i is synchronized and thus,

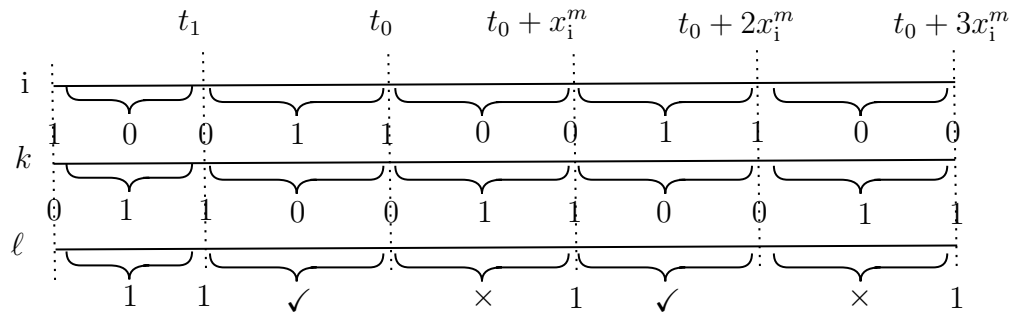


Figure 5.1: Scheme of the dynamics of nodes i, k and ℓ defined in the proof of Lemma 5.1. The checkmarks indicate where it is feasible for ℓ to be updated and the crosses mark the intervals on which ℓ can change its state.

each non-constant node is switching its state every $2x_i^m$ time steps. In addition, for each node i every non-constant neighbor is in the state 0 whenever i is in the state 1. Thus, the set of nodes which are not constant for $(\overline{x}^Q, \overline{x}^c, \overline{x}^m)$ i.e. $S(\overline{x}^Q, \overline{x}^c, \overline{x}^m)$, induces a two colorable subgraph. The result holds. □

As seen above, there is a qualitative jump between local clocks and periodic update schedules for conjunctive networks in the size of dynamical cycles. However, for transient and prediction problems, even general periodic update schedules fail to produce maximal complexity (under standard complexity classes separations assumptions).

Theorem 5.2 *Let $p \in \mathbb{N}$ and consider the family $\mathcal{F}^{\text{PER},p}_{\text{SYM-CONJ}}$ of conjunctive networks under periodic update schedules of period p. $\mathcal{F}^{\text{PER},p}_{\text{SYM-CONJ}}$ is neither dynamically nor computationally complex: more precisely, the transients of any network in $\mathcal{F}^{\text{PER},p}_{\text{SYM-CONJ}}$ with n nodes are of length at most $O(n^2)$, the problem* $\text{PREDU}_{\mathcal{F}}$ *can be solved by an $\mathbf{NC}^2$ algorithm and* $\text{PREDB}_{\mathcal{F}}$ *can be solved in polynomial time.*

Proof. Let $F \in \mathcal{F}^{\text{PER},p}_{\text{SYM-CONJ}}$with n nodes and consider any initial configuration x. By definition, the orbit of x under F^p is constant on the second and third component of states. Moreover, the action of F^p on the first component when starting from x is a particular non-symmetric conjunctive network F_x that can be seen as an arbitrary Boolean matrix M_x. First, by [19, Theorem 3.20], the transient of the orbit of x under F_x is of length at most $2n^2 - 3n + 2$. We deduce that the transient of x under F is in $O(n^2)$. Second, it is easy to compute M_x from F and x in $\mathbf{NC}^1$. Moreover, matrix multiplication can be done in $\mathbf{NC}^1$ and by fast exponentiation circuits we can compute M_x^t with polynomial circuits of depth $O(\log(t)\log(n))$. With a constant computational overhead, we can therefore efficiently compute $F^t(x)_v$ and the complexity upper bounds on $\text{PREDU}_{\mathcal{F}}$ and $\text{PREDB}_{\mathcal{F}}$ follow. □

5.1.2 Periodic update schemes

In this subsection, we show that symmetric conjunctive networks with a periodic update schedule of period 3 can break the latter limitation on attractor period and produce superpolynomial cycles. Observe that all the graphs over which we are defining networks in

this family (and also in the other concrete examples we will be exploring in next sections) are non-directed (symmetric). If we observe carefully the effect of periodic update schemes, we note that we are actually changing the interaction graph of the network by considering different orders for updating nodes and thus, breaking the symmetry in the different connections that nodes have in the network. Moreover, we corroborate this remark by showing that actually, we can simulate arbitrary conjunctive networks by using a periodic update scheme. We accomplish this by applying our formalism on simulation and gadget glueing. In fact, we show that the family of symmetric conjunctive networks admits coherent $\mathcal{G}_{\text{conj}}$-gadgets. Thus it is capable of simulating the familly $(\mathcal{F}_{\text{conj}}, \mathcal{F}^*_{\text{conj}})$. Particularly, this implies that this family admits attractors of superpolynomial period.

Proposition 5.3 *Let $p \geq 1$ and $\mathcal{F}^{\text{PER},p}_{\text{SYM-CONJ}}$ be the family of symmetric conjunctive networks under periodic update schedule of period p. $\mathcal{F}^{\text{PER},p}_{\text{SYM-CONJ}}$ is not universal and therefore it does not admit coherent $\mathcal{G}_m$-gadgets.*

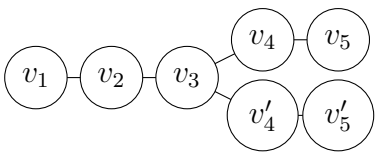
Proof. We actually show that the longest transient of any $F \in \mathcal{F}^{\text{PER},p}_{\text{SYM-CONJ}}$ with n nodes is $O(n^2)$, then the conclusions follow by Corollary 2.19 and Theorem 3.16. Recall that the alphabet is $\{0,1\} \times \{0,\dots,p\} \times 2^{\{0,\dots,p\}}$ and that, by definition, on any given configuration x the component $\{0,\dots,p\} \times 2^{\{0,\dots,p\}}$ is periodic of period p independently of the behavior on the first $\{0,1\}$ component. Moreover, the behavior on the $\{0,1\}$ component is that of a fixed (non-symmetric) conjunctive network F' in the following sense:

$$F^{t+p}(x) = F'(F^t(x)), \forall t \geq 0.$$

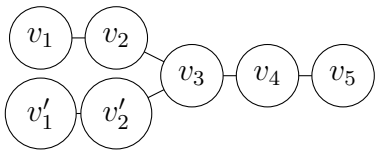
By [19, Theorem 3.20], the transient of any orbit of F' is $O(n^2)$. We deduce that the transient of the orbit of x under F is also $O(n^2)$. □

Theorem 5.4 *Let $\mathcal{F}^{\text{PER},3}_{\text{SYM-CONJ}}$ be the family of symmetric conjunctive networks under periodic update schedules of period 3. $\mathcal{F}^{\text{PER},3}_{\text{SYM-CONJ}}$ has coherent $\mathcal{G}_{conj}$-gadgets and therefore simulates the family of conjunctive networks $(\mathcal{F}_{conj}, \mathcal{F}^*_{conj})$. In particular, it can produce attractors of superpolynomial period.*

Proof. We start by showing that $\mathcal{F}^{\text{PER},3}_{\text{SYM-CONJ}}$ has the coherent $\mathcal{G}_{conj}$-gadgets. We use the notations from Definition 3.9. Let $F_{\text{COPY}} \in \mathcal{F}^{\text{PER},3}_{\text{SYM-CONJ}}$ be defined on the following graph with nodes $V_{\text{COPY}} = \{v_1, v_2, v_3, v_4, v_5, v'_4, v'_5\}$:



Let also $F_{\text{AND}} \in \mathcal{F}^{\text{PER},3}_{\text{SYM-CONJ}}$ be defined on the following graph with nodes $V_{\text{AND}} = \{v_1, v_2, v'_1, v'_2, v_3, v_4, v_5\}$:



Recall that both F_{COPY} and F_{AND} have alphabet $Q = \{0,1\} \times \{0,1,2\} \times 2^{\{0,1,2\}}$. Now let $C = C_{\mathrm{i}} \cup C_o$ be the glueing interface with $C_{\mathrm{i}} = \{\mathrm{i}\}$ and $C_o = \{o\}$. F_{COPY} is seen as a gadget with one input and two outputs for the gate COPY $\in \mathcal{G}_{conj}$ as follows:

- $\phi^{\mathrm{i}}_{\mathrm{COPY},1}(\mathrm{i}) = v_2$ and $\phi^{\mathrm{i}}_{\mathrm{COPY},1}(o) = v_1$;
- $\phi^{o}_{\mathrm{COPY},1}(\mathrm{i}) = v_5$ and $\phi^{o}_{\mathrm{COPY},1}(o) = v_4$.
- $\phi^{o}_{\mathrm{COPY},2}(\mathrm{i}) = v'_5$ and $\phi^{o}_{\mathrm{COPY},2}(o) = v'_4$.

F_{AND} is seen as a gadget with two inputs and one output for the gate AND $\in \mathcal{G}_{conj}$ as follows:

- $\phi^{\mathrm{i}}_{\mathrm{AND},1}(\mathrm{i}) = v_2$ and $\phi^{\mathrm{i}}_{\mathrm{AND},1}(o) = v_1$;
- $\phi^{\mathrm{i}}_{\mathrm{AND},2}(\mathrm{i}) = v'_2$ and $\phi^{\mathrm{i}}_{\mathrm{AND},2}(o) = v'_1$.
- $\phi^{o}_{\mathrm{AND},1}(\mathrm{i}) = v_5$ and $\phi^{o}_{\mathrm{AND},1}(o) = v_4$.

Note in particular that the conditions of Lemma 3.8 are satisfied so the closure by gadget glueing of these gadgets stays in our family $\mathcal{F}$. Indeed F_{COPY} has the same induced label graphs on all images of C under ϕ maps, and the neighborhood of $\phi^{\mathrm{i}}_{\mathrm{COPY},1}(C_o) = v_1$ is v_2 which belongs to $\phi^{\mathrm{i}}_{\mathrm{COPY},1}(C)$, and similarly for other ϕ maps. Corresponding properties hold also for F_{AND}. We now define the following elements required by Definition 3.9:

- The two state configurations s_q for $q \in \{0,1\}$ are defined by $s_q(\mathrm{i}) = (q, 0, \{0,2\})$ and $s_q(o) = (1, 0, \{0,1\})$.
- The context configuration is defined by $c(v_3) = (1, 0, \{1,2\})$ both for F_{COPY} and F_{AND}.
- The time constant is $T = 3$.
- The standard trace $\tau_{q,q'}$ over the glueing interface from $q \in \{0,1\}$ to $q' \in \{0,1\}$ is given by:

time	i	o
0	$(q, 0, \{0,2\})$	$(1, 0, \{0,1\})$
1	$(1, 1, \{0,2\})$	$(q, 1, \{0,1\})$
2	$(1, 2, \{0,2\})$	$(1, 2, \{0,1\})$
3	$(q', 0, \{0,2\})$	$(1, 0, \{0,1\})$

.

- For any $q_{\mathrm{i}}, q_{\mathrm{i}}^T, q_o, q_{o'} \in \{0,1\}$ we have the following $\{v_1, v_5, v'_5\}$-pseudo orbit for F_{COPY}:

time	v_1	v_2	v_3	v_4	v_5	v'_4	v'_5
0	$(q_{\mathrm{i}}, 0, \{0,2\})$	$(1, 0, \{0,1\})$	$(1, 0, \{1,2\})$	$(q_o, 0, \{0,2\})$	$(1, 0, \{0,1\})$	$(q'_o, 0, \{0,2\})$	$(1, 0, \{0,1\})$
1	$(1, 1, \{0,2\})$	$(q_{\mathrm{i}}, 1, \{0,1\})$	$(1, 1, \{1,2\})$	$(1, 1, \{0,2\})$	$(q_o, 1, \{0,1\})$	$(1, 1, \{0,2\})$	$(q'_o, 1, \{0,1\})$
2	$(1, 2, \{0,2\})$	$(1, 2, \{0,1\})$	$(q_{\mathrm{i}}, 2, \{1,2\})$	$(1, 2, \{0,2\})$	$(1, 2, \{0,1\})$	$(1, 2, \{0,2\})$	$(1, 2, \{0,1\})$
3	$(q_{\mathrm{i}}^T, 0, \{0,2\})$	$(1, 0, \{0,1\})$	$(1, 0, \{1,2\})$	$(q_{\mathrm{i}}, 0, \{0,2\})$	$(1, 0, \{0,1\})$	$(q_{\mathrm{i}}, 0, \{0,2\})$	$(1, 0, \{0,1\})$

.

- For any $q_{\mathrm{i}}, q_{\mathrm{i}}^T, q_{\mathrm{i}'}, q_{\mathrm{i}'}^T, q_o \in \{0,1\}$ we have the following $\{v_1, v'_1, v_5\}$-pseudo orbit for F_{AND}:

time	v_1	v_2	v'_1	v'_2	v_3	v_4	v_5
0	$(q_{\mathrm{i}}, 0, \{0,2\})$	$(1, 0, \{0,1\})$	$(q_{\mathrm{i}'}, 0, \{0,2\})$	$(1, 0, \{0,1\})$	$(1, 0, \{1,2\})$	$(q_o, 0, \{0,2\})$	$(1, 0, \{0,1\})$
1	$(1, 1, \{0,2\})$	$(q_{\mathrm{i}}, 1, \{0,1\})$	$(1, 1, \{0,2\})$	$(q_{\mathrm{i}'}, 1, \{0,1\})$	$(1, 1, \{1,2\})$	$(1, 1, \{0,2\})$	$(q_o, 1, \{0,1\})$.
2	$(1, 2, \{0,2\})$	$(1, 2, \{0,1\})$	$(1, 2, \{0,2\})$	$(1, 2, \{0,1\})$	$(q_{\mathrm{i}} \wedge q_{\mathrm{i}'}, 2, \{1,2\})$	$(1, 2, \{0,2\})$	$(1, 2, \{0,1\})$
3	$(q_{\mathrm{i}}^T, 0, \{0,2\})$	$(1, 0, \{0,1\})$	$(q_{\mathrm{i}'}^T, 0, \{0,2\})$	$(1, 0, \{0,1\})$	$(1, 0, \{1,2\})$	$(q_{\mathrm{i}} \wedge q_{\mathrm{i}'}, 0, \{0,2\})$	$(1, 0, \{0,1\})$

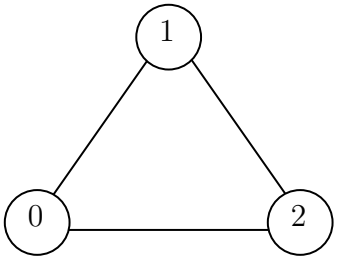
The conclusion about simulations of conjunctive networks and superpolynomial cycles follows from previous results as soon as we prove that the family has coherent $\mathcal{G}_{conj}$-gadgets: Theorem 3.23 for simulation of conjunctive networks and Proposition 3.19 to show that the family has coherent $\mathcal{G}_w$-gadgets (because the identity map is obtained by composition of COPY and AND) and then Theorem 3.21 for super-polynomial cycles. □

5.1.3 Firing memory schemes

In this section, we study conjunctive networks under the firing memory update scheme. As we pointed out in Chapter 4, this latter scheme was firstly introduced in [32] as a way to implement decays in the dynamics of Boolean networks and as a form of extension of discrete models in the context of applications in biology. Essentially, it is based on the concept of a non-symmetric delay or memory, which consists on each node having an internal clock which measures the time since which its state has changed from state 0 to 1 and fixes a minimum amount of time steps in which the node will remain in state 1 independently of its local dynamics. Observe that this internal clock keeps track of the transitions from 0 to 1 but not viceversa. As a consequence, we call it a "non-symmetric" delay. If a node is waiting because its state has recently changed from 0 to 1 and and if its local dynamics requests its state to become 1 again (for example, in the case of conjunctive networks all its neighbors may be at state 1) then, it resets its internal clock. Observe that, one can keep track only on the states of the internal clock of each node (that we call the delay state or simply the delay of each node) and deduce the original binary state of each node (if delay is at least 1 then, node is in state 1 and if not it must be in state 0). This way of interpreting states in firing memory coincides with the projection related to the second coordinate when an asynchronous extension for certain family of automata networks is considered. In addition, we remark again that conjunctive networks with firing memory are also a CSAN family as it was remarked in previous chapters.

The main result of this section is based on the results shown in [39] and are adapted in this section to our simulation formalism. We start by showing a gadget that we use for the main result consisting in a small network exhibiting attractors of period 3. In [39] it is shown that an analog reasoning can be used to construct networks exhibiting linear periods (related to the size of the network) and thus, symmetric conjunctive networks with firing memory exhibit non-polynomial period attractors, as well as symmetric conjunctive networks under general periodic update schemes. However, in this case we are able to show that we have far more than that, and that conjunctive networks with firing memory are strongly universal. We choose an alternative approach to the one followed in [39] and we show this latter result by exhibiting coherent $\mathcal{G}_{m,2}$-gadgets. More precisely, we show that conjunctive networks are capable of simulating arbitrary $\mathcal{G}_{m,2}$-networks which are strongly universal. This goes beyond the results presented in [39] as this notion of universality directly implies the existence of exponential period attractors. Finally, remember that the family of conjunctive networks with firing memory is noted by $\mathcal{F}_{\mathrm{CONJ}}^{\mathrm{FIR},\tau}$.

We start by the following proposition that states the existence of a simple gadget that we



Time/Node	0	1	2
0	$(0,0)$	$(1,1)$	$(1,2)$
1	$(1,2)$	$(0,0)$	$(1,1)$
2	$(1,1)$	$(1,2)$	$(0,0)$
3	$(0,0)$	$(1,1)$	$(1,2)$

Figure 5.2: Clock gadget implemented in a conjunctive network with firing memory. (Left panel) The interaction graph of the clock gadget. (Right panel) The dynamics of an attractor of period 3.

call a clock gadget. This network exhibits attractors of period 3 and will be essential to the construction of the gadgets we will use to show the strong universality result.

Proposition 5.5 *Let $\mathcal{F}^{\text{FIR},2}_{\text{SYM-CONJ}}$ be the family of symmetric conjunctive networks with firing memory and delay parameter $\tau = 2$. There exists a network $F_{clock} : (\{0,1\} \times \{0,1,2\})^3 \to (\{0,1\} \times \{0,1,2\})^3 \in \mathcal{F}^{\text{FIR},2}_{\text{SYM-CONJ}}$ which has limit cycles of period 3 such that:*

- $F(((0,0),(1,1),(1,2))) = (((1,2),(0,0),(1,1)))$,
- $F(((1,2),(0,0),(1,1))) = ((1,1),(1,2),(0,0))$,
- $F(((1,1),(1,2),(0,0))) = ((0,0),(1,1),(1,2))$.

Proof. The interaction graph of $F_{\text{clock}} : (\{0,1\} \times \{0,1,2\})^3 \to (\{0,1\} \times \{0,1,2\})^3$ is depicted in Figure 5.2. The desired property is shown in the same figure. □

Now, we are in conditions to show the main result of this section. We are going to show that $\mathcal{F}^{\text{FIR},2}_{\text{SYM-CONJ}}$ has coherent $\mathcal{G}_{m,2}$-gadgets by using the clock gadget described above. From now on, we are going to represent a clock gadget by a triangle and we are going to show only the delay value of the node which is connected to the main structure of the gadget (see Figure 5.4).

Lemma 5.6 *The family $\mathcal{F}^{\text{FIR},2}_{\text{SYM-CONJ}}$ of symmetric conjunctive networks with firing memory and delay parameter $\tau = 2$ has coherent $\mathcal{G}_{m,2}$-gadgets.*

Proof. We define $F_{\text{AND}} : (\{0,1\} \times \{0,1,2\})^{72} \to (\{0,1\} \times \{0,1,2\})^{72}$ and $F_{\text{OR}} : (\{0,1\} \times \{0,1,2\})^{60} \to (\{0,1\} \times \{0,1,2\})^{60}$ as conjunctive networks with firing memory and delay parameter $\tau = 2$ over the graphs shown in Figure 5.4. We are going to detail every element that defines a coherent $\mathcal{G}_{m,2}$ gadget in order to finish the proof:

1. We define the glueing surface $C = C_{\mathrm{i}} \cup C_o$ as $C_{\mathrm{i}} = \{c_{\mathrm{i}}\} \cup \{c_{\mathrm{i},1}, c_{\mathrm{i},2}, c_{\mathrm{i},3}\}$ and $C_o = \{c'_o, c_o\} \cup \{c_{o',1}, c_{o',2}, c_{o',3}, \} \cup \{c_{o,1}, c_{o,2}, c_{o,3}\}$ as shown in Figure 5.3.
2. The corresponding embedded copies of C in each gadget are shown in dashed boxes in Figure 5.4 and the functions $\phi^{\mathrm{i}}_{F_{\mathrm{AND}},k}$, $\phi^{o}_{F_{\mathrm{AND}},k}$, $\phi^{\mathrm{i}}_{F_{\mathrm{OR}},k}$ and $\phi^{o}_{F_{\mathrm{OR}},k}$ for $k = 1, 2$ are defined respecting the same order shown in Figure 5.3.
3. For each $x \in \{0, 1\}$ we define the maps (see Figure 5.4):
   - $s_x(c_{\mathrm{i}}) = 2x$, $(s_x(c_{\mathrm{i},1}), s_x(c_{\mathrm{i},2}), s_x(c_{\mathrm{i},3}) = (1, 0, 2)$;
   - $s_x(c'_o) = 1$, $(s_x(c'_{o,1}), s_x(c'_{o,2}), s_x(c'_{o,3}) = (2, 0, 1)$; and
   - $s_x(c_o) = 2$, $(s_x(c_{o,1}), s_x(c_{o,2}), s_x(c_{o,3}) = (0, 1, 2)$.
4. Context configurations are shown in Figure 5.3 for every node that is not inside a dashed box.
5. Each gadget takes $T = 9$ time steps to compute the simulation of an AND or an OR gate (See Figures 5.4 5.5, 5.6, 5.7, 5.8 and 5.9).
6. A standard trace and pseudo-orbit given by the dynamics shown in Figures 5.4 5.5, 5.6, 5.7, 5.8 and 5.9. We remark that the function $k(x, y)$ in Figure 5.9 is defined by $k(0, 0) = 0, k(1, 0) = k(1, 1) = 2$ and $k(0, 1) = 1$. Then, we have that $k(x, y)$ is equal to $\mathrm{OR}(x, y) = 2(x \vee y)$ with the exception that $\mathrm{OR}(1, 0) = 1$. Thus, since the local rule interprets 2 and 1 as 1 (we recall that we are representing $(1, 2)$ by 2 and $(1, 1)$ by 1) and any node that changes its state to 1 updates its internal clock to 2, the gadget computes $2(x \vee y)$ in the next step of computation, as it is shown in Figure 5.9.

□

As a consequence of Lemma 5.6, we have the following theorem.

Theorem 5.7 *The family $\mathcal{F}^{\mathrm{FIR},2}_{\mathrm{SYM\text{-}CONJ}}$ of symmetric conjunctive networks with firing memory and delay parameter $\tau = 2$ is strongly universal. In particular, it is dynamically and computationally complex.*

Proof. Strong universality is a direct consequence of the capability of $\mathcal{F}^{\mathrm{FIR},2}_{\mathrm{SYM\text{-}CONJ}}$ to simulate the family $\Gamma(\mathcal{G}_{m,2})$. This fact comes directly from the latter lemma, as $\mathcal{F}^{\mathrm{FIR},2}_{\mathrm{SYM\text{-}CONJ}}$ is a CSAN and has coherent $\mathcal{G}_{m,2}$-gadgets. Thus, it simulates $\Gamma(\mathcal{G}_{m,2})$ in constant time and linear space (see Corollary 3.12). Strong universality comes from the fact that $\Gamma(\mathcal{G}_{m,2})$ is strongly universal (see Theorem 3.16). Finally, the complexity results (both dynamically and computationally) are a consequence of Corollary 2.14 and Theorem 2.18. □

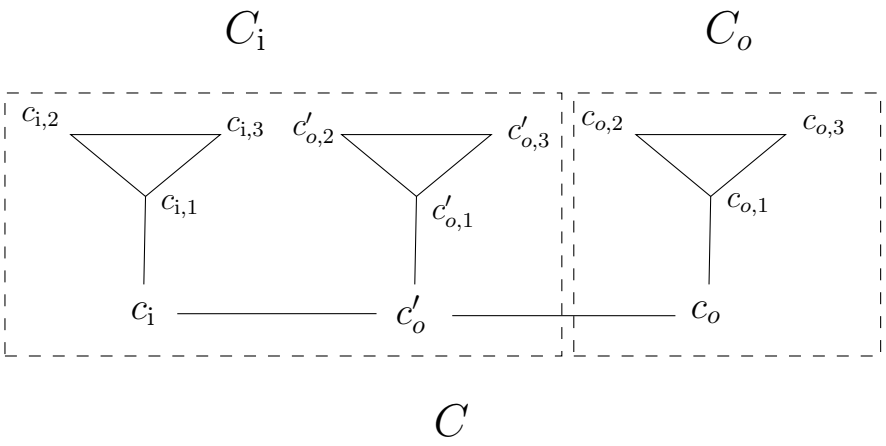


Figure 5.3: The glueing interface considered for AND/OR gadgets implemented over a conjunctive network with firing memory. The labels given by marking functions φ are assigned in each gadget accordingly.

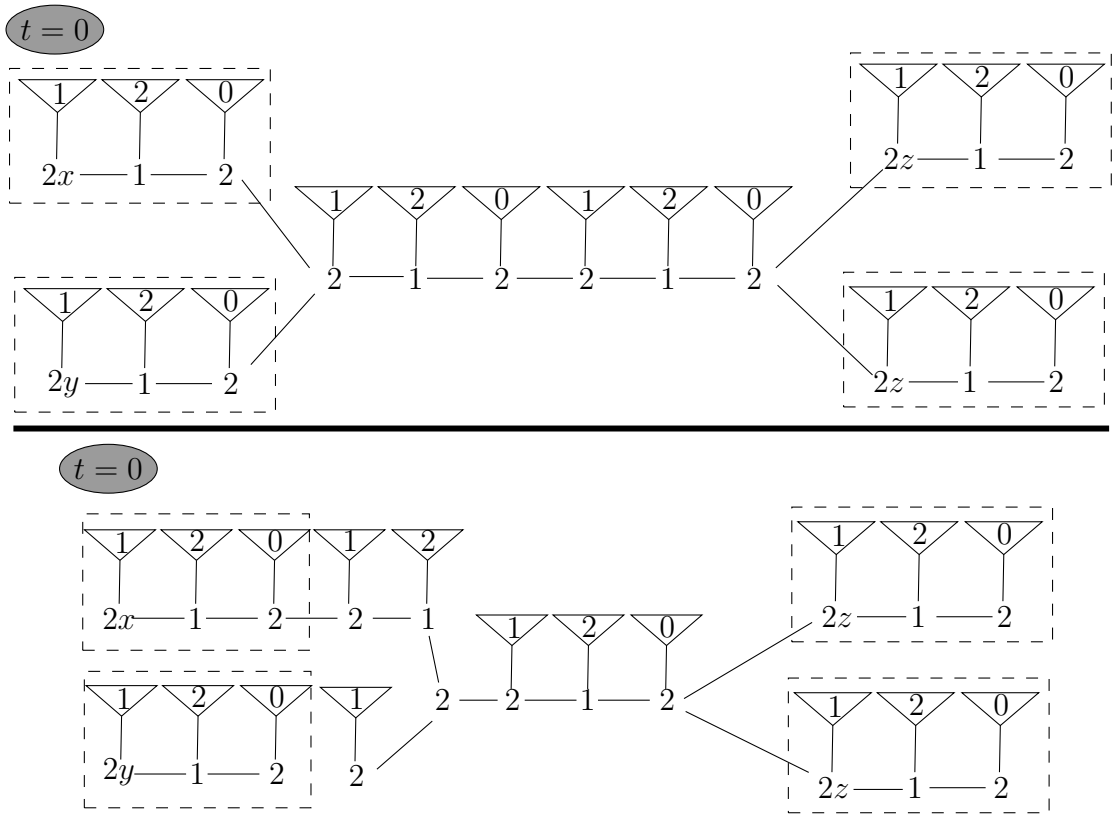


Figure 5.4: The initial condition and structure for AND/OR gadgets implemented over conjunctive networks with firing memory. (Upper panel) the AND gadget. (Bottom panel) the OR gadget. The variables $(x, y) \in \{0, 1\}^2$ represent the bits that the gadget is considering as inputs and $z \in \{0, 1\}$ is a bit that is going to serve as an input for other gadget. Total computation takes $T = 9$ time steps. The triangles represent clock gadgets. The dashed boxes mark the embedded copies of the glueing interface which plays the role of output/input.

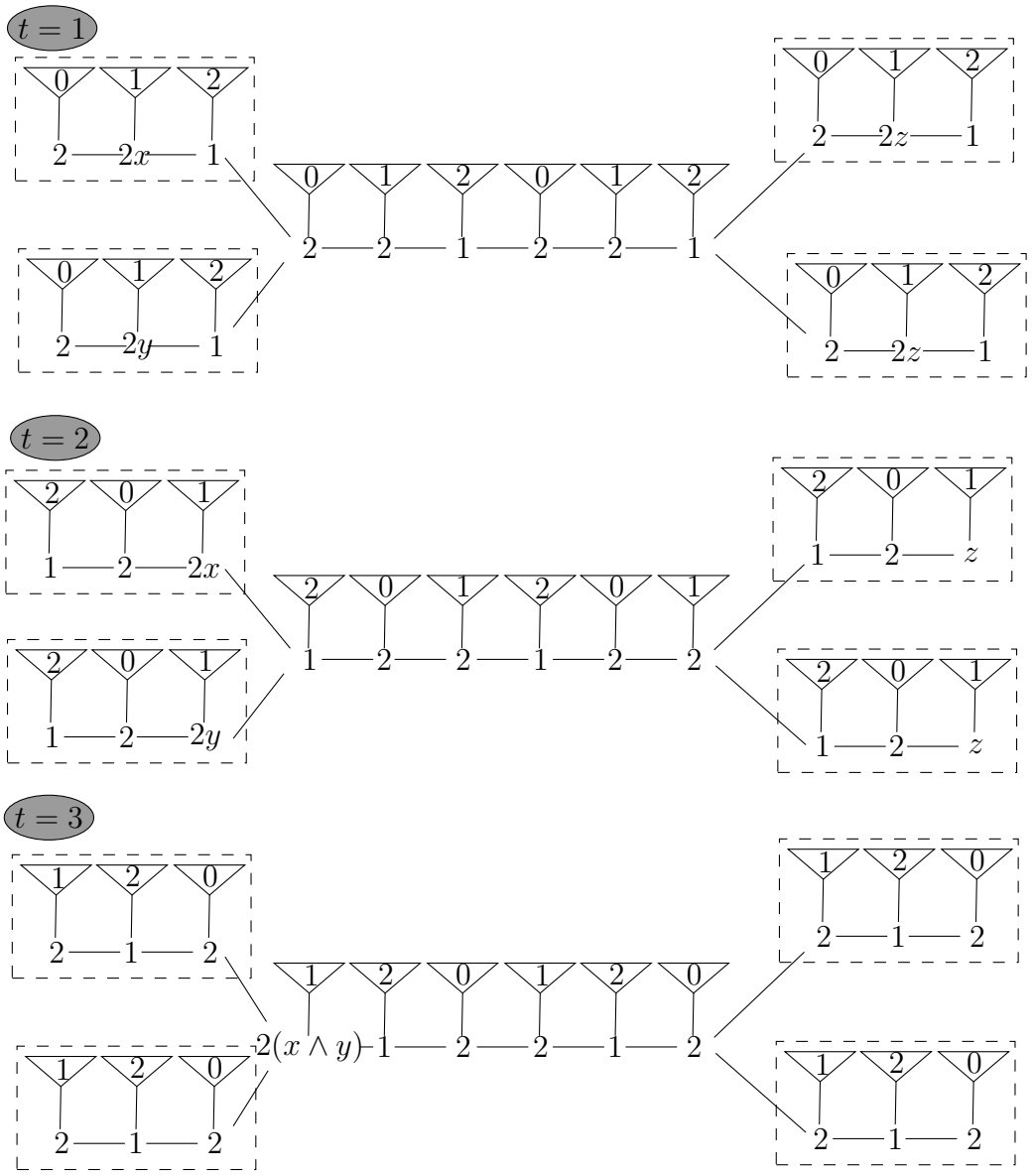


Figure 5.5: The first three steps of the dynamics of the AND gadget implemented in a conjunctive network with firing memory. The variables $(x, y) \in \{0, 1\}^2$ represent the bits that the gadget is considering as inputs and $z \in \{0, 1\}$ is a bit that is going to serve as an input for other gadget. Total computation takes $T = 9$ time steps. The triangles represent clock gadgets. The dashed boxes mark the embedded copies of the glueing interface which plays the role of output/input.

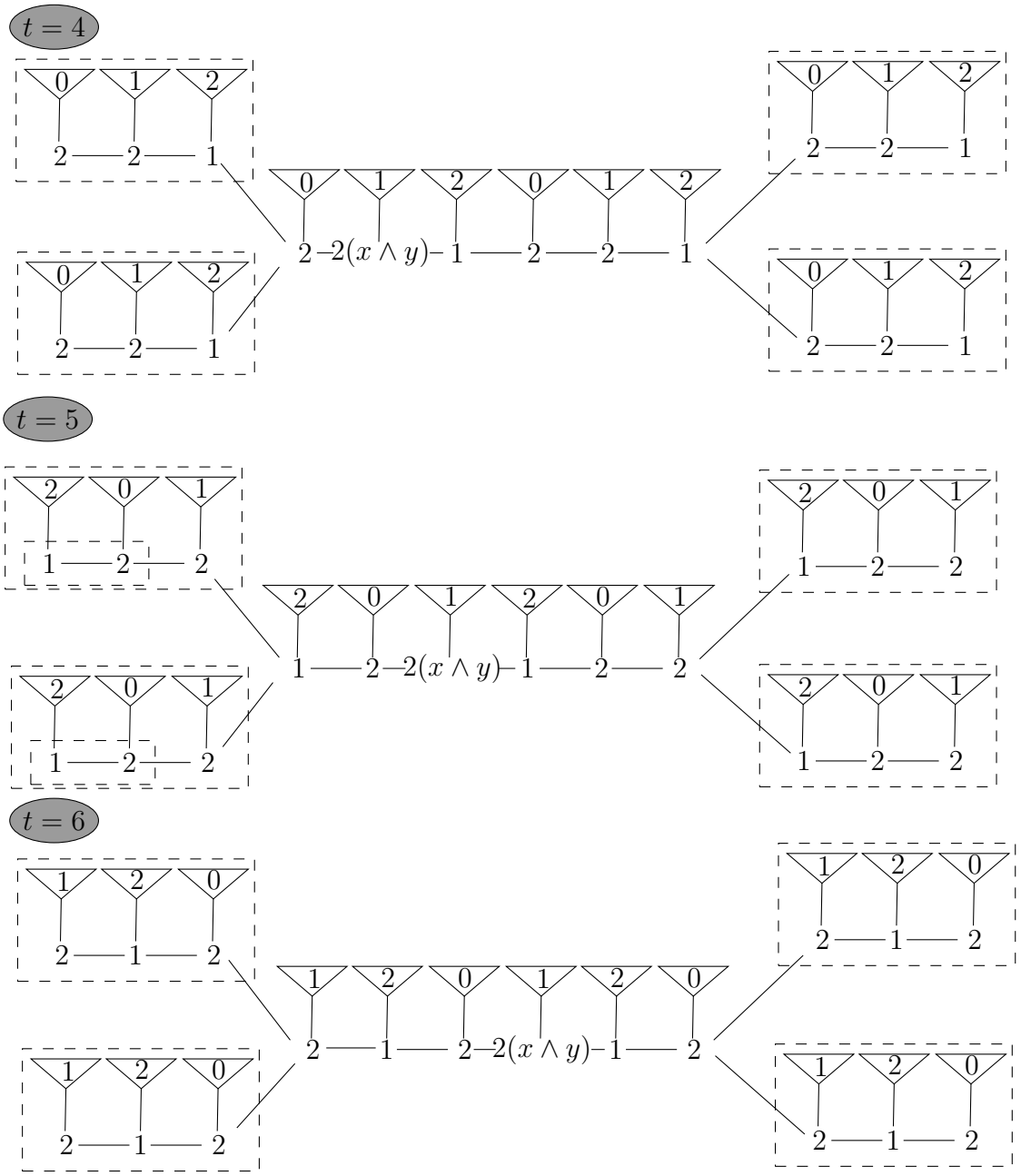


Figure 5.6: The fourth to sixth steps of the dynamics of the AND gadget implemented in a conjunctive network with firing memory. The variables $(x, y) \in \{0, 1\}^2$ represent the bits that the gadget is considering as inputs. Total computation takes $T = 9$ time steps. The triangles represent clock gadgets. The dashed boxes mark the embedded copies of the glueing interface which plays the role of output/input.

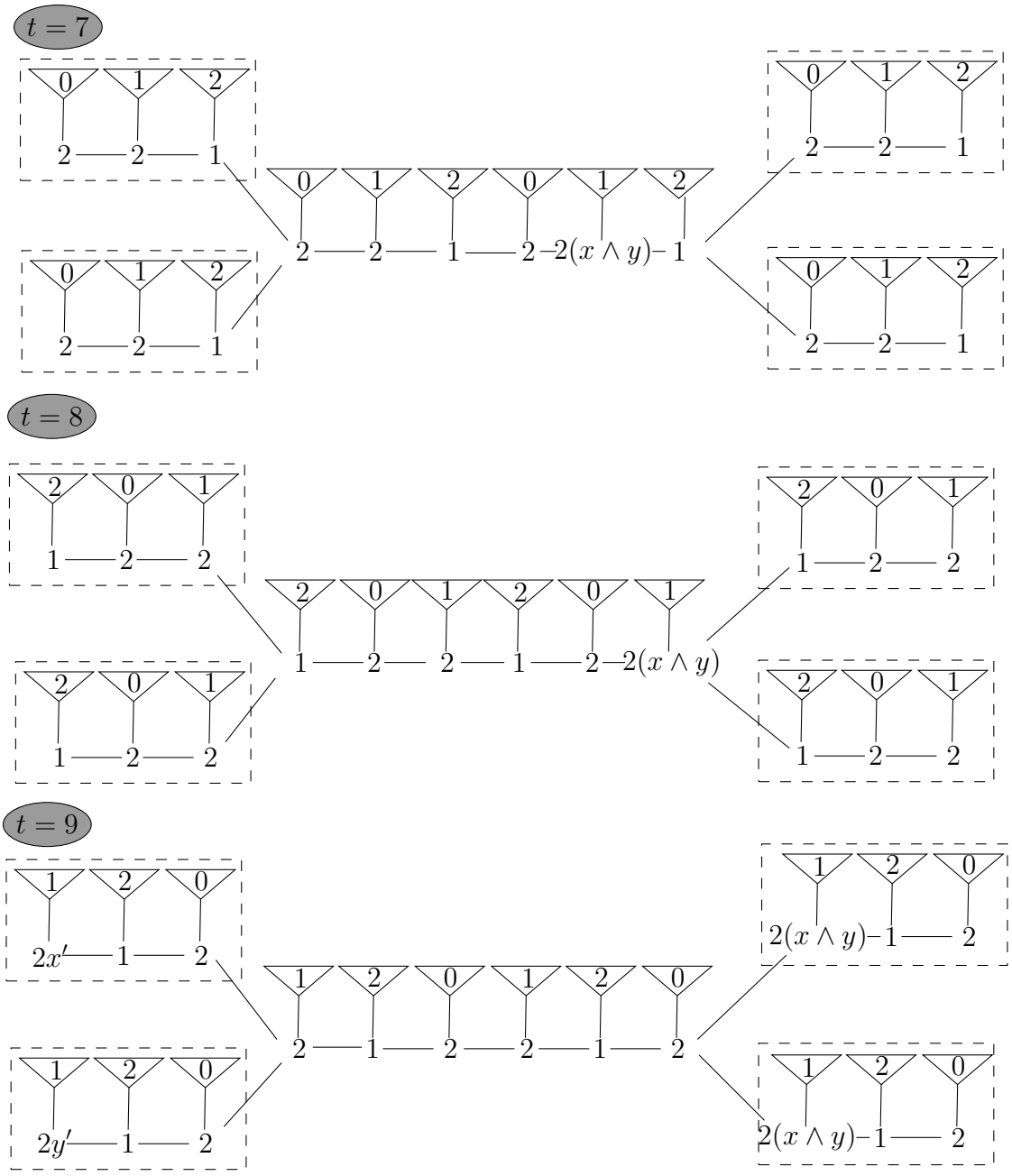


Figure 5.7: The final four steps of the dynamics of the AND gadget implemented in a conjunctive network with firing memory. The variables $(x, y) \in \{0, 1\}^2$ represent the bits that the gadget is considering as inputs. The variables $(x', y') \in \{0, 1\}^2$ represent the new information that the gadget is receiving as inputs. Total computation takes $T = 9$ time steps. The triangles represent clock gadgets. The dashed boxes mark the embedded copies of the glueing interface which plays the role of output/input.

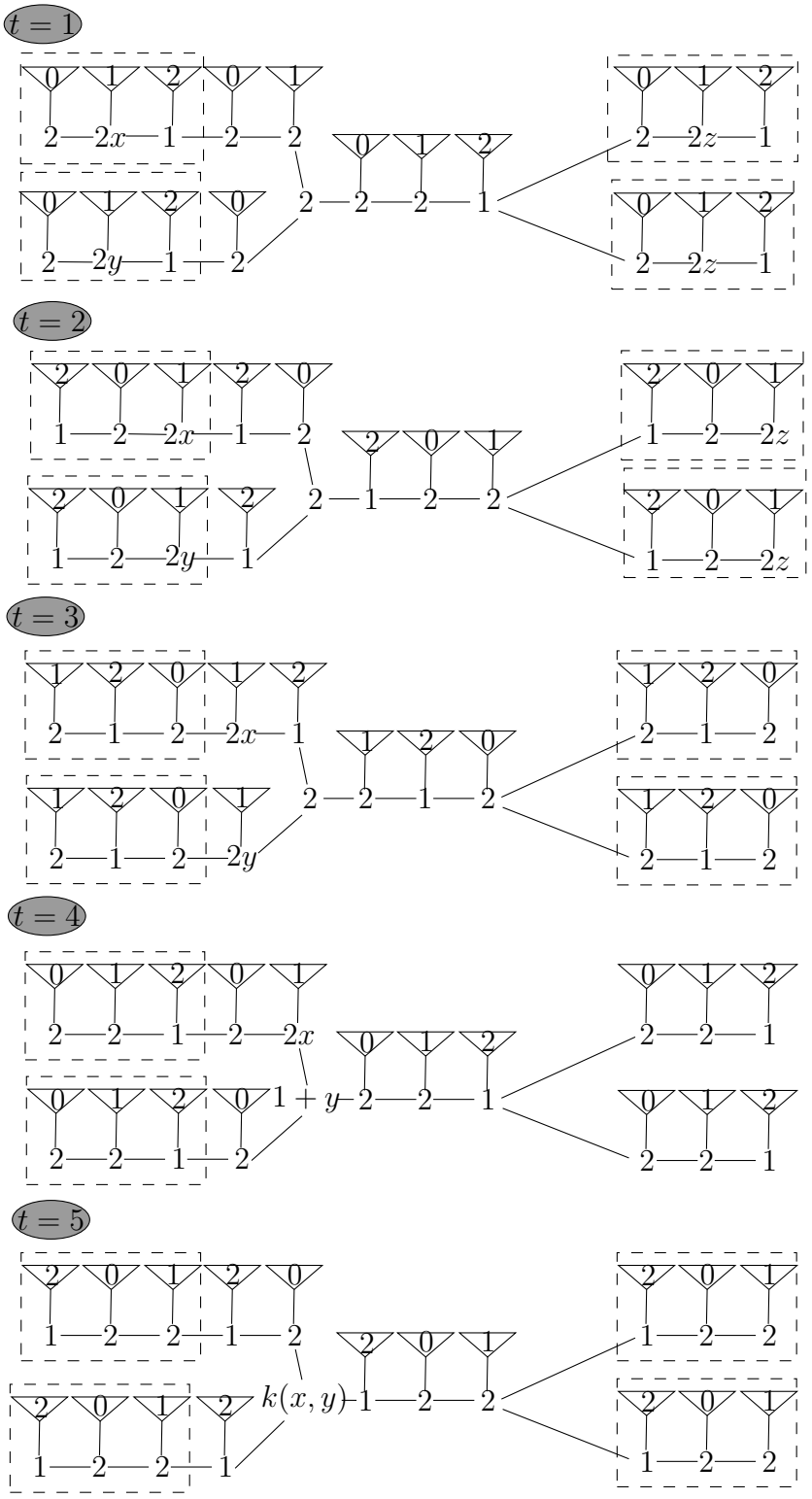


Figure 5.8: The first five steps of the OR gadget implemented in a conjunctive network with firing memory. The variables (x, y) represent the bits that the gadget is considering as inputs and $z \in \{0, 1\}$ correspond to some bit that is going to be send as input of other gadget. The function $k(x, y)$ is defined by $k(0, 0) = 0, k(1, 0) = k(1, 1) = 2$ and $k(0, 1) = 1$. Total computation takes $T = 9$ time steps. Triangles represent clock gadgets and different states are part of the context configurations. The dashed boxes mark the embedded copies of the glueing interface C which plays the role of outputs/inputs.

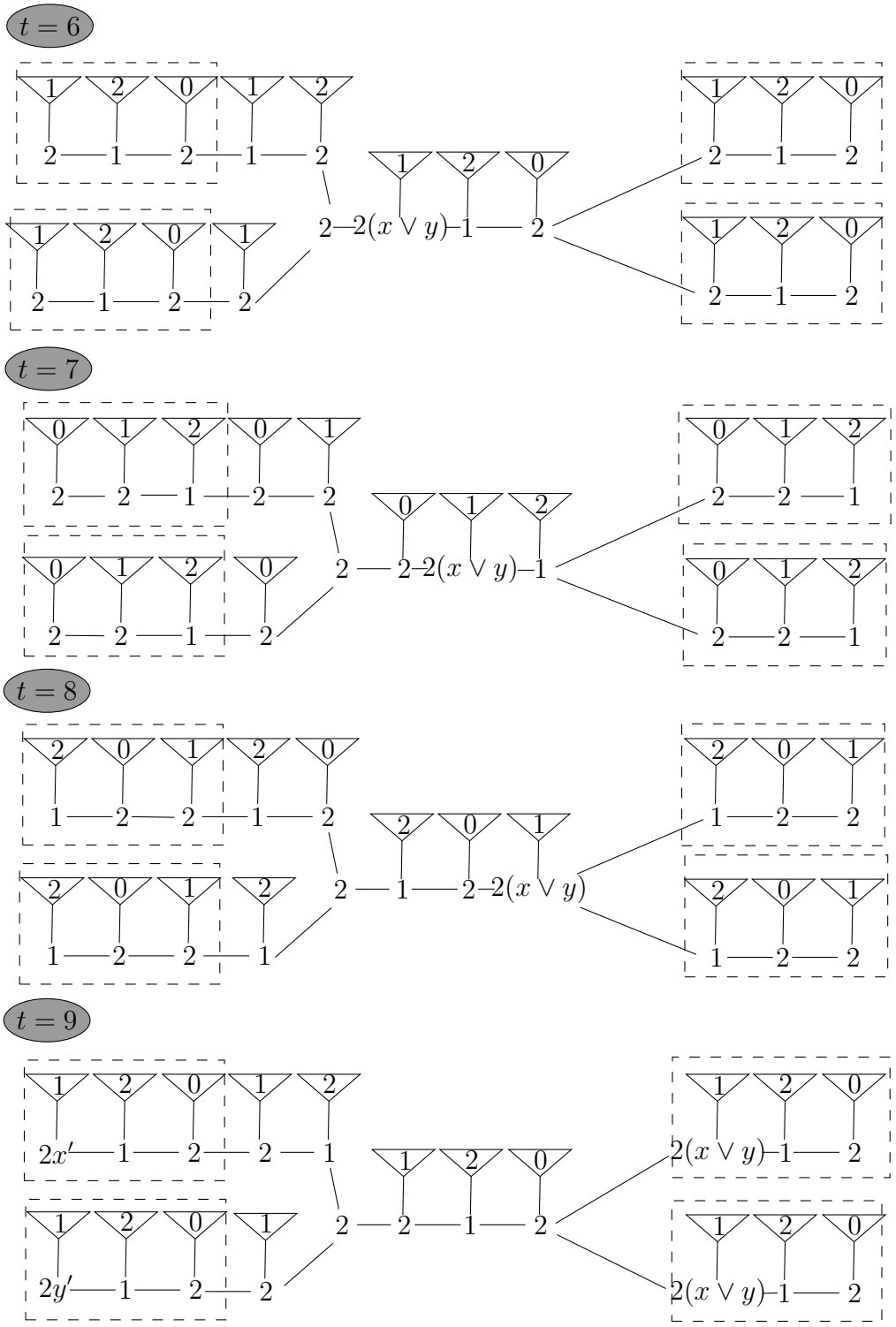


Figure 5.9: The last four steps of the OR gadget implemented in a conjunctive network with firing memory. The variables (x, y) represent the bits that the gadget is considering as inputs and $(x', y') \in \{0, 1\}^2$ correspond to new information that the gadget is interpreting as inputs. Total computation takes $T = 9$ time steps. Triangles represent clock gadgets and different states are part of the context configurations. The dashed boxes mark the embedded copies of the glueing interface C which plays the role of outputs/inputs.

5.2 Locally positive symmetric signed conjunctive networks

In this section, we study a generalization of conjunctive networks that we call *locally positive* symmetric signed conjunctive networks. We denote this family by $\mathcal{F}_{\text{locally-pos}}$. In this particular case, we allow edges to have negative signs (which will switch the state of the corresponding neighbor) but with a local constraint: no neighborhood in which all the connections are negative is allowed. More precisely, a locally positive symmetric conjunctive network is a CSAN (G, λ, ρ) in for any $v \in V(G)$ we have $\lambda_v(q', S) = \bigwedge_{q \in S} q$ and there exists $w \in N(v)$: $\rho(vw) = \text{Id}$.

We will show for this family that the *threshold of universality* when changing update modes is between block sequential update schemes and local clocks update schemes. More precisely, we show, on one hand, that the family remains dynamically constrained under block sequential schedule, and, on the other hand, we show that a local clocks version of this family is strongly universal as a consequence of its capability of simulating coherent $\mathcal{G}_{m,2}$-gadgets.

5.2.1 Block sequential update schemes

Theorem 5.8 *Fix any $b \geq 1$ and consider $\mathcal{F}^{\text{BLOCK},b}_{\text{LOCALLY-POS}}$ the family of all locally positive symmetric signed conjunctive networks under a block sequential schedule with at most b blocks. Any periodic orbit of any $\mathcal{F}^{\text{BLOCK},b}_{\text{LOCALLY-POS}}$ has period 1 or $2b$.*

Proof. Take some configuration x in a periodic orbit. If no node changes its state in the orbit then x is actually a fixed point. Otherwise take some node i that changes its state and consider a maximal time interval $I = [t_1; t_2]$ with $t_1 > 0$ during which i is in state 0: $\forall t \in I$, $F^t(x)_\text{i} = 0$ but $F^{t_1-1}(x)_\text{i} = F^{t_2+1}(x)_\text{i} = 1$. Let j be any positive neighborhood (i.e. such that $\rho(\text{i}, j)$ is the identity). First, we must have that $\forall t \in I$, $F^t(x)_j = 1$ because supposing $F^t(x)_\text{i} = F^t(x)_j = 0$ implies $F^{t'}(x)_\text{i} = F^{t'}(x)_j = 0$ for all $t' \geq t$ which would contradict the hypothesis that i changes its state in the orbit. Thus $t_2 - t_1 + 1 = b$ because it is by definition a multiple of b and if it were strictly larger than b then node j would be updated in the interval $[t_1; t_2 - 1]$ and therefore would turn into state 0 in the interval I which has just been proven impossible. The same argument actually shows that i and j must be updated synchronously. Therefore it is updated at time t_2 and we must have $F^{t_2+1}(x)_j = 0$. This implies that $F^{t_2+b+1}(x)_\text{i} = 0$ and shows that the maximal time interval starting from $t_2 + 1$ during which i is in state 1 is of length exactly b. We can then iterate this reasoning starting at time $t_2 + b + 1$ and we deduce that the orbit of x at node i alternates b steps in state 0 and b steps in state 1 forever. The same holds for any node that changes it state and finally we have shown that the orbit of x is of period 1 or $2b$. □

5.2.2 Local clocks update schemes

We now turn to the main construction of this section. We recall the definition of a local clocks version of a certain family. We will call $\mathcal{F}^{\text{CLOCK},c}_{\text{LOCALLY-POS}}$ the family of all locally positive symmetric conjunctive networks under local clocks update schemes with clock period c. A

local clock version of certain $(G, \lambda, \rho) \in \mathcal{F}_{\text{LOCALLY-POS}}$ is a CSAN over the alphabet $Q_c = Q \times \{0, \dots, c-1\} \times \{1, \dots, c\}$ defined by:

$$\overline{\lambda}(v)((x_Q)_v, (x_c)_v, (x_m)_v), (x_Q, x_c, x_m)|_{N(v)}) = \begin{cases} (\lambda(v)((x_Q)_v, x_Q|_{N(\mathrm{i})}), (\psi_{(x_m)_v}[(x_c)_v]) + 1 \mod (x_m)_v, (x_m)_v) & (x_c)_v = 0, \\ ((x_Q)_v, (\psi_{(x_m)_v}[(x_c)_v]) + 1 \mod (x_m)_v, (x_m)_v) & \text{otherwise} \end{cases}$$

where $\psi_m(r) : \{0, \dots, c-1\} \to \{0, \dots, c-1\}$ is such that $\psi_m[r] = \begin{cases} r & \text{if } r \leq m-1, \\ m-1 & \text{otherwise.} \end{cases}$

Now, we want to show that coherent AND/OR gadgets can be implemented in $\mathcal{F}^{\text{CLOCK},c}_{\text{LOCALLY-POS}}$, i.e. we want to show that this family has coherent $\mathcal{G}_{m,2}$ gadgets where $\mathcal{G}_{m,2} = \{\text{AND}_2, \text{OR}_2\}$ where $\text{OR}_2 : \{0,1\}^2 \to \{0,1\}^2$ is such that $\text{OR}_2(x, y) = (x \vee y, x \vee y)$ and the function $\text{AND}_2 : \{0,1\}^2 \to \{0,1\}^2$ is such that $\text{AND}_2(x, y) = (x \wedge y, x \wedge y)$. As a consequence, we will have that $\mathcal{F}^{\text{CLOCK},c}_{\text{LOCALLY-POS}}$ is strongly universal. However, in order to accomplish this task we need a construction that we will be using for the next subsection. In particular, we need to implement the gadgets that are shown in Figures 5.16 and 5.17. Then, we will adapt them in order to make them work for locally positive symmetric conjunctive networks.

Since the results on general signed symmetric conjunctive networks are required first in order to show the proof of the main theorem of this section, we will just state the main result and then, we will show the proof of Theorem 5.9 below in the next section, for sake of clarity. Doing so, we respect the hierarchical order between families and preserve coherence of results at the same time.

Theorem 5.9 *There exists $c > 0$ such that the family $\mathcal{F}^{\text{CLOCK},c}_{\text{LOCALLY-POS}}$ of all locally positive symmetric conjunctive networks under local clocks update scheme with clock parameter c has coherent $\mathcal{G}_{m,2}$-gadgets.*

Corollary 5.10 below is a direct consequence of Theorem 5.9.

Corollary 5.10 *There exists $c > 0$ such that the family $\mathcal{F}^{\text{CLOCK},c}_{\text{LOCALLY-POS}}$ of all locally positive symmetric conjunctive networks under local clocks update scheme with clock parameter c is strongly universal. In particular, $\mathcal{F}^{\text{CLOCK},c}_{\text{LOCALLY-POS}}$ is both dynamically and computationally complex.*

Proof. Proof is a direct consequence of Theorem 2.18, Corollary 2.14 and Corollary 3.18. □

5.3 Symmetric signed conjunctive networks

In this section we study symmetric signed conjunctive networks with no label constraint on edges. Formally the symmetric signed conjunctive networks family is a CSAN family in $\{0,1\}$ in which $\lambda_v : Q \times 2^Q \to Q$ is given by $\lambda(q, S) = 0$ if $0 \in S$ and $\lambda(q, S) = 1$ if $0 \notin S$ and for any $e \in E$ we have $\rho_e \in \{\text{Id}, \text{Switch}\}$ where $\text{Switch}(x) = 1 - x$. When an

edge is labeled by Id we say that it is a positive edge and when it is labeled by Switch we say that the edge is negative. We denote this family by $\mathcal{F}_{\text{SIGN-SYM-CONJ}}$ and we consider $\mathcal{F}^{\text{BLOCK},b}_{\text{SIGN-SYM-CONJ}}$, $\mathcal{F}^{\text{CLOCK},c}_{\text{SIGN-SYM-CONJ}}$ and $\mathcal{F}^{\text{PER},p}_{\text{SIGN-SYM-CONJ}}$ for block sequential, local clocks and periodic versions of this family respectively.

We start by remarking that for the parallel update scheme, $\mathcal{F}_{\text{SIGN-SYM-CONJ}}$ family is not universal since it is a type of threshold family and thus it has bounded period attractors (see [42]). Then, a natural question is whether this remains true for other update schemes. We are going to show that, $\mathcal{F}_{\text{SIGN-SYM-CONJ}}$ becomes strongly universal from block sequential update schemes. We recall that in $\mathcal{F}^{\text{BLOCK},b}_{\text{SIGN-SYM-CONJ}}$, local functions are modified by adding an internal clock. Each internal clock has the same period but a different initial shift. Thus, as explained in Chapter 4, a block sequential scheme on a network can be viewed as an ordered partition of its nodes into b blocks, such that the nodes belonging to a block are updated in parallel while the blocks are iterated sequentially. More precisely, we have the following definition:

$$\overline{\lambda(v)}(((x_Q)_v,(x_b)_v)(x_Q,x_b)|_{N(v)}) = \begin{cases} (\lambda(v)((x_Q)_v, x_Q|_{N(\mathrm{i})}), (x_b)_v - 1 \mod b) & \text{if } (x_b)_v = 0 \\ ((x_Q)_v, (x_b)_v - 1 \mod b) & \text{otherwise.} \end{cases}$$

5.3.1 Block sequential update schemes

In this section, we will show that $\mathcal{F}^{\text{BLOCK},b}_{\text{SIGN-SYM-CONJ}}$ is strongly universal as a consequence of its capability to implement coherent $\mathcal{G}_{m,2}$-gadgets. In addition, we conclude that, as a direct consequence of the latter property, that previous family is both dynamically complex and computationally complex. This means that for this family, complex behavior is exhibited under block sequential update schemes.

We start by showing this less powerful result: $\mathcal{F}^{\text{BLOCK},b}_{\text{SIGN-SYM-CONJ}}$ is able to simulate $\mathcal{G}_w$-networks. This is only a way to motivate our main result and to show how key structures work in a particular simple context. We do not use exactly this result in the actual proof of the main theorem but we apply the same type of ideas since the related structures are part of the gadgets we construct which have only negative labels, i.e. each edge is labeled by the Switch function.

Lemma 5.11 *The familly $\mathcal{F}^{\text{BLOCK},3}_{\text{SIGN-SYM-CONJ}}$ of all signed symmetric conjunctive networks under block sequential update schemes of at most 3 blocks has coherent $\mathcal{G}_w$-gadgets.*

Proof. We define a gadget simulating Id by considering two copies of the NOT gadget presented in Figures 5.10 and 5.11. Observe that this gadget has 3 central nodes (marked inside a thick dotted rectangle in Figures 5.10 and 5.11) together with 2 copies of a 4 nodes cycle graph. Generally speaking, the dynamics on these cycle graphs works as a clock which allows information to flow through the central part in only one direction (from left to right). In addition, they allow the gadget to erase information once it has been transmitted. This latter property allows the gadget to clean itself in order to receive new information. The wire

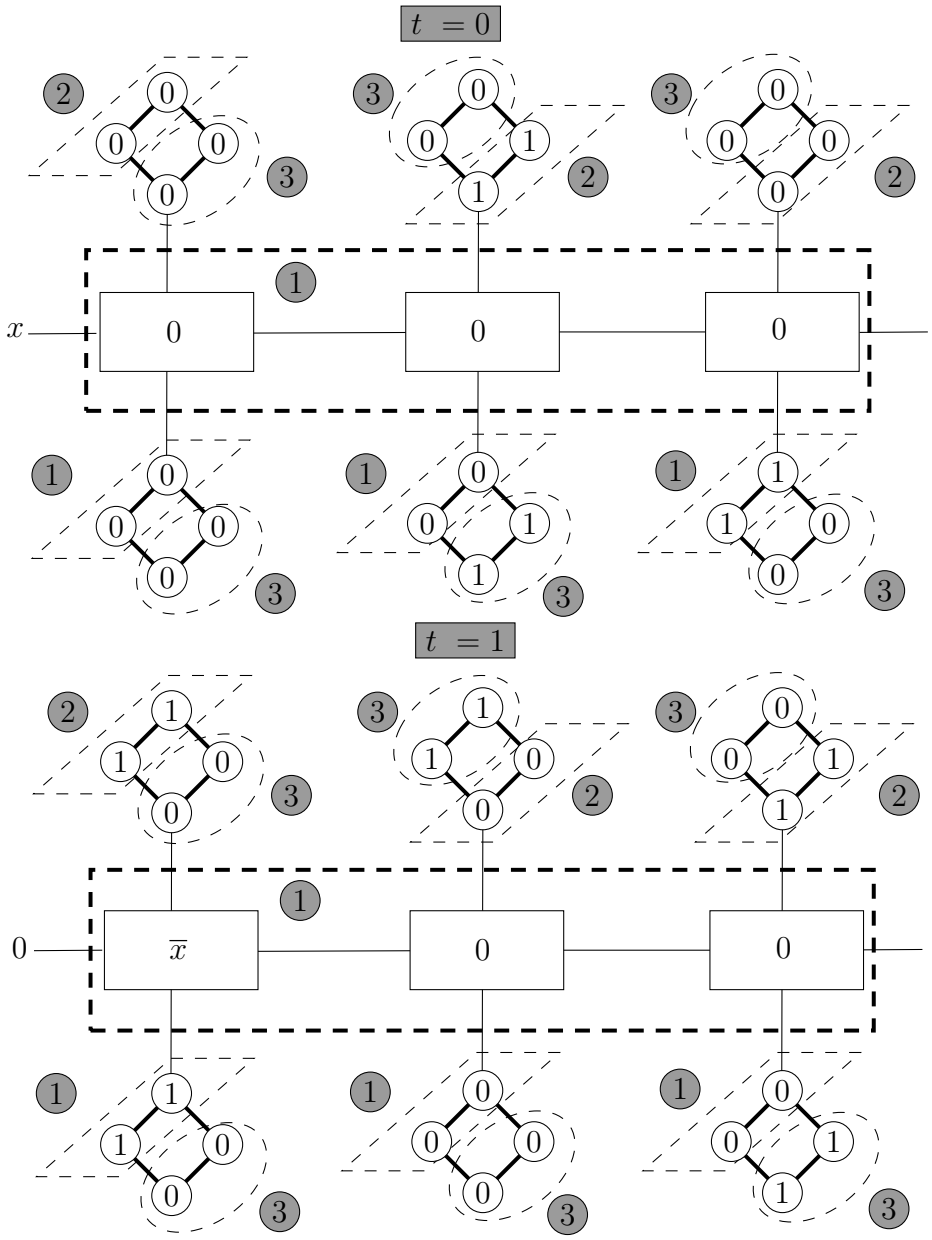


Figure 5.10: One step of the dynamics of the NOT part of AND/OR gadget implemented by a symmetric signed conjunctive network. Dashed ellipses and parallelograms represent blocks. Each block is labeled by its corresponding number (1, 2 and 3) in a gray colored circle. Thick dashed rectangle highlights nodes in the central part. All edges are negative. Each time step t is taken after three time steps (one for each block). Total simulation time is $T = 9$.

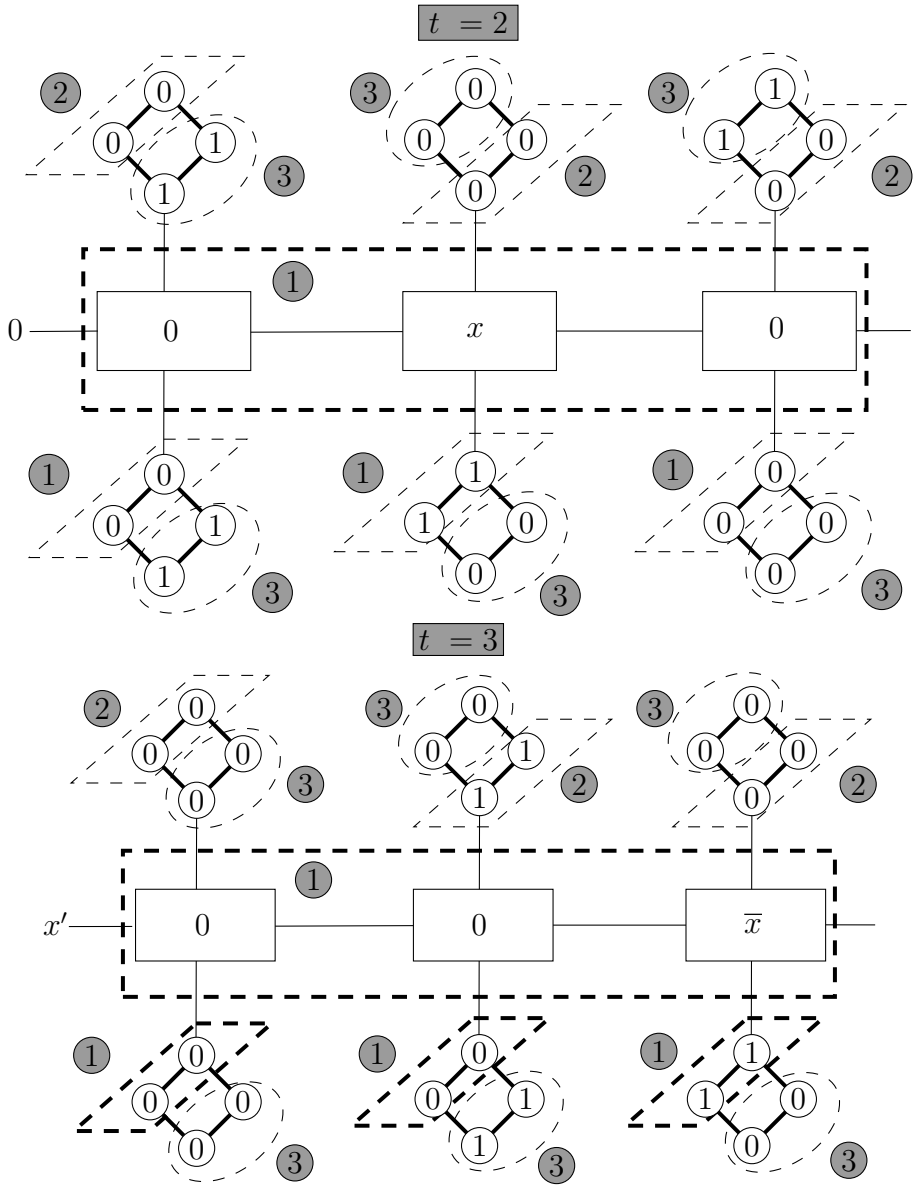


Figure 5.11: Two last steps of the dynamics described by the NOT part of AND/OR gadgets implemented by a symmetric signed conjunctive network. Dashed ellipses and parallelograms represent blocks. Each block is labeled by its corresponding number (1, 2 and 3) in a gray colored circle. Thick dashed rectangle highlights nodes in the central part. All edges are negative. Each time step t is taken after three time steps (one for each block). Total simulation time is $T = 9$.

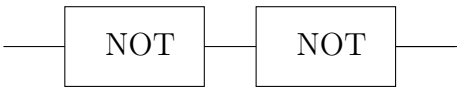


Figure 5.12: Wire gadget implemented on a signed symmetric conjuntive network. 2 copies of NOT gadget are combined in order to form a wire. Simulation time is $T = 6 \times 3$.

gadget is composed by two copies of the NOT gadget as is presented in Figure 5.12. Since this gadget is composed by two copies of the NOT gadget, this gadget is defined by a path graph with 6 central nodes together with $2 \times 2 \times 3 = 12$ cycle graphs (two copies for each node). We enumerate nodes in the central part from left to right by the following ordering: $\{0, 1, 2, 3, 4, 5\}$. Additionally, since the functioning of each copy of the NOT gadget is based in an ordered partition of 3 blocks, we take the union of the corresponding blocks on each partition in order to define 3 larger blocks for the wire gadget. For one of this larger blocks we use the notation $\{0, 1, 2\}$. Thus, observe that each wire takes $T = 6 \times 3 = 18$ time steps in order transport the information. This is because for each round of 3 time steps in which we update each block, we make the signal pass through exactly one node. We define the glueing interface as $C_{\mathrm{i}} = \{\mathrm{i}\}$ and $C_o = \{o\}$. We map input and output in the following way: $\phi^{\mathrm{i}}(\mathrm{i}) = 1$, $\phi^{\mathrm{i}}(o) = 0$, $\phi^o(\mathrm{i}) = \{5\}$ $\phi^o(o) = \{4\}$. Note that in each case the neighborhood of output and input part is completely contained in C_{i} and C_o respectively and also the image of C by functions ϕ is always the same (two nodes path graph). Thus, the glueing interface satisfies conditions of Lemma 3.8. Now we enumerate the remaining elements required by Definition 3.9:

- State configurations $s_q(\mathrm{i}) = (q, 0)$ and $s_q(o) = (q, 1)$ for any $q \in \{0, 1\}$.
- Context configurations are given in Figure 5.10 as well as the iteration order which defines the blocks.
- Standard trace and pseudo orbit for nodes $\{0, 1, 4, 5\}$ are defined in Figure 5.10 and Figure 5.11.

□

We are now in conditions to introduce the main result of this section:

Lemma 5.12 *The familly $\mathcal{F}^{\text{BLOCK},3}_{\text{SIGN-SYM-CONJ}}$ of all signed symmetric conjunctive networks under block sequential update schemes of at most 3 blocks has coherent $\mathcal{G}_{m,2}$-gadgets.*

Proof. We will show that the gadgets in the Figures 5.16 and 5.17 are coherent $\mathcal{G}_{m,2}$-gadgets. Note that both these gadgets are made of several wires made of NOT gadgets (we call the two in the left hand side of the figure input wires and the other two at the right hand side output wires) and a computation gadget. Observe that, each of these structures (wires and computation gadget) needs exactly 3 blocks. In addition, note that in Table 5.4 the dynamics of the 4-cycles that are attached to each node is shown. We recall that the function of these clocks is to allow information to flow in one direction only (all the interactions are symmetric so this is not straightforward) and to erase information once it has been copied or processed by the nodes in the gadget. We use the following notation in order to represent

nodes in these structures: $c_{s,\mathrm{i},j,p}$ where s is the number of the NOT gadget (a 3-node-path together with two 4-cycles for each node, see Figure 5.13) to which the cycle is attached, so $s \in \{1,2,3,4,5,6,7,8,9\}$ where the order is taken from left to right (for example in Figure 5.16 the copy associated to (v_1, v_2, v_3) comes first then, (v'_1, v'_2, v'_3), then (v_4, v_5, v_6) and so on). Additionally, i is in the position of the cycle in the NOT block, so $\mathrm{i} \in \{1,2,3\}$, j is the position of the node relative to the 4-cycle graph considered in counter clockwise order (see Figure 5.13), so $j \in \{1,2,3,4\}$ and p is the position of the cycle in the central structure of the gadget. Observe that, since there are two copies for each node, we denote them by *upper* and *lower* so $p \in \{u, l\}$. Since input wires and output wires are needed to be updated at the same time and are independent (we have two copies for each one) we combine their blocks in the obvious way (we take the union of pairs of blocks that have the same update order). More precisely, we combine 3×9 blocks of each particular part (there are 8 copies of NOT and one computation gadget having 3 blocks each one) in order to define again only 3 blocks. The precise definition, using notation shown in Figures 5.16 and 5.17, is the following:

- $B_0 = \bigcup_{\mathrm{i}=1}^{12} \{v_{\mathrm{i}}\} \cup \{v'_{\mathrm{i}}\} \cup \bigcup_{s,\mathrm{i}} \{c_{s,\mathrm{i},1,l}, c_{s,\mathrm{i},2,l}\}$;
- $B_1 = \bigcup_{s} \{c_{s,1,3,u}, c_{s,1,4,u}, c_{s,2,1,u}, c_{s,2,2,u}, c_{s,3,1,u}, c_{s,3,2,u}\}$; and
- $B_2 = \bigcup_{s,\mathrm{i}} \{c_{s,\mathrm{i},3,l}, c_{s,\mathrm{i},4,l}\}$.

Now we are going to use information in Tables 5.2, 5.3 and 5.4 in order to show that these gadgets satisfy the conditions of Definition 3.9. Observe that, on the one hand, for the OR gadget (see Figure 5.16), input wires compute the result in 3×3 (it needs to carry the signal through the three nodes in the wire and each of this intermediate steps takes three steps, one for each block) time steps and computation gadget takes 3×3 as well. On the other hand, for the AND gadget (see Figure 5.17) input wires compute the result in 3×6 time steps. We define the associated network as $F_{\mathrm{AND}} : (\{0,1\} \times \{0,1,2\})^{15\times 2\times 4} \to (\{0,1\} \times \{0,1,2\})^{15\times 2\times 4}$ and $F_{\mathrm{OR}} : (\{0,1\} \times \{0,1,2\})^{15\times 2\times 4} \to (\{0,1\} \times \{0,1,2\})^{15\times 2\times 4}$. In fact we have that:

1. There is a unique glueing interface given by $C = C_{\mathrm{i}} \cup C_o$ where:
   - $C_{\mathrm{i}} = \{\mathrm{i}\} \cup \{a(\mathrm{i},1), a(\mathrm{i},2), a(\mathrm{i},3), a(\mathrm{i},4)\} \cup \{a'(\mathrm{i},1), a'(\mathrm{i},2), a'(\mathrm{i},3), a'(\mathrm{i},4)\}$; and
   - $C_o = \{o', o\} \cup \{a(o',1), a(o',2), a(o',3), a(o',4)\} \cup \{a(o,1), a(o,2), a(o,3), a(o,4)\} \cup \{a'(o',1), a'(o',2), a'(o',3), a'(o',4)\} \cup \{a'(o,1), a'(o,2), a'(o,3), a'(o,4)\}$

   We define the labelling functions $\phi^{\mathrm{i}}_{\mathrm{AND},k}$, $\phi^{o}_{\mathrm{AND},k}$, $\phi^{\mathrm{i}}_{\mathrm{OR},k}$, $\phi^{o}_{\mathrm{OR},k}$ (for the sake of simplicity we show the definition for the AND gadget since the one for the OR gadget is completely analogous) for $k = 1, 2$ as:
   - $\phi^{\mathrm{i}}_{\mathrm{AND},1}(\mathrm{i}) = v_1$, $\phi^{\mathrm{i}}_{\mathrm{AND},1}(o') = v_2$ and $\phi^{\mathrm{i}}_{\mathrm{AND},1}(o) = v_3$;
   - $\phi^{\mathrm{i}}_{\mathrm{AND},1}(a(\mathrm{i},r)) = c_{1,1,r,u}$, $\phi^{\mathrm{i}}_{\mathrm{AND},1}(a'(\mathrm{i},r)) = c_{1,1,r,l}$ for $r = 1,2,3,4$, where 1 corresponds to the NOT gadget which starts with v_1 in Figure 5.17;
   - $\phi^{\mathrm{i}}_{\mathrm{AND},1}(a(o',r)) = c_{1,2,r,u}$, $\phi^{\mathrm{i}}_{\mathrm{AND},1}(a'(o',r)) = c_{1,2,r,l}$ for $r = 1,2,3,4$, where 1 corresponds to the NOT gadget which starts with v_1 in Figure 5.17;
   - $\phi^{\mathrm{i}}_{\mathrm{AND},1}(a(o,r)) = c_{1,3,r,u}$, $\phi^{\mathrm{i}}_{\mathrm{AND},1}(a'(o,r)) = c_{1,3,r,l}$ for $r = 1,2,3,4$, where 1 corresponds to the NOT gadget which starts with v'_1 in Figure 5.17;
   - $\phi^{\mathrm{i}}_{\mathrm{AND},2}(\mathrm{i}) = v'_1$, $\phi^{\mathrm{i}}_{\mathrm{AND},2}(o') = v'_2$ and $\phi^{\mathrm{i}}_{\mathrm{AND},2}(o) = v'_3$;

- $\phi^{\mathrm{i}}_{\mathrm{AND},2}(a(\mathrm{i},r)) = c_{1',1,r,u}$, $\phi^{\mathrm{i}}_{\mathrm{AND},1}(a'(\mathrm{i},r)) = c_{1',1,r,l}$ for $r = 1,2,3,4$, where 2 correspond to the NOT gadget which starts with v'_1 in Figure 5.17;
- $\phi^{\mathrm{i}}_{\mathrm{AND},2}(a(o',r)) = c_{1',2,r,u}$, $\phi^{\mathrm{i}}_{\mathrm{AND},1}(a'(o',r)) = c_{1',2,r,l}$ for $r = 1,2,3,4$, where 2 corresponds to the NOT gadget which starts with v'_1 in Figure 5.17;
- $\phi^{\mathrm{i}}_{\mathrm{AND},2}(a(o,r)) = c_{1',3,r,u}$, $\phi^{\mathrm{i}}_{\mathrm{AND},1}(a'(o,r)) = c_{1',3,r,l}$ for $r = 1,2,3,4$, where 2 corresponds to the NOT gadget which starts with v'_1 in Figure 5.17.
- $\phi^{o}_{\mathrm{AND},1}(\mathrm{i}) = v_{10}$, $\phi^{o}_{\mathrm{AND},1}(o') = v_{11}$ and $\phi^{o}_{\mathrm{AND},1}(o) = v_{12}$;
- $\phi^{o}_{\mathrm{AND},1}(a(\mathrm{i},r)) = c_{5,1,r,u}$, $\phi^{o}_{\mathrm{AND},1}(a'(\mathrm{i},r)) = c_{5,1,r,l}$ for $r = 1,2,3,4$, where 8 corresponds to the NOT gadget which starts with v_{10} in Figure 5.17.
- $\phi^{o}_{\mathrm{AND},1}(a(o',r)) = c_{5,2,r,u}$, $\phi^{o}_{\mathrm{AND},1}(a'(o',r)) = c_{5,2,r,l}$ for $r = 1,2,3,4$, where 8 corresponds to the NOT gadget which starts with v_{10} in Figure 5.17;
- $\phi^{o}_{\mathrm{AND},1}(a(o,r)) = c_{5,3,r,u}$, $\phi^{o}_{\mathrm{AND},1}(a'(o,r)) = c_{5,3,r,l}$ for $r = 1,2,3,4$, where 8 corresponds to the NOT gadget which starts with v_{10} in Figure 5.17;
- $\phi^{o}_{\mathrm{AND},2}(\mathrm{i}) = v'_{10}$, $\phi^{o}_{\mathrm{AND},2}(o') = v'_{11}$ and $\phi^{o}_{\mathrm{AND},2}(o) = v'_{12}$;
- $\phi^{\mathrm{i}}_{\mathrm{AND},2}(a(\mathrm{i},r)) = c_{5',1,r,u}$, $\phi^{\mathrm{i}}_{\mathrm{AND},2}(a'(\mathrm{i},r)) = c_{5',1,r,l}$ for $r = 1,2,3,4$, where 9 correspond to the NOT gadget which starts with v'_{10} in Figure 5.17;
- $\phi^{o}_{\mathrm{AND},2}(a(o',r)) = c_{5',2,r,u}$, $\phi^{o}_{\mathrm{AND},2}(a'(o',r)) = c_{5',2,r,l}$ for $r = 1,2,3,4$, where 9 corresponds to the NOT gadget which starts with v_{10} in Figure 5.17;
- $\phi^{o}_{\mathrm{AND},2}(a(o,r)) = c_{5',3,r,u}$, $\phi^{o}_{\mathrm{AND},2}(a'(o,r)) = c_{5',3,r,l}$ for $r = 1,2,3,4$, where 9 corresponds to the NOT gadget which starts with v'_{10} in Figure 5.17.

2. State configurations are defined for each $q \in \{0,1\}$ as $s_q(\mathrm{i}) = (q,1)$ and $s_q(o') = s_q(o) = (0,1)$ and the state configuration of the nodes in the clocks i.e. the ones labeled by a are constant and shown in Table 5.4. The block number for each of these nodes can is the same as the original NOT gadget 5.10.
3. Context configurations are described in Tables 5.2, 5.3 and 5.4 as the ones related to the cycles of length 4 connected to central path of the gadgets and nodes in the path which are not part of the glueing interface.
4. Standard trace is defined in Tables 5.2 and 5.3 (which contain the information related to the dynamics of nodes $v_1, v'_1, v_2, v'_2, v_3, v'_3, v_{10}, v'_{10}, v_{11}, v'_{11}, v_{12}, v'_{12}$) and in Table 5.4 (which contains the dynamics of the nodes in the 4-cycles).
5. Simulation constant is $T = 3 \times 12$ as it is shown in Tables 5.2,5.3 and
6. Pseudo-orbit is given by the dynamics shown in in Tables 5.2, 5.3, and 5.4 where x, y, x', y', z are variables.

□

Corollary 5.13 *The familly $\mathcal{F}^{\text{BLOCK},2}_{\text{SIGN-SYM-CONJ}}$ of all signed symmetric conjunctive networks under block sequential update schemes with at most 3 blocks is strongly universal. In particular, it is both dynamically and computationally complex.*

Proof. Strong universality holds from the fact that family is capable of simulating $\mathcal{G}_{m,2}$ in linear space and constant time (see Corollary 3.18). The family is dynamically and com-

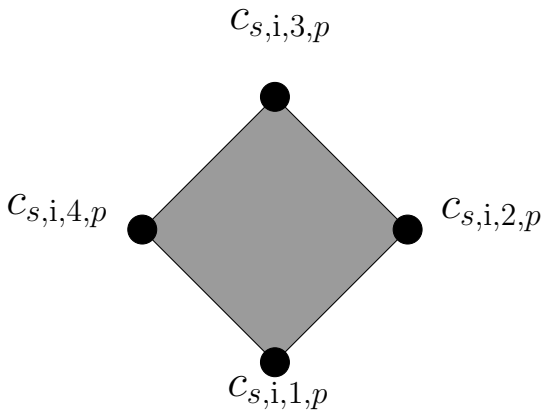


Figure 5.13: Scheme of labeling for 4-cycles in AND/OR gadgets. Notation is given by the following guidelines: s represent the associated group of three nodes, second two coordinates indicate its position relative the group of three nodes and its position in the 4-cycle graph (considering counter clock-wise order), and u, l stands for upper or lower according to its position in the gadget.

Node/Time	v_1	v_2	v_3	v_4	v_5	v_6	v'_1	v'_2	v'_3	v'_4	v'_5	v'_6	w_1	w_2	w_3	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v'_7	v'_8	v'_9	v'_{10}	v'_{11}	v'_{12}
0	x	0	0	0	0	0	y	0	0	0	0	0	0	0	0	0	0	0	z	0	0	0	0	0	z	0	0
3	0	$\overline{x}$	0	0	1	0	0	$\overline{y}$	0	0	1	0	0	1	0	0	1	0	0	$\overline{z}$	0	0	1	0	0	$\overline{z}$	0
6	0	0	x	0	0	0	0	0	y	0	0	0	0	0	0	0	0	0	0	0	z	0	0	0	0	0	z
9	1	0	0	$\overline{x}$	0	0	1	0	0	$\overline{y}$	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
12	0	0	0	0	x	0	0	0	0	0	y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	1	0	0	$\overline{x}$	0	0	1	0	0	$\overline{y}$	0	0	1	0	0	1	0	0	1	0	1	0	0	0	1
18	0	0	0	0	0	0	0	0	0	0	0	0	$x \wedge y$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	1	0	0	1	0	0	1	0	0	1	0	0	$\overline{x \wedge y}$	0	0	1	0	0	1	0	0	1	0	0	1	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$x \wedge y$	0	0	0	0	0	0	0	0	0	0	0	0
27	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	$\overline{x \wedge y}$	0	0	1	0	0	$\overline{x \wedge y}$	0	0	1	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$x \wedge y$	0	0	0	0	0	$x \wedge y$	0	0	0	0
33	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	$\overline{x \wedge y}$	0	0	1	0	0	$\overline{x \wedge y}$	0	0	1
36	x'	0	0	0	0	0	y'	0	0	0	0	0	0	0	0	0	0	0	$x \wedge y$	0	0	0	0	0	$x \wedge y$	0	0

Table 5.2: Dynamics of the central gadgets in an AND gadget implemented over a symmetric signed conjunctive network. Notation is the same of the one used in Figure 5.17

putationally complex as a direct consequence of strong universality (see Theorem 2.18 and Corollary 2.14). □

We are now in conditions to resume the proof of Theorem 5.9. Let us recall its statement and provide the related proof.

> There exists $c > 0$ such that the family $\mathcal{F}^{\text{CLOCK},c}_{\text{LOCALLY-POS}}$ of all locally positive symmetric conjunctive networks under local clocks update scheme with clock parameter c has coherent $\mathcal{G}_{m,2}$-gadgets.

Proof. We start by observing that the gadgets in Figures 5.16 and 5.17 can be implemented in $\mathcal{F}^{\text{CLOCK},c}_{\text{LOCALLY-POS}}$ for some c. This can be easily done by adding a positive node to each node in the gadget. The main idea here is that each of these artificial positive nodes will play no role in calculations and will stay in state 1 most of the time. In fact, it suffices that these positive neighbors reach state 1 before critical steps of computation are performed inside the gadget.

In order to illustrate this idea, let us consider two different cases and analyze why com-

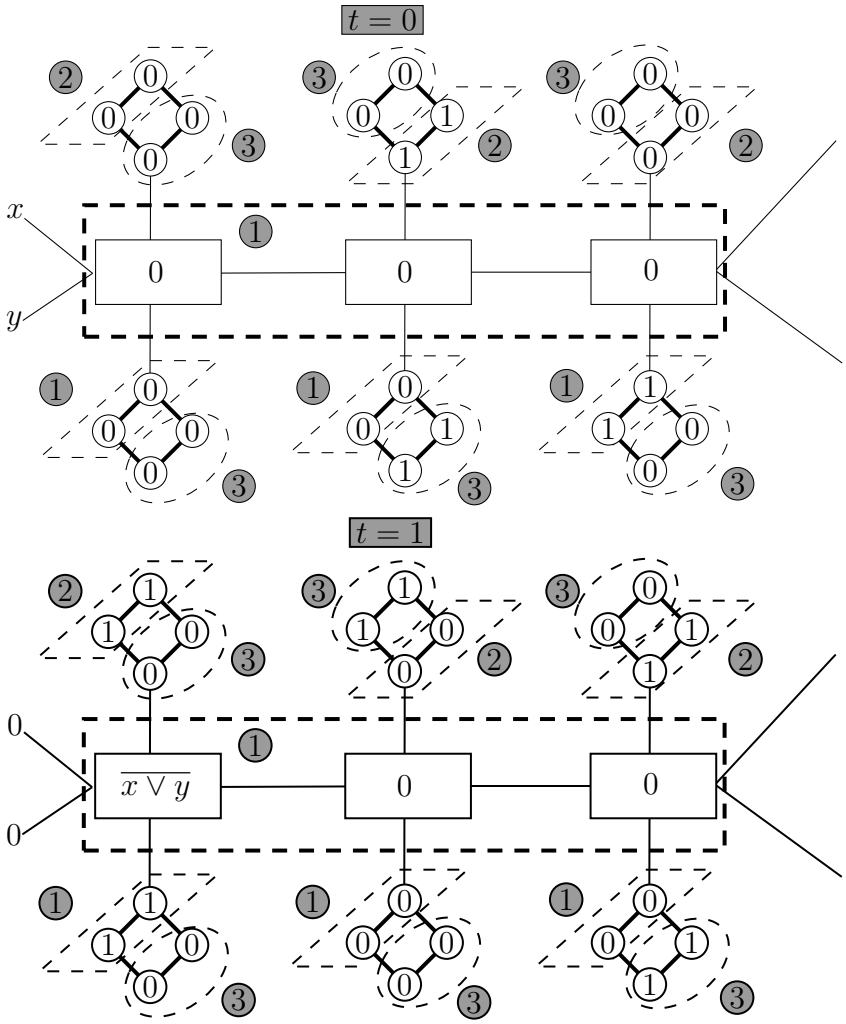


Figure 5.14: One step of the dynamics of the computation gadget inside AND/OR gadget implemented by a symmetric signed conjunctive network. Dashed ellipses and parallelograms represent blocks. Each block is labeled by its corresponding number (1, 2 and 3) in a gray colored circle. Thick dashed rectangle highlights nodes in the central part. All edges are negative. Each time step t is taken after three time steps (one for each block). Total simulation time is $T = 9$.

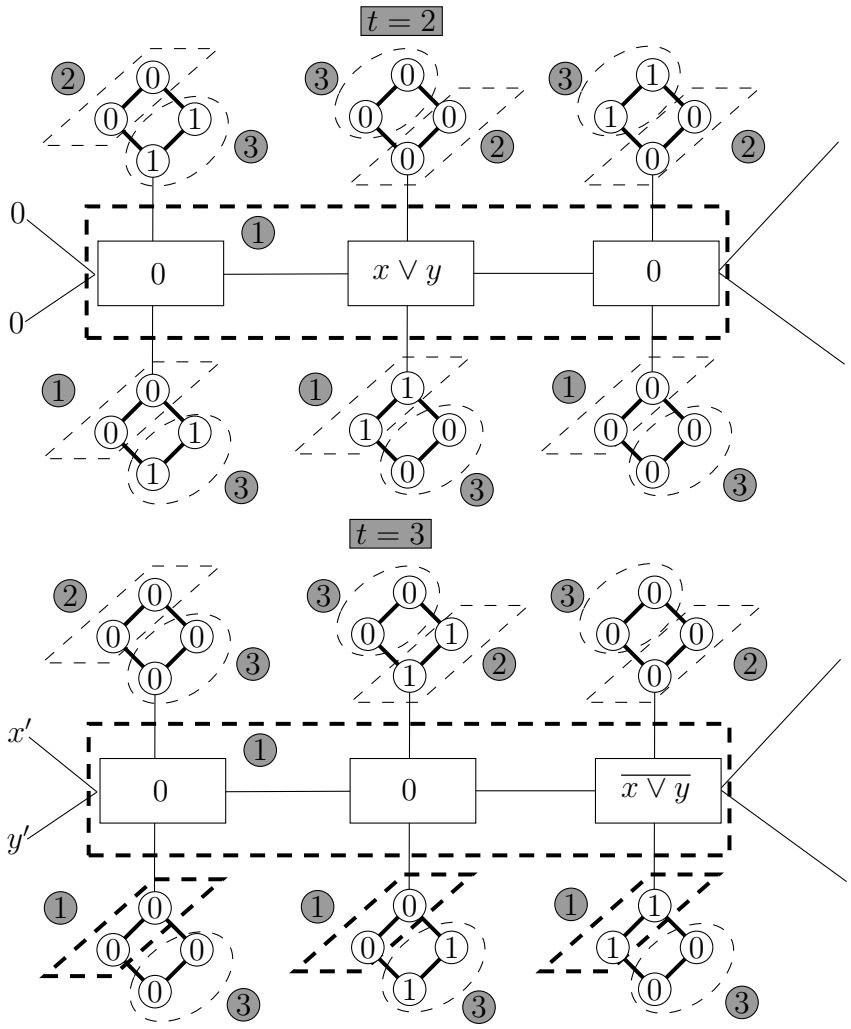


Figure 5.15: Last two steps of the dynamics of the computation gadget inside AND/OR gadget implemented by a symmetric signed conjunctive network. Dashed ellipses and parallelograms represent blocks. Each block is labeled by its corresponding number (1, 2 and 3) in a gray colored circle. Thick dashed rectangle highlights nodes in the central part. All edges are negative. Each time step t is taken after three time steps (one for each block). Total simulation time is $T = 9$.

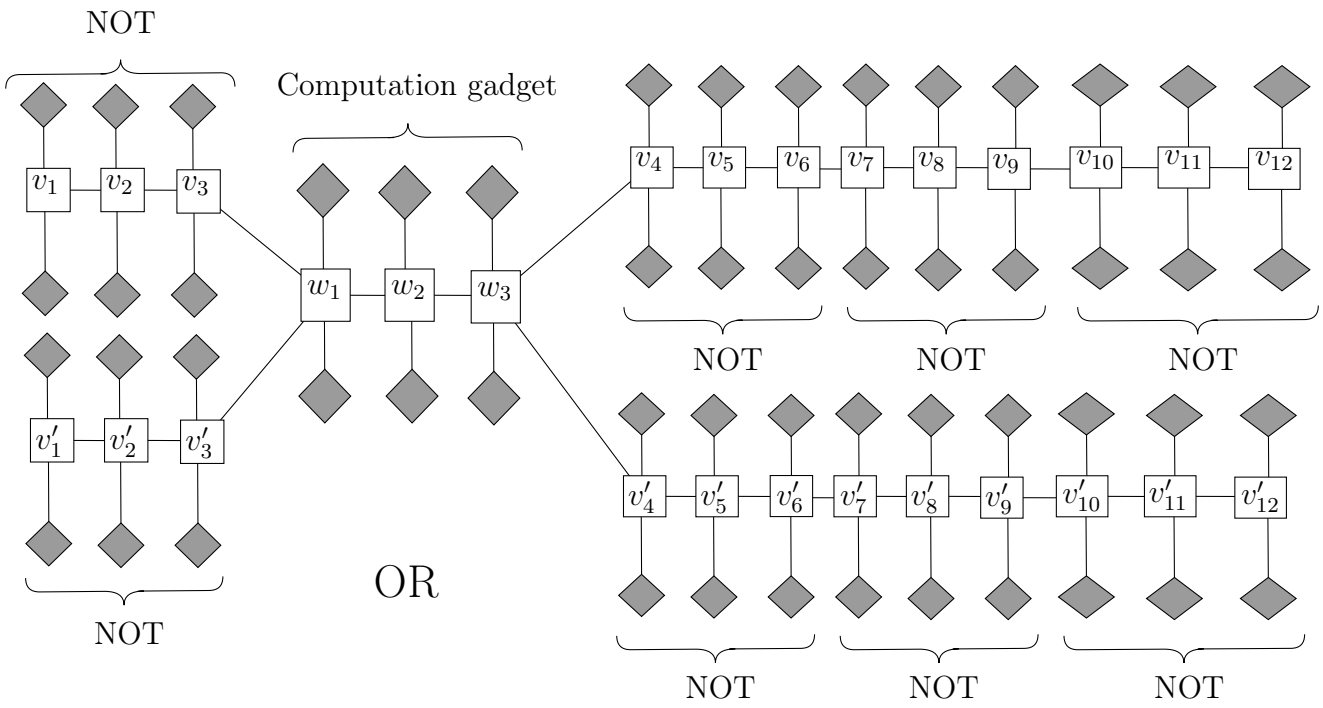


Figure 5.16: OR gadget structure. In order to produce a OR gadget, wire gadget and NOT gadget are combined with the computation part depicted in Figures 5.14 and 5.15.

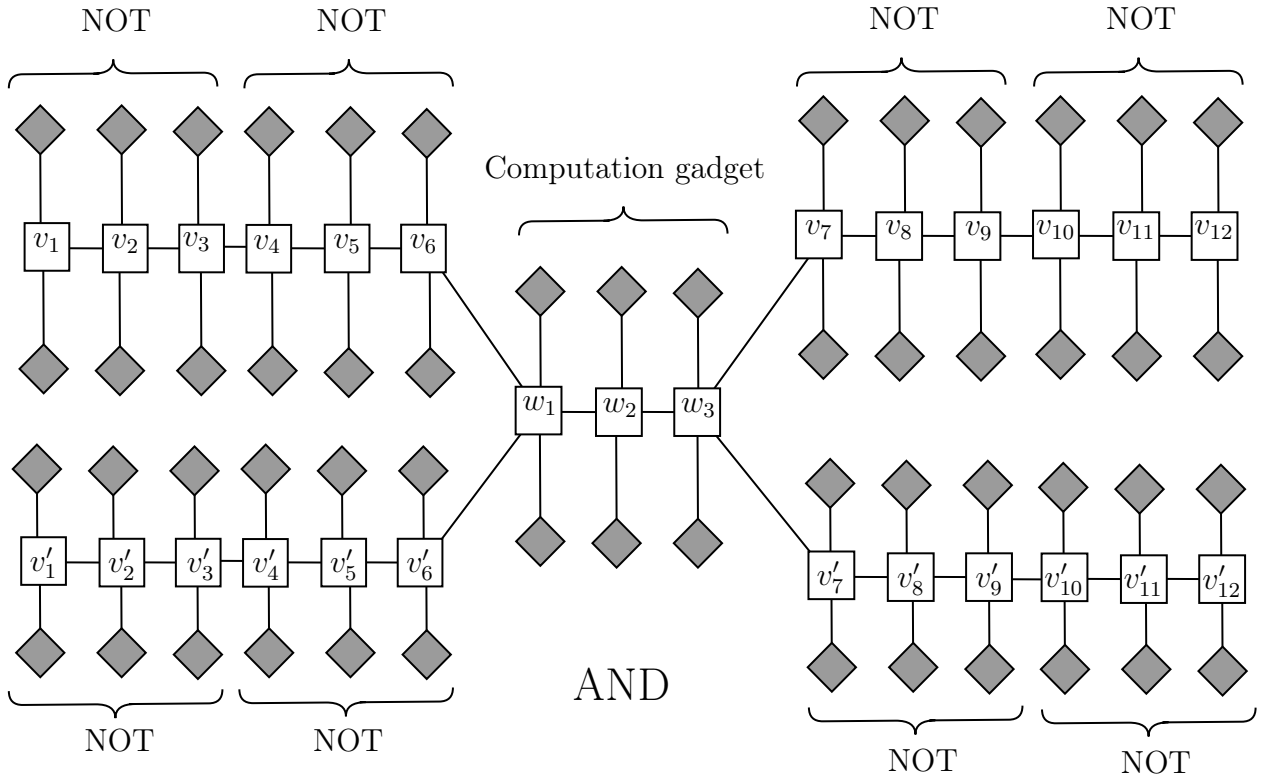


Figure 5.17: AND gadget structure. In order to implement an AND gadget, wire gadget and NOT gadget are combined with the computation part depicted in Figures 5.14 and 5.15.

Node/Time	v_1	v_2	v_3	v'_1	v'_2	v'_3	w_1	w_2	w_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v'_4	v'_5	v'_6	v'_7	v'_8	v'_9	v'_{10}	v'_{11}	v'_{12}
0	x	0	0	y	0	0	0	0	0	0	0	0	0	0	0	z	0	0	0	0	0	0	0	0	z	0	0
3	0	$\overline{x}$	0	0	$\overline{y}$	0	0	1	0	0	1	0	0	1	0	0	$\overline{z}$	0	0	1	0	0	1	0	0	$\overline{z}$	0
6	0	0	x	0	0	y	0	0	0	0	0	0	0	0	0	0	0	z	0	0	0	0	0	0	0	0	z
9	1	0	0	1	0	0	$\overline{x \vee y}$	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
12	0	0	0	0	0	0	0	$x \vee y$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	1	0	0	1	0	0	$\overline{x \vee y}$	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0
18	0	0	0	0	0	0	0	0	0	$x \vee y$	0	0	0	0	0	0	0	0	$x \vee y$	0	0	0	0	0	0	0	0
21	0	1	0	0	1	0	0	1	0	0	$\overline{x \vee y}$	0	0	1	0	0	1	0	0	$\overline{x \vee y}$	0	0	1	0	0	1	0
24	0	0	0	0	0	0	0	0	0	0	0	$x \vee y$	0	0	0	0	0	0	0	0	$x \vee y$	0	0	0	0	0	0
27	1	0	0	1	0	0	1	0	0	1	0	0	$\overline{x \vee y}$	0	0	1	0	0	1	0	0	$\overline{x \vee y}$	0	0	1	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	$x \vee y$	0	0	0	0	0	0	0	0	$x \vee y$	0	0	0	0
33	0	0	1	0	0	1	0	0	1	0	0	1	0	0	$\overline{x \vee y}$	0	0	1	0	0	1	0	0	$\overline{x \vee y}$	0	0	1
36	x'	0	0	y'	0	0	0	0	0	0	0	0	0	0	0	$x \vee y$	0	0	0	0	0	0	0	0	$x \vee y$	0	0

Table 5.3: Dynamics for central gadgets in OR gadget implemented over a symmetric signed conjunctive network. Notation is the same of the one shown in Figure 5.16

Node/Time	$c_{s,1,1,u}$	$c_{s,1,2,u}$	$c_{s,1,3,u}$	$c_{s,1,4,u}$	$c_{s,2,1,u}$	$c_{s,2,2,u}$	$c_{s,2,3,u}$	$c_{s,2,4,u}$	$c_{s,3,1,u}$	$c_{s,3,2,u}$	$c_{s,3,3,u}$	$c_{s,3,4,u}$
0	0	0	1	1	0	0	1	1	1	1	0	0
3	1	1	0	0	0	0	0	0	0	0	1	1
6	0	0	0	0	1	1	0	0	0	0	0	0
9	0	0	1	1	0	0	1	1	1	1	0	0
Node/Time	$c_{s,1,1,l}$	$c_{s,1,2,l}$	$c_{s,1,3,l}$	$c_{s,1,4,l}$	$c_{s,2,1,l}$	$c_{s,2,2,l}$	$c_{s,2,3,l}$	$c_{s,2,4,l}$	$c_{s,3,1,l}$	$c_{s,3,2,l}$	$c_{s,3,3,b}$	$c_{s,3,4,b}$
0	1	1	0	0	0	0	0	0	0	0	1	1
3	0	0	1	1	1	1	0	0	0	0	0	0
6	0	0	0	0	0	0	1	1	1	1	0	0
9	1	1	0	0	0	0	0	0	0	0	1	1

Table 5.4: Dynamics for context in AND/OR gadgets implemented on symmetric signed conjunctive networks.

putation gadget still works in this case:

1. **Nodes in 4-cycles**: observe that these nodes have a fixed trajectory that is independent on the input that computation part is handling. Thus, it suffices to note that each node in the context effectively changes its state (they are in an attractor of period 3×3 as it is shown in Figure 5.4). As a consequence of this latter observation, we can set the local period of each positive neighbor so it is updated when its neighbor in the clock is in state 1. More precisely, we fix the corresponding local period value to 9 and correctly initialize them so each positive neighbor is updated exactly when their correspondent node is in state 1.
2. **Central nodes**: Observe that, in this case, we have that in the pseudo-orbit given in Table 5.2 and Table 5.3 each node eventually reaches the state 1 independently from the value of x, y, z, x' and y'. Thus, as same as the nodes that are in the 4-cycles, it suffices to set up the local clock of each positive neighbor in order to be updated while its neighbor in the gadget is in state 1. More precisely, it suffices to set up clocks following values in Table 5.2 and Table 5.3 and set clocks to be updated every 18 time-steps. Note that this works since nodes in the central part are in the first block so positive neighbors are updated at the same time as its neighbors but only when nodes in the gadget are in state 1.

Thus, gadgets in Figures 5.16 and 5.17 can be implemented as same as we did for general symmetric signed conjunctive networks and the desired result holds.

□

5.4 Symmetric min-max networks

In this section, we study min-max networks. This is also a particular CSAN family in which local functions take the maximum or minimum value of some set of states. More precisely, a min-max network with ordered alphabet Q is a CSAN (G, λ, ρ) characterized by local functions $\lambda_v(q, S) = \min S$ or $\lambda_v(q, S) = \max S$ for every $v \in V(G)$ and edge labels $\rho_{\mathrm{e}} = \{\mathrm{Id}\}$ for each $\mathrm{e} \in E(G)$. Note that in the particular case in which $Q = \{0, 1\}$ we have $\lambda_v(q, S) = \bigvee_{x \in S} x$ or $\lambda_v(q, S) = \bigwedge_{x \in S} s$ for every $v \in V(G)$. We call this particular CSAN family the family of AND-OR networks and we write $\mathcal{F}_{\text{AND-OR}}$, $\mathcal{F}^{\text{BLOCK},b}_{\text{AND-OR}}$, $\mathcal{F}^{\text{CLOCK},c}_{\text{AND-OR}}$ and $\mathcal{F}^{\text{PER},p}_{\text{AND-OR}}$ to denote different update schemes as we did before. We use the notation $\mathcal{F}_{\text{MIN-MAX}}$, $\mathcal{F}^{\text{BLOCK},b}_{\text{MIN-MAX}}$, $\mathcal{F}^{\text{CLOCK},c}_{\text{MIN-MAX}}$ and $\mathcal{F}^{\text{PER},p}_{\text{MIN-MAX}}$ for any alphabet Q.

In the Boolean case, max and min functions are threshold functions because

$$\min(x_1, \dots, x_k) = 1 \Leftrightarrow \sum x_{\mathrm{i}} = k$$

and

$$\max(x_1, \dots, x_k) = 0 \Leftrightarrow \sum x_{\mathrm{i}} = 0.$$

Therefore, their periodic orbits are of length at most 2 by [42]. In the non-Boolean case, they are not threshold functions. However, as shown by the following lemma the general alphabet case can be understood through multiple factorings onto the Boolean case.

Lemma 5.14 *Let $n \geq 2$ and let $\mathcal{A} = (G, \lambda, \rho)$ be a min-max automata network with alphabet Q, $|Q| > 1$, such that $|V(G)| = n$. Let F be a global rule for $\mathcal{A}$. There exists an AND-OR automata network $\mathcal{A}^* = (G, \lambda, \rho)$ and a global rule $F^* : \{0,1\}^n \to \{0,1\}^n$ such that for every $\alpha \in Q$ the function $\pi^\alpha : Q^n \to \{0,1\}^n$ is such that $\pi^\alpha \circ F = F^* \circ \pi^\alpha$ where* $\pi^\alpha(x)_{\mathrm{i}} = \begin{cases} 1 & \text{if } x_{\mathrm{i}} \geq \alpha \\ 0 & \text{else.} \end{cases}$

Proof. Let us take F^* as the global function given by the same min/max labels than F but considering the fact that in the alphabet $\{0, 1\}$ we have $\min(x, y) = x \wedge y$ and $\max(x, y) = x \vee y$. Fix $\alpha \in Q$ and let $x \in Q^n$. Additionally, let us fix $\mathrm{i} \in V$. Suppose that $F(x)_{\mathrm{i}} = \max(x_{\mathrm{i}_1}, \dots, x_{\mathrm{i}_k})$ then $F^*(\pi^\alpha(x))_{\mathrm{i}} = \bigvee_{s=1}^{k} \pi^\alpha(x)_{\mathrm{i}_s}$ where $N(\mathrm{i}) = \{\mathrm{i}_1, \dots \mathrm{i}_k\}$. Note that

$$\pi^\alpha(F(x))_{\mathrm{i}} = \begin{cases} 1 & \text{if } \max(x_{\mathrm{i}_1}, \dots, x_{\mathrm{i}_k}) \geq \alpha, \\ 0 & \text{otherwise.} \end{cases}$$

Then, it is clear that we have $\pi^\alpha(F(x))_{\mathrm{i}} = 1$ if and only if $\pi^\alpha(x)_{\mathrm{i}_s} = 1$ for some $s \in \{1, \dots, k\}$ and thus if and only if $\bigvee_{s=1}^{k} \pi^\alpha(x)_{\mathrm{i}_s} = 1$ which is equivalent to $F^* \circ \pi(x)_{\mathrm{i}} = 1$. The case in which $F(x)_{\mathrm{i}} = \min(x_{\mathrm{i}_1}, \dots, x_{\mathrm{i}_k})$ is analogous since we have that $F^*(\pi^\alpha(x))_{\mathrm{i}} = \bigwedge_{s=1}^{k} \pi^\alpha(x)_{\mathrm{i}_s}$. □

We deduce that periodic orbits in MIN-MAX networks are of length at most 2 and therefore the family cannot be universal.

Corollary 5.15 *Let F be any MIN-MAX network over alphabet Q and x any limit configuration (i.e. such that $F^t(x) = x$ for some t). Then, $F^2(x) = x$. Therefore the family of MIN-MAX networks cannot be universal (considered under the parallel update scheme).*

Proof. We show that $F^2(x)_\mathrm{i} = x_\mathrm{i}$ for any node i. Let q be the maximum state appearing in the sequence $(F^t(x)_\mathrm{i})_{t\in\mathbb{N}}$. Using Lemma 5.14 with projection π^q and the fact that periodic orbits of AND-OR networks have period at most 2 we deduce that if $F^t(x)_\mathrm{i} = q$ then $F^{t+2}(x)_\mathrm{i} = q$ for some t. Suppose without loss of generality that $x_\mathrm{i} = q$. If $F(x)_\mathrm{i} = q$ we are done because then $(F^t(x)_\mathrm{i})_{t\in\mathbb{N}}$ is constant equal to q. Otherwise let q' be the minimum state appearing in the sequence $(F^t(x)_\mathrm{i})_{t\in\mathbb{N}}$. Necessarily $q' < q$, and using again Lemma 5.14 with projection $\pi^{q'}$ we deduce that if $F^t(x)_\mathrm{i} = q'$ then $F^{t+2}(x)_\mathrm{i} = q'$. With our assumption that $x_\mathrm{i} = q$ it must be the case that $F^{2k+1}(x)_\mathrm{i} = q'$ for some $k \geq 0$. From the previous facts and periodicity of the orbit of x we deduce that $F^t(x)_\mathrm{i}$ is q when t is even and q' when t is odd. The claim about non-universality follows from Theorem 2.18. □

5.4.1 Block sequential update schemes

We have shown that MIN-MAX networks are very limited under the parallel update mode. We now consider them under block sequential update schedules. We actually show that AND-OR networks under such update modes can simulate AND-NOT networks and therefore inherit the universality property. Since MIN-MAX networks on any alphabet Q with $|Q| > 1$ simulate Boolean AND-OR networks (by just restricting their alphabet to size 2), we only focus on AND-OR networks.

Strong universality

As we did for $\mathcal{F}^{\text{BLOCK},b}_{\text{SIGN-SYM-CONJ}}$ we will show that $\mathcal{F}^{\text{BLOCK},b}_{\text{AND-OR}}$ is also strongly universal as a direct consequence of the fact that we can simulate $\mathcal{F}^{\text{BLOCK},b}_{\text{SIGN-SYM-CONJ}}$ in linear space and constant time. We accomplish this by using again the coding trick of "double railed logic". In fact, given the interaction graph of a signed symmetric conjunctive network having n nodes, we simply double each node and thus, our simulator has $2n$ nodes. Simulation is made in real time so $T = 1$. We precise these ideas in the following lemma:

Lemma 5.16 *Let $\mathcal{F}^{\text{PER},p}_{\text{AND-OR}}$ be the family of AND-OR networks updated according to some arbitrary periodic update scheme of period p. Let T be the constant function equal to 1 and $S : \mathbb{N} \to \mathbb{N}$ be defined as $S(n) = 2n$. Then, we have that $\mathcal{F}^{\text{PER},p}_{\text{SIGN-SYM-CONJ}} \preccurlyeq^S_T \mathcal{F}^{\text{PER},p}_{\text{AND-OR}}$.*

Proof. Let us a fix a periodic update scheme of period $p \in \mathbb{N}$. Let us take the graph representation of some arbitrary network (G, λ, ρ) in $\mathcal{F}^{\text{PER},p}_{\text{SIGN-SYM-CONJ}}$. We construct a network in $\mathcal{F}^{\text{PER},p}_{\text{AND-OR}}$ capable to simulate the latter network in the following way: first, we represent each state $q \in \{0, 1\}$ by $(q, \overline{q})$ where $\overline{q} = 1 - q$; then, for each $v \in V(G)$ we consider two nodes $v', \overline{v}$. By doing this, we store the original state of v in v' and $\overline{v}$ stores its complement. More precisely, we replace each node in the network by the gadget in Figure 5.18. As it is shown in the same figure, we define $\lambda'(v') \equiv$ AND and $\lambda'(\overline{v}) \equiv$ OR. Let us define the set $V' = \{(v', \overline{v}) : v \in V\}$. We define a set of edges in V' denoted E' as follows: for each edge (u, v) in $E(G)$ we add the following edges depending on $\rho((u, v))$:

- if $\rho((u,v)) = \mathrm{Id}$, we add the edges (u', v') and $(\overline{u}, \overline{v})$.
- if $\rho((u,v)) = \mathrm{Switch}$, we add the edges $(u', \overline{v})$ and $(\overline{u}, v')$.

Note that this immediately defines a local rule for the simulator network that we call f'_w for each $w \in V'$. We show that this local rule effectively simulates (G, λ, ρ). More precisely, let us call f_u the local rule of (G, λ, ρ) for each $u \in V(G)$. In addition, let us call $N(u)_+$ to the neighborhood of u in G such that any $v \in N(u)_+$ satisfies $\rho((u,v)) = \mathrm{Id}$ and also let us define $N(u)_-$ such that $v \in N(u)_-$ if and only if $\rho((u,v)) = \mathrm{Switch}$. Finally, for each $x \in \{0,1\}^n$ let us call $x' \in \{0,1\}^{2n}$ the configuration defined by $x'_{u'} = x_u$ and $x'_{\overline{u}} = \overline{x_u}$, for each $u \in V(G)$. Fix $u \in V$, we have for $(u', \overline{u})$ the following result:

- $f'_{u'}(x'|_{N(u')}) = (\bigwedge_{v' \in N_{G'}(u') : v \in N(u)_+} x'_{v'}) \wedge (\bigwedge_{\overline{v} \in N_{G'}(u') : v \in N(u)_-} x'_{\overline{v}}) = (\bigwedge_{v \in N(u)_+} x_v) \wedge (\bigwedge_{v \in N(u)_-} \overline{x}_v) = f_u(x)$
- $f'_{\overline{u}}(x'|_{N(\overline{u})}) = (\bigvee_{\overline{v} \in N_{G'}(\overline{u}) : v \in N(u)_+} x'_{\overline{v}}) \vee (\bigvee_{v' \in N_{G'}(\overline{u}) : v \in N(u)_-} x'_{v'}) = (\bigvee_{v \in N(u)_+} \overline{x}_v) \vee (\bigvee_{v \in N(u)_-} x_v) = \overline{(\bigwedge_{v \in N(u)_+} x_v) \wedge (\bigwedge_{v \in N(u)_-} \overline{x}_v)} = \overline{f_u(x)}$

And thus, we have that $x'_{u'} \to f_u(x)$ and $x'_{\overline{u}} \to \overline{f_u(x)}$.

Finally, in order to be coherent with the given periodic update scheme, it suffices to define an update scheme with period p such that the nodes $(u', \overline{u})$ are updated at each time step at which $u \in V(G)$ is. We conclude that the network $(G', \lambda', \rho') \in \mathcal{F}^{\mathrm{PER},p}_{\text{AND-OR}}$ (where $\rho(u,v) = \mathrm{Id}$ for each $(u,v) \in E$), simulates (G, λ, ρ) in constant time $T = 1$ and linear space $S(n) = 2n$. □

Theorem 5.17 *There exists some $b > 0$ such that the familly $\mathcal{F}^{\mathrm{BLOCK},b}_{\text{AND-OR}}$ of all AND-OR networks under block sequential update scheme of block parameter b is strongly universal. In particular, $\mathcal{F}^{\mathrm{BLOCK},b}_{\text{AND-OR}}$ is dynamically and computationally complex.*

Proof. The result is a direct consequence of Lemma 5.16 and strong universality of $\mathcal{F}^{\mathrm{BLOCK},b}_{\text{SIGN-SYM-CONJ}}$ given by Corollary 5.13. □

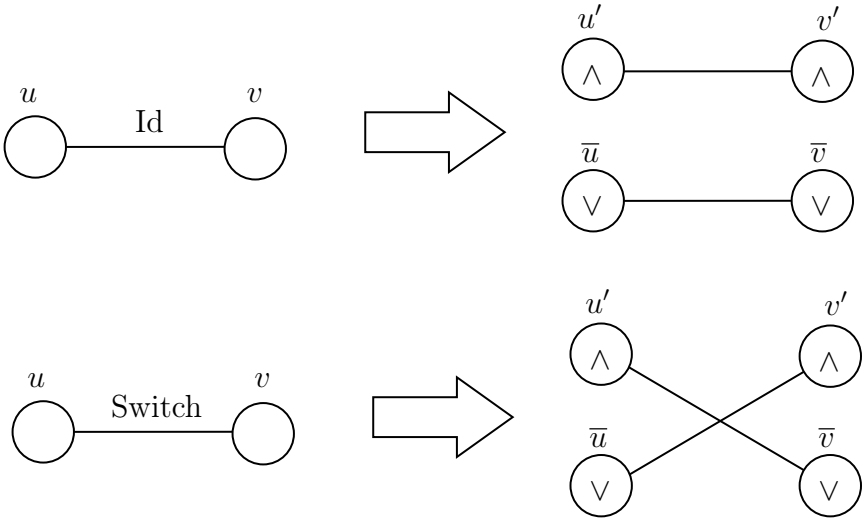


Figure 5.18: Gadget used for simulation of a symmetric signed conjunctive network with arbitrary periodic update scheme implemented over an AND-OR network with periodic update scheme.

Chapter 6

Freezing dynamics

The main aim of this section is to study the dynamics of freezing automata networks. Roughly, a freezing automata network is an automata network in which the alphabet is equipped with a partial order and the global function is non-decreasing for this order. This section is organized in two subsections:

1. in the first subsection a general framework that captures several classical dynamical decision problems is defined. In particular, this task is accomplished by introducing a model checking problem called SPEC, which asks for the existence of an orbit of a freezing automata network that satisfies certain local constraints in the trace of each node. In addition, a set of three main parameters are identified, grouped in two categories: interaction graph parameters (maximum degree and treewidth of interaction graph) and the alphabet size. In this context, a fast parallel algorithm to solve general SPEC problem is presented. Then, it is shown that latter decision problem, when parametrized using the previous parameter set is not fixed parameter tractable **W[2]** = **FPT** (which is considered unlikely). Finally, the effect of treewidth parameter is studied by considering latter problem restricted to families of sufficiently large treewidth. More precisely, it is shown that in this context latter classical dynamical decision problems are complete in their respective classes.
2. The second subsection goes further in the study of non-deterministic freezing automata networks by considering a disease spread model based in the well-known Bootsrap Percolation model. We particularly address the problem of computing the probability of contagion for some fixed node in the network, and evaluate how the network topology impacts the computational complexity of the latter task (addressed as a counting problem). In this context, the previous problem is seen as a counting version of the prediction problem. Thus, a parallel between it and previous results on the prediction problem in the context of this model is proposed.

6.1 Specification checking problem: a canonical model checking problem to capture many classical dynamical problems.

The present subsection aims at understanding what are the key parameters which influence the overall computational complexity of finite freezing automata networks. A natural first parameter is the alphabet size, since automata networks are usually considered as simple machines having a number of states that is independent of the size of the network. In addition, in the case of freezing dynamics, the alphabet is also a bound for the number of changes that some node can exhibit in an arbitrary orbit. For the same reasons, a second parameter that we consider is the maximum degree of the network, as a simple machine might not be able to handle the information incoming from a large number of neighbors. The last parameter is inspired by the results which show some computational limitations between freezing automata networks defined over bi-dimensional grids and one-dimensional grids [66, 46, 77] (i.e. paths or rings). Since Courcelle's theorem on MSO properties [17], graph parameters like treewidth [71] are used to measure a sort of distance to a tree graph and indirectly an indication of the largest grid minor structure. Indeed, it is known that paths or rings have constant treewidth, and the treewidth of a graph is polynomially related to the size of its largest grid minor [16]. Therefore treewidth is a natural parameter for our study.

The present subsection is divided in several subparts organized as follows:

Localized trace properties. We define a general model checking problem, called SPEC, that asks whether a given freezing automata network has an orbit that satisfies a given set of local constraints on the trace at each node (Problem 6.1.1). It takes advantage of the sparse orbits of freezing automata networks (a bounded number of changes per node in any orbit) which allow to express properties in the temporal dimension in an efficient way. We show thanks to a kind of pumping lemma on orbits (Lemma 6.2) that it captures many standard problems in automata network theory, among which we consider four ones: prediction [33, 31, 46], nilpotency [70, 27, 51], predecessor [52, 45] and asynchronous reachability [21]. Note that since Boolean circuits are easily embedded into freezing automata networks, our framework also includes classical problems on circuit: circuit value problem is a sub-problem of our prediction problem (see Theorem 6.21) and SAT is a sub-problem of our nilpotency problem (see Remark 6.4 and Theorem 6.19).

Fast parallel algorithm. We present a **NC** algorithm that solves our general model checking problem SPEC on any freezing automata network with bounded number of states and graph with bounded degree and bounded treewidth (Theorem 6.13). It solves in particular the four canonical problems above in **NC** for such graphs, as well as circuit value problem and SAT (see [73, 11] for better known results for these specific problems). Note that our algorithm is uniform in the sense that, besides the graph, both the automata network rule and the constraint to test are part of the input and not hidden in an expensive pre-processing step. As suggested above, temporal traces of the evolution of a bounded set of nodes have a space efficient representation. However, it is generally hard to distinguish real orbits projected on a set of nodes from locally valid sequences of states that respect the transition rule for these

nodes. Our algorithm exploits bounded treewidth and bounded degree to solve this problem via dynamic programming for any finite set of nodes. In the deterministic case, our algorithm can completely reconstruct the orbit from the initial configuration.

Hardness results. In light of Courcelle's theorem, one might think that our algorithm solving SPEC could be directly obtained (or even improved) by simply expressing the problem in MSOL (monadic second order logic) [17]. We show that it is impossible, unless **W[2]** = **FPT** (which is unlikely). More precisely, we show that the version of our model checking problem where treewidth is the unique parameter and alphabet and degree are fixed is **W[2]**-hard (Corollary 6.16), and thus it is not believed to be fixed parameter tractable. We obtain a similar result on a slight variation of our problem when the alphabet is the unique parameter and treewidth and degree are fixed (Corollary 6.17). In addition, one might also interpret SPEC as a particular instance of the Constraint Satisfaction Problem (CSP) which consists in some sort of generalization of SAT that considers a more versatile variable constraints. However, as we remark in the next sections, this approach does not provide any improvements compared to our algorithm.

Finally, we show that the four problems mentioned before, (namely prediction, nilpotency, predecessor and asynchronous reachability) are complete in their respective class (respectively **P**-complete and **coNP**-complete for the first two, and **NP**-complete for the last one) when we restrict the input graphs to a constructible family of sufficiently large treewidth (Theorems 6.19, 6.20 and 6.21). To do so, we depend on an efficient algorithm to embed arbitrary (but polynomially smaller) digraph into our input graph (Lemma 6.18), which relies on polynomial perfect brambles that can be efficiently found in graphs with polynomially large treewidth [55] (here by polynomial we mean $\Omega(n^\alpha)$ for some positive real number α). This embedding allows to simulate a precise dynamics on the desired digraph inside the input graph and essentially lifts us from the graph family constraint as soon as the treewidth is large enough. Moreover, for problems prediction (Theorem 6.21), predecessor and asynchronous reachability (Theorem 6.20) we achieve the hardness result with a fixed uniform set-defined rule (*i.e.* a rule that change the state of each node depending only on the set of states seen in the neighborhood) which is not part of the input. This shows that there is a uniform isotropic universally hard rule for these problems, which makes sense for applications like bootstrap percolation, epidemic propagation or cristal growth where models are generally isotropic and spatially uniform.

6.1.1 Localized Trace Properties

In this section we formalize the general decision problem we consider on our dynamical systems. Freezing automata networks have temporally sparse orbits, however the set of possible configurations is still exponential. Our formalism takes this into account by considering properties that are spatially localized but without restriction in their temporal expressive power. More precisely, we introduce the concept of a specification.

Definition 6.1 *Let t be a natural number and $\mathcal{A} = (G = (V, E), F)$ a non-deterministic freezing automata network in some partially-ordered alphabet Q. A $(Q, t, \mathcal{A})$-*specification *(or simply a t-*specification *when the context is clear) is a function $\mathcal{E}_t : V \to \mathcal{P}(Q^t)$ such that,*

for every $v \in V$, the sequences in $\mathcal{E}_t(v)$ are non-decreasing.

Pumping lemma on orbits. The following lemma shows that for all freezing automata networks the set of orbits of any length restricted to a set of nodes is determined by the set of orbits of fixed (polynomial) length restricted to these nodes. Moreover, if the set of considered nodes is finite, then the fixed length can be chosen linear.

Lemma 6.2 *Let Q be an alphabet, V a set of nodes with $|V| = n$ and $U \subseteq V$. Let $L = |U||Q|(|Q|n + 1)$. Then if two non-deterministic freezing automata networks over Q^V have the same set of orbits restricted to U of length L then they have the same set of orbits restricted to U of any length.*

Proof. Any orbit restricted to U of any length can be seen as a sequence of elements of Q^U and, since the considered automata network is freezing, there are at most $|U||Q|$ changes in this sequence so that it can be written $p_1^{t_1} p_2^{t_2} \cdots p_m^{t_m}$ with $m \leq |U||Q|$, $p_{\mathrm{i}} \in Q^U$ and $t_{\mathrm{i}} \in \mathbb{N}$. The key observation is that $p_1^{t_1} p_2^{t_2} \cdots p_{\mathrm{i}-1}^{t_{\mathrm{i}-1}} p_{\mathrm{i}}^{|Q|n+1} p_{\mathrm{i}+1}^{t_{\mathrm{i}+1}} \cdots p_m^{t_m}$ is a valid restricted orbit if and only if $p_1^{t_1} p_2^{t_2} \cdots p_{\mathrm{i}-1}^{t_{\mathrm{i}-1}} p_{\mathrm{i}}^{T} p_{\mathrm{i}+1}^{t_{\mathrm{i}+1}} \cdots p_m^{t_m}$ is a valid restricted orbit for all $T \geq |Q|n + 1$: this is because any sequence of $|Q|n + 1$ configurations in any orbit must contain two consecutive identical configurations since $|Q|n$ is the maximal total number of possible state changes. From this we deduce that it is sufficient to know all the restricted orbits of the form $p_1^{t_1} p_2^{t_2} \cdots p_m^{t_m}$ with $t_{\mathrm{i}} \leq |Q|n + 1$ and $m \leq |U||Q|$ to know all restricted orbits of any length. The lemma follows. □

Note that as a consequence of the last lemma, for any freezing non-deterministic automata network it suffices to consider t-specifications with t being linear in the size of the interaction graph defining the network.

Specification checking problem.

We observe that the number of possible t-specifications can be represented in polynomial space in the size of the network (as a Boolean vector indicating the allowed t-specifications). Also, in the absence of explicit mention, all the considered graphs will have bounded degree Δ by default, so a freezing automata network rule can be represented as the list of local update rules for each node which are maps of the form $Q^{\Delta} \to \mathcal{P}(Q)$ whose representation as a transition table is of size $O\big(|Q|^{\Delta+1}\big)$.

The specification checking problem we consider asks whether a given freezing automata network verifies a given localized trace property on the set of orbits whose restriction on each node adheres to a given t-specification. To this end, we introduce the concept of a satisfiable t-specification.

Definition 6.3 *Let $\mathcal{A} = (G, F)$ be a non-deterministic automata network and let $\mathcal{E}_t$ a t-specification. We say that $\mathcal{E}_t$ is satisfiable by $\mathcal{A}$ if there exists an orbit $O \in \mathcal{O}(\mathcal{A}, t)$ such that $O_v \in \mathcal{E}_t(v)$ for every $v \in V$.*

If $\mathcal{E}_t$ is a satisfiable t-specification for some automata network $\mathcal{A}$ we write $\mathcal{A} \models \mathcal{E}_t$. We present now the *Specification checking problem* as the problem of verifying whether a given t-specification is satisfiable by some automata network $\mathcal{A}$.

Problem (Specification checking problem (SPEC))

Parameters: alphabet Q, family of graphs $\mathcal{G}$ of max degree Δ.

Input:

1. a non-deterministic freezing automata network $\mathcal{A} = (G, F)$ on alphabet Q, with set of nodes V and $G \in \mathcal{G}$;
2. a time $t \in \mathbb{N}$.
3. a t-specification $\mathcal{E}_t$

Question: $\mathcal{A} \models \mathcal{E}_t$?

We remark that it could be also possible to present some sort of universal version of the last problem, in which we could ask not only if the given t-specification $\mathcal{E}_t$ is satisfiable in the sense of checking for the **existence** of some orbit of the system verifying some property coded in $\mathcal{E}_t$ but checking if **every** orbit verifies the latter property.

Four canonical problems.

When studying a dynamical system, one is often interested in determining properties of the future state of the system given its initial state. In the context of deterministic automata networks, various decision problems have been studied where a question about the evolution of the dynamics at a given node is asked. Usually, the computational complexity of such problems is compared to the complexity of simulating step by step the automata network. Roughly speaking, one can observe that some systems are complex in some way if the complexity of latter problems are “as hard" as simply simulating the system.

Problem (Prediction problem)

Parameters: alphabet Q, family of graphs $\mathcal{G}$ of max degree Δ

Input:

1. a deterministic freezing automata network $\mathcal{A} = (G, F)$ on alphabet Q, with set of nodes V with $n = |V|$ and $G \in \mathcal{G}$;
2. an initial configuration $c \in Q^n$;
3. a node $v \in V$ and a time $t \in \mathbb{N}$;
4. A t-specification $\mathcal{E}_t$ satisfying: for all $y \in \mathcal{E}_t(v), y_0 = c_v$.

Question: $\mathcal{A} \models \mathcal{E}_t$

Note that this prediction problem is clearly a subproblem of SPEC. Also, observe that a specification allows us to address various questions considered in the literature: what will be the state of the node at a given time [31, 46], will the node change its state during the evolution [33, 35, 34], or, thanks to Lemma 6.2, what will be state of the node once a fixed point is reached [66, Section 5]. Note that the classical circuit value problem for Boolean circuits easily reduces to the prediction problem above when we take G to be the DAG of the Boolean circuit and choose local rules at each node that implement circuit gates. Theorem 6.21 below gives a much stronger result using such a reduction where the graph and the rule are independent of the circuit.

Now we turn to the classical problem (in the context of deterministic automata networks) of finding predecessors back in time to a given configuration [52, 44].

Problem (Predecessor Problem)

Parameters: alphabet Q, family of graphs $\mathcal{G}$ of max degree Δ

Input:

1. a deterministic freezing automata network $\mathcal{A} = (G, F)$ on alphabet Q, with set of nodes V with $n = |V|$ and $G \in \mathcal{G}$;
2. a configuration $c \in Q^V$
3. a time $t \in \mathbb{N}$

Question: $\exists y \in Q^V : F^t(y) = c$?

Note that, analogously to the previous case, the final configuration in the input can be given through a particular t-specification $\mathcal{E}_t$, such that for all $y \in \mathcal{E}_t(v) : y_t = c$ for any $v \in V$. Thus, by considering $\mathcal{E}_t$ we can see predecessor problem as a subproblem of SPEC.

Deterministic automata networks have ultimately periodic orbits. When they are freezing, any configuration reaches a fixed point. Nilpotency asks whether their is a unique fixed point whose basin of attraction is the set of all configurations. It is a fundamental problem in finite automata networks theory [70, 27] as well as in cellular automata theory where the problem is undecidable for any space dimension [51], but whose decidability depends on the space dimension in the freezing case [66].

Problem (Nilpotency problem)

Parameters: alphabet Q, family of graphs $\mathcal{G}$ of max degree Δ

Input: a deterministic freezing automata network $\mathcal{A} = (G, F)$ on alphabet Q, with set of nodes V and $G \in \mathcal{G}$;

Question: is there $t \geq 1$ such that $F^t(Q^V)$ is a singleton?

In this case, it is not clear that Nilpotency is actually a subproblem of SPEC. However, we will show that we can solve Nilpotency by solving a polynomial amount of instances (linear on the size of the interaction graph of the network $|G|$) of SPEC in parallel. More precisely, we show that there exist a **NC** Turing reduction of NIL to SPEC. In order to do that, first, note that we can use Lemma 6.2 to fix $t = \lambda(n)$, where $\lambda(n)$ is an appropriate polynomial. Then, we express that $F^{\lambda(n)}(Q^V)$ is a singleton as the following formula, which intuitively says that for each node there is a state such that all orbits terminate in that state at this node: $\bigwedge_{s\in V} \bigvee_{q_0\in Q} \mathcal{A} \models \mathcal{E}^{q_0,s}_{\lambda(n)}$, where $\mathcal{E}^{q_0,s}_{\lambda(n)}$ are $\lambda(n)$-specifications satisfying $\mathcal{E}^{q_0,s}_{\lambda(n)}(v) = Q^t$, for every $v \neq s$ and $\mathcal{E}^{q_0,s}_{\lambda(n)}(s)$ is the set of orbits y such that $y_{\lambda(n)} = q_0$. The reduction holds.

It is straightforward to reduce coloring problems (does the graph admit a proper coloring with colors in Q) and more generally tilings problems to nilpotency using an error state that spread across the network when a local condition is not satisfied (note that tiling problem are known to be tightly related to nilpotency in cellular automata [51]). Using the same idea one can reduce SAT to nilpotency by choosing G to be the DAG of a circuit computing the given SAT formula (see Theorem 6.19 below for a stronger reduction that works on any family of graphs with polynomial treewidth).

Remark 6.4 If we allow the input automata network to be associated to a graph of unbounded degree (the local rule is then given as a circuit), it is possible to reduce any SAT instance to an automata network on a star graph with alphabet $Q = \{0, 1, \varepsilon\}$ where the central node simply checks that the Boolean values on leafs represent a satisfying instance of the SAT formula and produces a ε state that spreads over the network if it is not the case. The circuit representing the update rule of each node is **NC** in this case, and the automata network is nilpotent if and only if the formula is not satisfiable.

Given a deterministic freezing automata network of global rule $F : Q^V \to Q^V$, we define the associated non-deterministic global rule F^* where each node can at each step to apply F or to stay unchanged, formally: $F^*_v(c) = \{F_v(c), c_v\}$. It represents the system F under totally asynchronous update mode.

Problem (Asynchronous reachability Problem)

Parameters: alphabet Q, family of graphs $\mathcal{G}$ of max degree Δ

Input:

1. a deterministic freezing automata network $\mathcal{A} = (G, F)$ on alphabet Q, with set of nodes V with $n = |V|$ and $G \in \mathcal{G}$;
2. an initial configuration $c_0 \in Q^V$
3. a final configuration $c_1 \in Q^V$

Question: can c_1 be reached starting from c_0 under F^*?

Note that no bound is given in the problem for the time needed to reach the target

configuration. However, Lemma 6.2 ensures that c_1 can be reached from c_0 if and only if it can be reach in a polynomial number of steps (in n). Thus this problem can be reduced to SPEC by defining a $\lambda(n)$-specification $\mathcal{E}_{\lambda(n)}$ such that for any $y \in \mathcal{E}_{\lambda(n)} : y_0 = c_0 \wedge y_{\lambda(n)} = c_1$. This bound on the maximum time needed to reach the target ensures that the problem is NP (a witness of reachability is an orbit of polynomial length). Note that the problem is PSPACE-complete for general automata networks: in fact it is PSPACE-complete even when the networks considered are one-dimensional (network is a ring) cellular automata (same local rule everywhere) [21].

6.1.2 A fast parallel algorithm for Specification Checking

In this section we present a fast-parallel algorithm for solving the Specification Checking Problem when the input graph is restricted to the family of graphs with bounded degree and treewidth. More precisely, we show that the problem can be solved by a CREW PRAM that runs in time $\mathcal{O}(\log^2(n))$ where n is the amount of nodes of the network. Thus, restricted to graphs of bounded degree and bounded treewidth, Specification Checking Problem belongs to the class **NC**.

To explain how our main algorithm solve the latter problem, we will divide it in a number of sub-routines, that can be executed efficiently in parallel. Then, we will present an NC algorithm for Specification Checking problem as a combination of these sub-routines. We begin fixing sets Q, $\mathcal{G}$, and natural numbers Δ and k. Let $\mathcal{A} = (G, \mathcal{F})$, t and $\mathcal{E}_t$ be an instance of the Specification Checking Problem, that we consider for the following definitions.

Definition 6.5 *A* locally-valid trace of a node $v \in V$ *is a function* $\alpha : N[v] \to Q^t$ *such that:*

1. $\alpha(v)_{s+1} \in F_v((\alpha(u)_s)_{u \in N[v]})$ *for all* $0 \leq s < t$,
2. $\alpha(v)$ *belongs to* $\mathcal{E}_t(v)$.

We call the set of all locally-valid traces of v *as* $LVT(v)$

Intuitively, a locally-valid trace of a vertex v is a sequence of state-transitions of all the vertices in $N[v]$ which are consistent with the local rule of v, but not necessarily consistent with the local rules of the vertices in $N(v)$. We also ask that the state-transitions of v satisfy the specification $\mathcal{E}$.

Given two finite sets A, B, and a function $f : A \to B$. We define the restriction function of f to a subset $A' \subseteq A$ as the function $f|'_A : A' \to B$ such that, for all $v \in A'$ we have that $f|'_A(v) = f(v)$.

Definition 6.6 *Let* $U \subseteq V$ *be a subset of nodes.* *A* partially-valid trace of a set of nodes $U \subseteq V$ *is a function* $\beta : N[U] \to Q^t$ *such that* $\beta|_{N[v]}$ *belongs to* $LVT(v)$ *for each* $v \in U$.

We call the set of all partially-valid traces of U *as* $PVT(U)$.

Roughly, a partially-valid trace for a set U is a sequence of state-transitions of all the

vertices in $N[U]$, which are consistent with the local rules of all vertices in U, but not necessarily consistent with the local-rules of the vertices in $N(U)$.

Let $(W, F, \{X_w : w \in W\})$ be a rooted binary-tree-decomposition of the graph G with root r, that we assume of width at most $(3\,\mathrm{tw}(G) + 2)$. For $w \in W$, we call T_w the set of all the descendants of w, including w.

Our algorithm consists in a dynamic programming scheme over the bags of the tree. First, we assume that $PVT(X_w)$ is nonempty for all bags $w \in W$, otherwise the answer of the Specification Checking problem is false. For each bag $w \in T$ and $\beta^w \in PVT(X_w)$ we call $\mathrm{Sol}_w(\beta^w)$ the partial answer of the problem on the vertices contained bags in T_w, when the locally-valid traces of the vertices in X_w are induced by β^w. We say that $\mathrm{Sol}_w(\beta^w) = \mathbf{accept}$ when it is possible to extend β^w into a partially-valid trace of all the vertices in bags of T_w, and **reject** otherwise. More precisely, if w is a leaf of T, we define $\mathrm{Sol}_w(\beta^w) = \mathbf{accept}$ for all $\beta^w \in PVT(X_w)$. For the other bags, $\mathrm{Sol}_w(\beta^w) = \mathbf{accept}$ if and only if there exists a $\beta \in PVT(\bigcup_{z \in T_w} X_z)$ such that $\beta(u) = \beta^w(u)$, for all $u \in X_w$. Observe that the instance of the Specification Checking problem is accepted when there exists a $\beta^r \in PVT(X_r)$ such that $\mathrm{Sol}_r(\beta^r) = \mathbf{accept}$ where r is the root of the tree. The following lemma is the core of our dynamic programming scheme:

Lemma 6.7 *Let w be a bag of T that is not a leaf and $\beta^w \in PVT(X_w)$. Then $Sol_w(\beta^w) =$ **accept** if and only if for each child v of w in T_w there exists a $\beta^v \in PVT(X_v)$ such that*

1. *$\beta^w(u) = \beta^v(u)$ for all $u \in N[X_w] \cap N[X_v]$,*
2. *$Sol_v(\beta^v) =$ **accept***

Proof. First, let us assume that $\mathrm{Sol}_w(\beta^w) = \mathbf{True}$ and let v be one of the children of w in T_w. This implies that there exists a partially-valid trace $\beta \in PVT(\bigcup_{z \in T_w} X_z)$ such that $\beta(u) = \beta^w(u)$, for all $u \in X_w$. Observe that $N[\cup_{z \in T_v} X_z] \subseteq N[(\bigcup_{z \in T_w} X_z]$. Since β is defined over $N[(\bigcup_{z \in T_w} X_z]$, we can define β^v and β^{T_v} as the restrictions of β to the sets $N[X_v]$ and $N[\cup_{z \in T_v} X_z]$, respectively. Observe that β^v satisfies the condition (1) and (2) because, by definition, $\beta^v(u)$ and $\beta^w(u)$ are both equal to $\beta(u)$ for all $u \in N[X_w] \cap N[X_v]$. Moreover, $\mathrm{Sol}_{w_L}(\beta^v) = \mathbf{accept}$ because β^{T_v} is a partially-valid trace of $\bigcup_{z \in T_v} X_z$ such that $\beta^v(u) = \beta(u) = \beta^{T_v}(u)$ for each $u \in X_v$.

Conversely, suppose that we have that conditions (1), (2) for each child of w. If w is a leaf the proposition is trivially true. Suppose then that w is not a leaf. For each child v of w, let β^v be the partially-valid trace of X_v satisfying that $\mathrm{Sol}_v(\beta^v) = \mathbf{accept}$ and $\beta^v(u) = \beta^w(u)$ for each $u \in N[X_w] \cap N[X_v]$. Since $\mathrm{Sol}_v(\beta^v) = \mathbf{accept}$ we know that β^v can be extended into a partially-valid trace of $\cup_{z \in T_v} X_z$, that we call β^{T_v}. Let us call v_1 and v_2 the children of w. We define then the function $\beta : N[\cup_{z \in T_w} X_z] \to Q^t$.

$$\beta(u) = \begin{cases} \beta^w(u) & \text{if} \quad u \in N[X_w] \\ \beta^{T_{v_1}}(u) & \text{if} \quad u \in N[\bigcup_{z \in T_{v_1}} X_z] \\ \beta^{T_{v_2}}(u) & \text{if} \quad u \in N[\bigcup_{z \in T_{v_2}} X_z] \end{cases}$$

We claim that there is no ambiguity in the definition of β. First, we claim that $N[\bigcup_{z\in T_{v_1}} X_v]\cap N[\bigcup_{z\in T_{v_2}} X_v]$ is contained in $N[X_u]$. Indeed, let u be a vertex in $N[\bigcup_{z\in T_{v_1}} X_z]\cap N[\bigcup_{z\in T_{v_2}} X_z]$. There are three possibilities:

- u belongs to a bag in T_{v_1} and to another bag in T_{v_2} . In this case necessarily $u \in X_w$, because otherwise the bags containing u would not induce a (connected) subtree of T.
- u is not contained in a bag of T_{v_1}. Since u belongs to $N[\bigcup_{z\in T_{v_1}} X_z]$, there exists a vertex $\tilde{u}$ adjacent to u and contained in a bag of T_{v_1}. Note that X_w contains $\tilde{u}$, because otherwise all the bags containing $\tilde{u}$ would be in T_{v_1}. Then, no bag would contain both u and $\tilde{u}$. That contradicts the property of a tree-decomposition that states that for each edge of the graph G, there must exist a bag containing both endpoints. We deduce $\tilde{u}$ is contained in X_w and then u is contained in $N[X_w]$.
- u is not contained in a bag of T_{v_2}. This case is analogous to the previous one.

Following an analogous argument, we deduce that $N[\bigcup_{z\in T_{v_1}} X_z] \cap N[X_w]$ is contained in $N[X_{v_1}]$ and that $N[\bigcup_{z\in T_{v_2}} X_z] \cap N[X_w]$ is contained in $N[X_{v_2}]$. We deduce that β is well defined. Moreover, β is a partially-valid trace of $\bigcup_{z\in T_w} X_z$ which restricted to $N[X_w]$ equals β^w. We conclude that $\mathrm{Sol}_w(\beta^w) = \mathbf{accept}$. □

In order to solve our problem efficiently in parallel, we define a data structure that allows us to efficiently encode locally-valid traces and partially-valid traces. More precisely, in $N[v]$ there are at most $|Q|^\Delta$ possible state transitions. Therefore, when t is comparable to n, most of the time the vertices in $N[v]$ remain in the same state. Then, in order to efficiently encode a trace, it is enough to keep track only of the time-steps on which some state-transition occurs.

Let U be a set of vertices of G. A *(U,t)-sequence* $\mathcal{S}$ is a function $\mathcal{S} : U \to Q^t$ such that the sequence $\mathcal{S}(u)$ is non-decreasing, for all $u \in U$. For each $0 \le s \le t$ let us call $\mathcal{S}_s$ the sequence $(\mathcal{S}(u)_s)_{u\in U} \in |Q|^{|U|}$. Let $\mathsf{Times}(\mathcal{S}) = (t_0, t_1, \ldots, t_\ell)$ be the strictly increasing sequence of minimum length satisfying that $\mathcal{S}_{t_\mathrm{i}} = \mathcal{S}_s$ for each $t_\mathrm{i} \le s < t_{\mathrm{i}+1}$ and each $0 \le \mathrm{i} < \ell$. Observe that $t_0 = 0$ and $\ell = \ell(\mathcal{S}) \le |Q|^{|U|}$. For natural numbers m and ℓ, let us call $\langle m\rangle_\ell$ the binary representation of m using ℓ bits, padded with $\ell - \lceil \log m\rceil$ zeros when $\ell > \lceil \log m\rceil$.

Definition 6.8 *Let $\mathcal{S}$ be a (U,t)-sequence. A* succinct representation of $\mathcal{S}$, *denoted $\varepsilon(\mathcal{S})$, is a pair* $(\textsc{Times}(\mathcal{S}), \textsc{States}(\mathcal{S}))$ *such that:*

- $\textsc{Times}(\mathcal{S})$ *is a list of elements of* $\{0,1\}^{\lceil\log(t+1)\rceil}$ *of length* $|Q|^{|U|}$, *such that*

$$\textsc{Times}(\mathcal{S})_\mathrm{i} = \begin{cases} \langle t_\mathrm{i}\rangle_{\lceil\log(t+1)\rceil} & \text{if } \mathrm{i} \le \ell(\mathcal{S}) \\ \langle t\rangle_{\lceil\log(t+1)\rceil} & \text{if } \mathrm{i} > \ell(\mathcal{S}) \end{cases}$$

- $\textsc{States}(\mathcal{S})$ *is a matrix of elements of* $\{0,1\}^{\lceil\log|Q|\rceil}$ *of dimensions* $|Q|^{|U|}\times|U|$, *such that, if we call* $u_1, \ldots, u_{|U|}$ *the vertices of U sorted by their labels, then:*

$$\textsc{States}(\mathcal{S})_{\mathrm{i},j} = \begin{cases} \langle \mathcal{S}(u_j)_{t_\mathrm{i}}\rangle_{\lceil\log|Q|\rceil} & \text{if } \mathrm{i} \le \ell(\mathcal{S}) \\ \langle \mathcal{S}(u_j)_t\rangle_{\lceil\log|Q|\rceil} & \text{if } \mathrm{i} > \ell(\mathcal{S}) \end{cases}.$$

We also call $\#(U,t) = |Q|^{|U|}\lceil \log(t+1)\rceil + |U||Q|^{|U|}\lceil \log |Q|\rceil$

Observe that $\varepsilon(\mathcal{S})$ can be written using exactly $N = \#(U,t)$ bits. In other words, the succinct representations of all (U,t)-sequences can be stored in the same number of bits, which is $\mathcal{O}(|U|\log t)$. Therefore, there are at most $2^N = t^{g(|U|)}$ possible (U,t)-sequences, for some function g exponential in $|U|$. Moreover, we identify the succinct representation of (U,t)-sequence $\mathcal{S}$ with a number $x \in \{0\ldots,2^N\}$, such that $\varepsilon(\mathcal{S}) = \langle x\rangle_N$.

The restriction of $\mathcal{E}_t$ to the nodes in U is denoted $\mathcal{E}_t(U)$. When $U = \{u\}$ we denote $\mathcal{E}_t(\{u\})$ simply $\mathcal{E}_t(u)$.

Definition 6.9 *Let U be a set of vertices an let us call $N = \#(U,t)$. A* succinct representation *of a $\mathcal{E}_t(U)$ is a Boolean vector $\mathcal{X} = \varepsilon(\mathcal{E}_t(U))$ of length 2^N such that $\mathcal{X}_{\mathrm{i}} = \mathsf{True}$ when* i *represents the succinct representation of a (U,t)-sequence contained in $\mathcal{E}_t(U)$.*

Next lemma states that the succinct representation of a (U,t)-specification can be computed by a fast parallel algorithm.

Lemma 6.10 *For each set of vertices U, there exist a function f and CREW PRAM algorithms performing the following tasks in time $f(|U||Q|)\log n$ using $n^{f(|U||Q|)}$ processors:*

- *Given a (U,t)-sequence $\mathcal{S}$ as a $t\times|U|$ table of states in Q, compute $\varepsilon(\mathcal{S})$*
- *Given a $\mathcal{E}_t(U)$ as a list of (U,t)-sequences, compute $\varepsilon(\mathcal{E}_t(U))$*

Proof.

- The algorithm first computes $\mathsf{Times}(\mathcal{S})$. Then, it constructs the list $\textsc{Times}(\mathcal{S})$ and the matrix $\textsc{States}(\mathcal{S})$ copying the lines of $\mathcal{S}$ given in $\mathsf{Times}(\mathcal{S})$.

  The algorithm starts reserving $N = \#(U,t)$ bits of memory for the list Times and the matrix States, and $t+1$ bits of memory represented in a vector called indices. The vector $\textsc{indices}_s$ stores the time-steps on which states have changed. This information is contained in $\mathsf{Times}(\mathcal{S})$.

  For each $\mathrm{i}\in\{1,\ldots,t\}$ the algorithm initializes a processor P_{i} and assigns the i-th bit of indices to it. Processor P_{i} looks at the i-th and $\mathrm{i}-1$-th lines of $\mathcal{S}$. If $\mathcal{S}_{\mathrm{i}} \neq \mathcal{S}_{\mathrm{i}-1}$ then processor writes a 1 in $\textsc{indices}_{\mathrm{i}}$. Otherwise, the processor writes a 0 in $\textsc{indices}_{\mathrm{i}}$. Then P_{i} stops. All this process can be done in time $\mathcal{O}(|U|\log|Q|+\log t)$ per processor.

  Then, the algorithm computes the vector p of length t such that $p_j = \sum_{j=1}^{\mathrm{i}} \textsc{indices}_j$, for each $j\in\{1,\ldots,t\}$. This process can be done in time $\mathcal{O}(\log t)$ using $\mathcal{O}(t)$ processors using the prefix sum algorithm given by [50] (Proposition 1.19). Observe that if $\textsc{indices}_{\mathrm{i}} = 1$ for some index i, then $\mathrm{i} = \mathsf{Times}(\mathcal{S})_{p_{\mathrm{i}}}$. Moreover, $p_t = \ell(\mathcal{S})$.

  Once every processor $(P_{\mathrm{i}})_{0<\mathrm{i}\leq t}$ stops, the algorithms reinitialize them. For each $0 < \mathrm{i}\leq t$, each processor P_{i} looks at $\textsc{indices}_{\mathrm{i}}$. If $p_{\mathrm{i}} < p_t$ and $\textsc{indices}_{\mathrm{i}} = 0$ then processor P_{i} stops. If $p_{\mathrm{i}} = p_{\mathrm{i}-1} = p_t$ the processor stops. If $p_{\mathrm{i}} \neq p_t$ and $\textsc{indices}_{\mathrm{i}} = 1$, then the algorithm writes $\langle \mathrm{i}\rangle_{\lceil\log(t+1)\rceil}$ in $\textsc{Times}_{p_{\mathrm{i}}}$, and for each $u\in\{1,\ldots,|U|\}$ writes $\langle \mathcal{S}_{\mathrm{i},u}\rangle_{\lceil\log|Q|\rceil}$ in $\textsc{States}_{p_{\mathrm{i}},u}$. If $p_{\mathrm{i}} = p_t$ and $p_{\mathrm{i}-1} \neq p_{\mathrm{i}}$, then the processor P_{i} writes $\langle t\rangle_{\lceil\log(t+1)\rceil}$ in

$\textsc{Times}_j$ and writes $\langle \mathcal{S}_{t,u} \rangle_{\lceil \log |Q| \rceil}$ in $\textsc{States}_{j,u}$ for each $p_{\mathrm{i}} \leq j \leq |Q|^{|U|}$ and for each $u \in \{1, \ldots, |U|\}$. The algorithm writes $\textsc{Times}_0 = \langle 0 \rangle_{\lceil \log (t+1) \rceil}$ and writes $\langle \mathcal{S}_{0,u} \rangle_{\lceil \log |Q| \rceil}$ in $\textsc{States}_{j,u}$ for each $u \in \{1, \ldots, |U|\}$. All this process can be done in time $\mathcal{O}(|Q|^{|U|} \log t)$ per processor.

The algorithm returns $\varepsilon(\mathcal{S}) = (\textsc{Times}, \textsc{States})$. The whole process takes time $\mathcal{O}(|Q|^{|U|} \log t)$ and $\mathcal{O}(t)$ processors.

- The algorithm initializes $\mathcal{X} = \varepsilon(\mathcal{E}_t(U))$ as 2^N bits of memory bits, all in 0. Then, it assigns one processor $P_{\mathcal{S}}$ to each (U,t)-sequence $\mathcal{S}$ in $\mathcal{E}_t(U)$. For each $\mathcal{S} \in \mathcal{E}_t(U)$, processor $P_{\mathcal{S}}$ uses the previous algorithm to compute $y = \varepsilon(\mathcal{S})$. Then processor $P_{\mathcal{S}}$ writes $\mathcal{X}_y = 1$. Once every processor has finished, the algorithm returns $\mathcal{X}$. The whole process takes time $\mathcal{O}(\log |\mathcal{X}||Q|^{|U|} \log t)$ and uses $|\mathcal{X}| t^{\mathcal{O}(1)} = n^{\mathcal{O}(|Q|^{|U|})}$ processors. We deduce that the algorithm runs in time $\mathcal{O}(|Q|^{|U|} \log t)$ using 2^N processors.

□

Observe that if β is a partially-valid trace of U, then in particular β is a $(N[U], t)$-sequence. Therefore, there exists an $x \leq 2^N$ with $N = \#(N[U], t)$, such that $\varepsilon(\beta) = x$. In the following lemma we show how to characterize the values on $x \leq 2^N$ that are the encoding of some partially-valid trace of U. We need the following definition. Let U be a set of vertices and let $x \in \{0, \ldots, 2^N\}$, with $N = \#(U, t)$. Then we call $\textsc{Times}(x)$ and $\textsc{States}(x)$ the vector and matrix such that $x = (\textsc{Times}(x), \textsc{States}(x))$. More precisely:

- $\textsc{Times}(x)$ are the first $|Q|^{|U|}$ bits of x interpreted as sequence of elements of $\{0,1\}^{\lceil \log (t+1) \rceil}$ of length $|Q|^{|U|}$.
- $\textsc{States}(x)$ are the rest of the bits of x interpreted as the matrix of elements of $\{0,1\}^{\lceil \log |Q| \rceil}$ of dimensions $|Q|^{|U|} \times |U|$.

Lemma 6.11 *Let S be a (U,t)-sequence and $Z \subseteq U$. There is a sequential algorithm which given $\varepsilon(\mathcal{S})$ computes $\varepsilon(\mathcal{S}|_Z)$ in time linear in the size of $\varepsilon(\mathcal{S})$.*

Proof. Let $\kappa = |Q|$. The computes algorithm $\varepsilon(\mathcal{S}|_Z)$ checking each pair of lines of $\textsc{States}$ and verifying if the columns of Z differ on any coordinate, keeping only the lines on which some of the vertices in Z switches states for the first time. More precisely, let $u_1, \ldots, u_\kappa$ be the set U ordered by their labels. Let $J \in \{j_1, \ldots, j_{|Z|}\}$ be the set of indices of vertices of Z (i.e., $u_{j_q} \in Z$ for all $q \in \{1, \ldots, |Z|\}$). The algorithm computes the set L of indices $\mathrm{i} \leq |Q|^\kappa$ such that $\mathrm{i} \in L$ if and only if there exists $q \in J$ such that $\textsc{States}_{\mathrm{i},j_q} \neq \textsc{States}_{\mathrm{i}-1,j_q}$. Let $\{\mathrm{i}_1, \ldots, \mathrm{i}_{|L|}\}$ the indices in L. Observe that $|L| \leq |Q|^{|Z|}$. Then for each $p \leq |Q|^{|Z|}$ and $q \in |Z|$,

$$\textsc{Times}[Z]_p = \begin{cases} \textsc{Times}_{\mathrm{i}_p} & \text{if } p \leq |L| \\ \langle t \rangle_{\lceil \log (t+1) \rceil} & \text{if } \mathrm{i} > |L| \end{cases}$$

$$\textsc{States}[Z]_{p,q} = \begin{cases} \textsc{States}_{\mathrm{i}_p,j_q} & \text{if } p \leq |L| \\ \textsc{States}_{t,j_q} & \text{if } p > |L| \end{cases}.$$

The algorithm returns $(\textsc{Times}[Z], \textsc{States}[Z])$. □

Lemma 6.12 *Let U be a set of vertices and let $N = \#(N[U], t)$. There is a sequential algorithm which, given $x \geq 0$ and $\varepsilon(\mathcal{E}_t(u))$ for each $u \in V(G)$, decides in time $f(|N[U]||Q|)\log n$ whether x is a succinct representation of a partially-valid trace of U, where f is an exponential function.*

Proof. Let U be a set of vertices containing S and let $x \in \{0, \ldots, 2^N\}$, with $N = \#(N[U], t)$. Let $\kappa = |N[U]|$. Let $\{u_1, \ldots, u_\kappa\}$ be the vertices of $N[U]$ ordered by label. The algorithm first verifies that $x \leq 2^N$ and rejects otherwise. Then, the algorithm verifies that the pair $(\textsc{Times}, \textsc{States}) = (\textsc{Times}(x), \textsc{States}(x))$ satisfies $\textsc{Times}_0 = 0$ and that $\textsc{Times}$ and each column of $\textsc{States}$ are increasing. Otherwise, the algorithm rejects because x is not a succinct representation of a $(N[U], t)$-sequence. If the algorithm passes this test we assume that $x = \varepsilon(\mathcal{S})$ for some $(N[U], t)$-sequence $\mathcal{S}$. For a subset of vertices Z, let us call $(\textsc{Times}[Z], \textsc{States}[Z]) = \varepsilon(\mathcal{S}|_Z)$. Consider now the following conditions:

1. $\textsc{States}_{\mathrm{i},j} \in F_{u_j}(\textsc{States}[N[u_j]]_\mathrm{i})$ for each $\mathrm{i} \in \{0, \ldots, |Q|^\kappa\}$ and $j \in \{1, \ldots, \kappa\}$ such that $u_j \in U$.
2. $(\textsc{Times}[\{u_j\}], \textsc{States}[\{u_j\}])$ belongs to $\mathcal{E}_t(u_j)$ for each $j \in \{1, \ldots, \kappa\}$.

When this conditions are satisfied, we can deduce that $x = \varepsilon(\beta)$ for some partially-valid trace β of U. Indeed, as x is representation of $\mathcal{S}$, the vertices in $N[U]$ only have state-transitions of the time-steps given by the $\textsc{Times}$. Therefore, condition (1.) and (2.) imply that $\mathcal{S}|_{N[u]}$ is a locally-valid trace of u. To verify condition (1.) and (2.) we use the algorithm of Lemma 6.11 to compute $\textsc{States}[Z]$ for a given set of vertices $Z \subseteq N[U]$. Observe that the algorithm computes $\textsc{Times}[Z]$ and $\textsc{States}[Z]$ in time $\mathcal{O}(\kappa|Q|^\kappa \log n)$. The algorithm checks (1.) by looking at each row of $\textsc{States}[N[u]]$ and the column corresponding to vertex u, and the table of F_u given in the input. The algorithm verifies (2.) computing $\varepsilon(\mathcal{S}(u_j)) = (\textsc{Times}[\{u_j\}], \textsc{States}[\{u_j\}])$ and then looking at the $\varepsilon(\mathcal{S}(u_j))$-element of the table $\varepsilon(\mathcal{E}_t(u_j))$. All these processes take time $\mathcal{O}(|U|\kappa|Q|^\kappa \log n)$. Overall the whole algorithm takes time $\mathcal{O}(|U|\kappa|Q|^\kappa \log n) = f(|N[U]||Q|)\log n$. □

We are now ready to give our algorithm solving the Specification Checking problem.

Theorem 6.13 *Specification Checking problem can be solved by a CREW PRAM algorithm running in time $\mathcal{O}(\log^2 n)$ and using $n^{\mathcal{O}(1)}$ processors on graphs of bounded treewidth.*

Proof. Our algorithm consists in an implementation of the dynamic programming scheme explained at the beginning of this section. It starts computing a rooted binary-tree decomposition $(W, F, \{X_w : w \in W\}$ of the input graph using the logarithmic-space algorithm given by Proposition 1.18.

The algorithm also computes the succinct representations of $\mathcal{E}$ and $\mathcal{I}_v$ for each $v \in V$ using Lemma 6.10.

Then, the algorithm performs the dynamic programming scheme over T. Let r be the root of T. Then, for a bag $w \in W$, we define the *level of* w denoted by $\textsc{Level}(w)$, as the distance between w and the root r. There is a fast-parallel algorithm computing the level of each vertex of a tree by a EREW PRAM running in time $\mathcal{O}(\log n)$ and using $\mathcal{O}(n)$ processors [50]. Using a prefix-sum algorithm we can compute the maximum level M of a vertex, which corresponds to the leafs of the binary-tree T. For each $\mathrm{i} \in \{0, \ldots, M\}$, let $\mathcal{L}_\mathrm{i}$ the set of bags w such that $\textsc{Level}(w) = M - \mathrm{i}$.

For each $w \in W$, we represent the values of the function Sol_w as a table S^w indexed as a table of size 2^N, with $N = \mathcal{O}(|Q|^{\Delta(3\,\mathrm{tw}(G)+2+k)} \log n)$ greater that $\#(N[X_w], t)$ for all $w \in W$. Each $x \in \{0, \ldots, 2^N\}$ is interpreted as a potentially succinct encoding of a partial-valid trace β. Initially $S^w = 0^{2^N}$, which meaning that a priori we reject all $x \in \{0, \ldots, 2^N\}$. Then, our algorithm iterates in a reverse order over the levels of the tree, starting from $\mathcal{L}_0$ until reaching the the root $r \in \mathcal{L}_M$. In the i-th iteration, we compute for each bag $w \in \mathcal{L}_\mathrm{i}$ the set of all $x \in \{0, \ldots, 2^N\}$ that represent partially-valid traces $\beta^w \in PVT(X_w)$ such that $\mathrm{Sol}_w(\beta^w) = \textbf{accept}$. To do so, the algorithm uses the calculations done on the bags in $\mathcal{L}_{\mathrm{i}-1}$, and use Lemma 6.7. The algorithm saves the answer of each partial solution in a variable out consisting in $|W|$ bits, such that, and the end of the algorithm $\mathsf{out} = 1^{|W|}$ if and only the instance of the Specification Checking problem is accepted.

At the first iteration, for each $w \in \mathcal{L}_0$ the algorithm sets in parallel $S^w_x = 1$ for all x representing a partially-valid trace of w, because $\mathrm{Sol}_w(\beta^w)$ is defined to **accept** for all partially-valid trace of a leaf of T. Therefore, in parallel for all bag $w \in \mathcal{L}_0$, the algorithm runs 2^N parallel instances of the algorithm of Lemma 6.12, one for each $x \in \{0, \ldots, 2^N\}$, and for each one that is accepted, the algorithm writes $S^w_x = 1$. Once every parallel verification finishes, the algorithm sets $\mathsf{out}_w = 1$. We now detail the algorithm on the i-th iteration, assuming that we have computed S^w for all bag $w \in \mathcal{L}_{\mathrm{i}-1}$.

Let w be a vertex in $\mathcal{L}_\mathrm{i}$ and let us call w_L and w_R the children of w, which belong to $\mathcal{L}_{\mathrm{i}-1}$. Roughly, as we know the partial solutions restricted to the subtrees rooted at w_1 and w_2, the algorithm will try to extend it to a partial solution of w according to the gluing procedure given by Lemma 6.7, testing all possible combinations. More precisely, we initialize a set of $|\mathcal{L}_\mathrm{i}|$ processors $\{P^w\}_{w \in \mathcal{L}_\mathrm{i}}$, one assigned each bag in $\mathcal{L}_\mathrm{i}$. Each processor P^w verifies if $\mathsf{out}_{w_L} = \mathsf{out}_{w_R} = 1$, or stops and writes $\mathsf{out}_w = 0$. Otherwise, processor P^w initializes a set of 2^N processors, that we call $\{P^w_z\}_{z \in \{1, \ldots, 2^N\}}$, and reserves 2^N bits of memory $S^w \in \{0,1\}^{2^N}$. For each $z \in 2^N$, processor P^w_z verifies if z is a succinct representation of a partially-valid trace of X_w using Lemma 6.12. If its not the case then P^w_z stops and writes a 0 in S^w_z. Otherwise, processor P^w_z initializes $(2^N)^2$ processors $\{P^w_{z,z_R,z_L} : z_R, z_L \in \{1, \ldots, 2^N\}\}$ and reserves $2^N \times 2^N$ bits of memory $(\tilde{S}^w_z) \in \{0,1\}^N \times \{0,1\}^N$.

If $S^{w_L}_{z_L} = 0$ or $S^{w_R}_{z_R} = 0$ the processor P^w_{z,z_R,z_L} stops and writes a 0 in $\tilde{S}^w_{z,z_R,z_L}$. Otherwise, the processor P^w_{z,z_R,z_L} interprets z, z_R and z_L as $\varepsilon(\beta^w_z)$, $\varepsilon(\beta^w_{z_L})$ and $\varepsilon(\beta^w_{z_R})$, for partially-valid traces β_z, β_{w_L} and β_{w_R} of X_w, X_{w_L} and X_{w_R}, respectively. Which means that β^w_z belongs to $PVT(X_w)$ and $\mathrm{Sol}_{w_L}(\beta_{z_L}) = \mathrm{Sol}_{w_L}(\beta_{z_L}) = \textbf{accept}$. Therefore β_z is a partially-valid trace of X_w and β_{z_L} and β_{z_R} verify the condition (2) of Lemma 6.7. Up to this point, all verifications can be done in time $\mathcal{O}(N) = \mathcal{O}(|Q|^{\Delta(3\,\mathrm{tw}(G)+2+k)} \log n)$ because we are just looking at the coordinates in the given tables.

Then, the processor P^w_{z,z_L,z_R} computes sets $Y_L = N[X_w] \cap N[X_{w_L}]$ and using the algorithm of Lemma 6.11 computes $\varepsilon(\beta^w|_{Y_L})$ and $\varepsilon(\beta^{w_L}|_{Y_L})$. If $\varepsilon(\beta^w|_{Y_L}) = \varepsilon(\beta^{w_L}|_{Y_L})$ the processor deduces that $\beta^w(u) = \beta^{w_L}(u)$, for all $u \in N[X_w] \cap N[X_{w_L}]$. Then P^w_{z,z_L,z_R} computes sets $Y_R = N[X_w] \cap N[X_{w_R}]$ and using the algorithm of Lemma 6.11 computes $\varepsilon(\beta^w|_{Y_R})$ and $\varepsilon(\beta^{w_R}|_{Y_R})$. Then, if $\varepsilon(\beta^w|_{Y_R}) = \varepsilon(\beta^{w_R}|_{Y_R})$ the processor deduces that $\beta^w(u) = \beta^{w_R}(u)$, for all $u \in N[X_w] \cap N[X_{w_R}]$. If both verifications are satisfied, processor P^w_{z,z_R,z_L} stops and writes a 1 in $\tilde{S}^w_{z,z_R,z_L}$. Otherwise, the processor P^w_{z,z_R,z_L} stops and writes a 0 in $\tilde{S}^w_{z,z_R,z_L}$. All of these verifications can be executed by P^w_{z,z_L,z_R} in time $\mathcal{O}(N)$.

Once that all processors in $\{P^w_{z,z_1,z_2} : z_L, z_R \in \{1, \dots 2^N\}\}$ finished, processor P^w_z runs a prefix-sum algorithm in $\tilde{S}^w_z$, simply summing the elements of the vector to verify if some instance was accepted. If the result is different than 0, processor P^w_z writes a 1 in S^w_z, and writes a 0 otherwise. When every processor $(P^w_z)_{z\in\{1,\dots,2^N\}}$ finishes, we obtain that S^w is the table representing function Sol_w. Then processor P^w runs a prefix-sum algorithm on S^w to verify that there exists a partial solution for bag X_w. If the result of the prefix sum equals zero, processor P^w stops and writes a $\mathsf{out}_w = 0$. Otherwise, it writes $\mathsf{out}_w = 1$ and stops.

After all processors $(P^w)_{w\in\mathcal{L}_\mathrm{i}}$ have finished, the algorithm continues with the next level. When the last level is reached, before halting processor P^r decides if $\mathsf{out}_w = 1$ for all $w \in W$ using a prefix-sum algorithm. If the answer is affirmative the algorithm **accepts** the input, and otherwise **rejects**. On each level, the algorithm takes time $\mathcal{O}(\Delta|Q|^{2\Delta(3\,\mathrm{tw}(G)+2)}\log n)$ and uses $n^{\mathcal{O}(|Q|^{\Delta(3\,\mathrm{tw}(G)+2)})}$ processors. Proposition 1.18 provides a construction of a binary-tree-decomposition T of depth $\mathcal{O}(\log n)$. This means that $M = \mathcal{O}(\log n)$, and implies that the whole takes time $\mathcal{O}(\Delta|Q|^{2\Delta(3\,\mathrm{tw}(G)+2)}\log^2 n) = \mathcal{O}(\log^2 n)$ and $n^{\mathcal{O}(|Q|^{\Delta(3\,\mathrm{tw}(G)+2)})} = n^{\mathcal{O}(1)}$ processors. The correctness of the algorithm is given by Lemmas 6.7, 6.10, 6.11 and 6.12.

□

Remark 6.14 *The algorithm given in the proof of Theorem 6.13 not only computes the answer of Specification Checking problem but it also gives the coding of the orbits satisfying specification $\mathcal{E}_t$. In the case in which the freezing automata network $\mathcal{A} = (G, F)$ is deterministic, we can say a lot more using those algorithms. Giving t and an initial condition $x \in Q^n$, we are actually capable of testing any global dynamic property in **NC** provided that this property has $F^t(x)$ as input and it is decidable in **NC**. In fact, note that given an initial condition $x \in Q^n$, there is only one possible orbit for each node $v \in V(G)$. Therefore, we are able to calculate the global evolution of the system in time t starting from x.*

The proof of the previous Theorem 6.13 shows that SPEC can be solved in time $f(|Q| + \Delta(G) + \mathrm{tw}(G))\log n$ using $n^{f(|Q|+\Delta(G)+\mathrm{tw}(G))}$ processors in a PRAM machine, hence in time $n^{g(|Q|+\Delta(G)+\mathrm{tw}(G))}$ on a sequential machine, for some computable functions f and g. In other words, when the alphabet, the maximum degree and the tree-width of the input automata network are parameters, our result shows that SPEC is in **XP**. In the next section, we show that SPEC is not in **FPT**, unless **FPT**=**W[2]**.

Constraint Satisfaction Problem

We remark that problem SPEC can be interpreted as a specific instance of the Constraint Satisfaction Problem (CSP). Problem CSP is a sort of generalization of SAT into a set of more versatile variable constraint. It is formally defined as a triple (X, D, C), where $X = \{X_1, \ldots, X_n\}$ is a set of *variables*, $D = \{D_1, \ldots, D_n\}$ is a set of *domains* where are picked each variable, and a set $C = \{C_1, \ldots, C_m\}$ of *constraints*, which are k-ary relations of some set of k variables. The question is whether there exists an assignment of values to each variables in their corresponding domains, in order to satisfy each one of the constraints. As we mentioned, SPEC can be seen as a particular instance of CSP, where we choose one variable for each node of the input graph. The domain of each variable is the set of all locally valid traces of the corresponding node. Finally, we define one constraint for each node, where the variable involved are all the vertices in the close-neighborhood of the corresponding node, and the relation corresponds to the consistency in the information of the locally-valid traces involved.

Now consider an instance of SPEC with constant tree-width, maximum degree and size of the alphabet, and construct the instance of CSP with the reduction described in the previous paragraph. Then, the obtained instance of CSP has polynomially-bounded domains and constant tree-width, where the tree-width of a CSP instance is defined as the tree-width of the graph where each variable is a node, and two nodes are adjacent if the corresponding variables appear in some restriction. Interestingly, it is already known that in these conditions CSP can be solved in polynomial time [72, 18, 59] . This implies that, subject to the given restrictions, SPEC is solvable in polynomial time using the given algorithm for CSP as a blackbox.

The algorithm given in the proof of Theorem 6.13 is better than the use of the CSP blackbox in two senses. First, we obtain explicit dependencies on the size of the alphabet, maximum degree and tree-width. Second, the *Prediction Problem* is trivially solvable in polynomial time, and then the use of the CSP blackbox gives no new information for this problem. Moreover, as we mentioned in Remark 6.14, our algorithm does not only decides SPEC, but can also be used to obtain a coding of the orbit satisfying the given specification, and moreover, the possibility to test any **NC** property on a deterministic freezing automata networks.

6.1.3 W[2]-hardness results

The goal of this section is to show that, even when the alphabet and the degree are fixed and the treewidth is considered as the only parameter, the SPEC problem is **W[2]**-hard (see [23] for an introduction to the W hierarchy) and thus, it is not believed to be fixed parameter tractable. This is in contrast with classical results of Courcelle establishing that model-checking of MSO formulas parametrized by the treewidth is fixed-parameter tractable [17] (see [54, 23] to place the result in a wider context).

Lemma 6.15 *There is a fixed alphabet Q and an algorithm which, given $k \in \mathbb{N}$ and a graph G of size n, produces in time $O(k \cdot n^{O(1)})$:*

- *a deterministic freezing automata network* $\mathcal{A} = (G', \mathcal{F})$ *with alphabet* Q *and where* G' *has treewidth* $O(k)$ *and degree* 4
- *a* $O(n^2)$*-specification* $\mathcal{E}$

such that G *admits a dominating set of size* k *if and only if* $\mathcal{A} \models \mathcal{E}$.

The construction of the lemma works by producing a freezing automata network on a $O(k) \times n^2$-grid together with a specification which intuitively work as follows. A row of the grid is forced (by the specification) to contain the adjacency matrix of the graph, k rows serve as selection of a subset of k nodes of G, and another row is used to check domination of the candidate subset. The key of the construction is to use the dynamics of the network to test that the information in each row is encoded coherently as intended, and raise an error if not. The specification serves both as a partial initialization (graph adjacency matrix and tests launching are forced, but the choice in selection rows is free) and a check that no error are raised by the tests.

Proof. Let $G' = (V', E')$ be the $(k+2) \times n^2$-grid where

$$\begin{aligned} V' &= \{(\mathrm{i}, j) : 0 \leq \mathrm{i} < n^2, 1 \leq j \leq k+2\}; \text{ and} \\ E' &= \{\{(\mathrm{i}, j), (\mathrm{i} \pm 1 \bmod n^2, j)\} : 0 \leq \mathrm{i} < n^2, 1 \leq j \leq k+2\} \\ &\quad \cup \{\{(\mathrm{i}, j), (\mathrm{i}, j')\} : 0 \leq \mathrm{i} < n^2, 1 \leq j, j' \leq k+2, |j - j'| = 1\}. \end{aligned}$$

Clearly G' has a $O(k)$ treewidth. The horizontal dimension (coordinate i in the grid) should be thought as n block of size n. For each j we denote the jth row by $V_j = \{(\mathrm{i}, j) : 0 \leq \mathrm{i} < n^2\}$. The alphabet of the automata network is $Q = \{0, 1\} \times Q'$ where the $\{0, 1\}$ is the *marker component* and Q' is *verification component*. A position is said to be marked if it has a 1 in its marker component. Vertically, the network is organized as follows.

- Rows V_1 to V_k are called *selection rows* and they all have the same behavior: marking the same unique position in each block, *i.e.* having horizontal coordinates s, $s+n$, $s+2n$, ..., $s+(n-1)n$ marked for some s with $0 \leq s < n$. Intuitively the role of each selection row is to select a node among the k nodes of the candidate dominating set and ensure that the selection information is coherently spread across the n blocs.
- Row V_{k+2} is the *graph row* and its role is to hold the adjacency matrix of graph G laid out in a single row (bloc i contains the incidence vector of node i).
- Row V_{k+1} is the *domination row*. Its role is to witness that there is a position i in each bloc (possibly different from one bloc to another) where i is marked in the graph row and also marked in at least one selection row. Said differently its role is to give a certificate that the k selected nodes in the selection rows are indeed a dominating set for the graph encoded in the graph row.

The description by rows above gives some conditions on the marked position in the network. It should be clear that these conditions are satisfied if and only if the k selection rows represent k nodes of the graph G that form a dominating set.

We now complete the description of $\mathcal{A}$. In the verification component of states Q' there is a special error state. The behavior of the automata network is to perform two tests to ensure

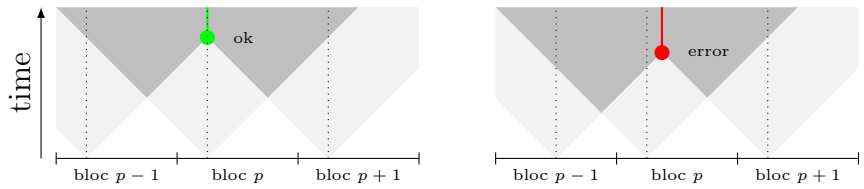


Figure 6.1: Checking that the same node is marked in each bloc in a selection row: on the left, a valid test in bloc p, on the right an invalid test in bloc p generating an error state. Dotted lines indicate the marked position in each bloc. The shades of gray indicates the state changes involved in the implementation of freezing signals: at each position the sequence of states in non-decreasing with time.

that each row is holding marks that satisfies the conditions above. If any test fails somewhere the error state is raised and stays forever. The two test run in parallel (using two independent subcomponents inside Q') and their technical implementation as a freezing automata network on graph G' is straightforward using a constant number of state component within Q'. Note that in the description below, what we call signals are freezing signals: a state change from q to q'' possibly with intermediate state q' with $q \leq q' \leq q''$ that propagates in some direction like a flame in a wick (and not a particle in state q_1 that move inside a context of q_0 like classical signals are). The tests are as follows:

- the domination test works vertically: each marked position i in the domination row checks that the position i is also marked in the neighboring graph row and then launch a signal that moves from $(\mathrm{i}, k+1)$ downto $(\mathrm{i}, 1)$ until it finds a marked position in some selection row. If the signal reaches position $(\mathrm{i}, 1)$ without having encountered any mark, then it raises an error state.
- the selection test happens in each selection row horizontally: first it checks that exactly one position is marked within each bloc (this can be done in one step by using a layer of alphabet $\{>, <\}$ and forcing the language $>^*<^+$ by forbidding the pattern $<>$ to appear in two adjacent position in a bloc); second, each marked position (in each bloc) launches a signal going left and a signal going right both moving at the speed of one position per time step. Each signal goes one, crosses a first signal going in the opposite direction, continues, and when it encounters a signal of the opposite direction for the second time it stops and checks that the position reach contains a mark. This process ensures that the position marked is the same in each bloc by comparing the distance between marks in bloc $p-1$ and bloc $p+1$ and the distance between marks in blocs p and $p+1$ for all p (see Figure 6.1).

Finally, the n-specification $\mathcal{E}$ consists in:

- forcing the initial marking of the graph row to be the actual adjacency matrix of G;
- allowing any marking in the other rows;
- forcing the Q' component to be without error at any time step;
- initializing the Q' component to properly launch the tests.

It should be clear that both automata network $\mathcal{A}$ and specification $\mathcal{E}$ can be constructed in time $O(k \cdot n^{O(1)})$ from G. The construction is such that G admits a dominating set of size

k if and only if $\mathcal{A} \models \mathcal{E}$: indeed, from the initialization imposed by $\mathcal{E}$ it would take at most n steps for any of the two test to raise an error state, so $\mathcal{E}$ ensures that there exists an initial marking that encodes a valid dominating set of size (at most) k as explained above. □

From Lemma 6.15 and $W[2]$-hardness of the k-Dominating-Set problem [22], we immediately get the following corollary.

Corollary 6.16 *The* SPEC *problem with fixed degree and fixed alphabet and with treewidth as unique parameter is* ***W[2]****-hard.*

A freezing automata network on a $O(k) \times n^2$-grid with alphabet Q can be seen as a freezing automata network on a line of length n^2 with alphabet $Q^{O(k)}$. One might therefore want to adapt the above result to show **W[2]**-hardness in the case where treewidth and degree are fixed while alphabet is the parameter. However, the specification which is part of the input, has an exponential dependence on the alphabet (a t-specification is of size $O(n \cdot t^{|Q|})$). Therefore **FPT** reductions are not possible when the alphabet is the parameter. We can circumvent this problem by considering a new variant of the SPEC problem where specification are given in a more succinct way through regular expressions. A regular (Q, V)-specification is a map from V to regular expressions over alphabet Q. We therefore consider the problem **REGSPEC** which is the same as SPEC except that the specification must be a regular specification. With this modified settings, the construction of Lemma 6.15 can be adapted to deal with the alphabet as parameter.

Corollary 6.17 *The* ***REGSPEC*** *problem with fixed degree and fixed treewidth and with alphabet as unique parameter is $W[2]$-hard.*

Proof. Using the construction of Lemma 6.15 and compressing the k rows into a single one by enlarging the alphabet, we can construct a freezing automata network $\mathcal{A}^\alpha$ of alphabet Q^{k+2} on a graph $G'' = (V'', E'')$ which is a cycle of length n^2 (therefore of constant treewidth and constant degree) and that has the same behaviour with respect to k-dominating sets of the graph G of the lemma. The local map at each node of $\mathcal{A}^\alpha$ can be described by a transition table of size $O(Q^{3k})$ so the global description is of size $O(Q^{3k} \cdot n^2)$. Noting that the specification produced in Lemma 6.15 is actually a regular specification of the form: $v \in V' \mapsto Q_{\mathrm{i},v} Q_{\mathrm{e},v}^*$ where $Q_{\mathrm{i},v}$ take care of the initialization and $Q_{\mathrm{e},v}$ is the subset of states with no error. We deduce that the corresponding regular specification for $\mathcal{A}^\alpha$ is of the form $v \in V'' \mapsto \left(Q_{\mathrm{i},v_1} \times \cdots \times Q_{\mathrm{i},v_{k+2}}\right)\left(Q_{\mathrm{e},v_1} \times \cdots \times Q_{\mathrm{e},v_{k+2}}\right)^*$. Hence its size is $O(|Q|^{O(k)} \cdot n^2)$. The total size of the input produced for problem **REGSPEC** is therefore also $O(|Q|^{O(k)} \cdot n^2)$ and it can be produced in time $O(|Q|^{O(k)} \cdot n^{O(1)})$. This proves that the k-dominating set problem can be FPT reduced to **REGSPEC** with alphabet as parameter. □

6.1.4 Hardness results for polynomial treewidth networks

We say a family of graphs $\mathcal{G}$ has polynomial treewidth if the graphs of the family are of size at most polynomial in their treewidth, precisely: if there is a non-constant polynomial map $p_{\mathcal{G}}$

(with rational exponents in $(0,1)$) such that for any $G=(V,E)\in\mathcal{G}$ it holds $\mathrm{tw}(G)\geq p_{\mathcal{G}}(|V|)$. Moreover, we say the family is *constructible* if there is a polynomial time algorithm that given n produces a connex graph $G_n\in\mathcal{G}$ with n nodes. The following lemma is based on a polynomial time algorithm to find large perfect brambles in graphs [55]. This structure allows to embed any digraph in an input graph with sufficiently large treewidth via path routing while controlling the maximum number of intersections per node of the set of paths.

Lemma 6.18 (Subgraph routing lemma) *For any family $\mathcal{G}$ of graphs with polynomial treewidth, there is a polynomial map p and a deterministic polynomial time algorithm that, given any graph $G=(V,E)\in\mathcal{G}$ and any digraph $D=(V',E')$ of maximum (in/out) degree Δ and size at most $p(|V|)$, outputs:*

- *a mapping $\mu:V'\to V$ such that, for each $v\in V$, $\mu^{-1}(v)$ contains at most two elements,*
- *a collection $\mathcal{C}=(p_{e'})_{e'\in E'}$ of paths connecting $\mu(v'_1)$ to $\mu(v'_2)$ for each $(v'_1,v'_2)\in E'$, and such that any node in V belongs to at most 4Δ paths from $\mathcal{C}$.*

Proof. By [55, Theorem 5.3] there exists a polynomial map p_1 and a polynomial time algorithm that given a graph $G=(V,E)\in\mathcal{G}$ finds a *perfect bramble* $\mathcal{B}=(B_1,\dots,B_k)$ with $k\geq p_1(p_{\mathcal{G}}(|V|))$, *i.e.* a list of connected subgraphs $B_{\mathrm{i}}\subseteq V$ such that:

1. $B_{\mathrm{i}}\cap B_j\neq\emptyset$ for all i and j,
2. for all $v\in V$ there are at most two elements of $\mathcal{B}$ that contain v.

We set the polynomial map of the lemma to be $p=p_1\circ p_{\mathcal{G}}$ and consider any digraph $D=(V',E')$ of maximum (in/out) degree Δ and size at most $p(|V|)$. We suppose $k=|V'|$ (by forgetting some elements of $\mathcal{B}$) and reindex the element of $\mathcal{B}$ by V'. The map $\mu:V'\to V$ is constructed by picking some element $\mu(v')\in B_{v'}$ for all $v'\in V'$. The fact that any vertex $v\in V$ is contained in at most two elements of the bramble $\mathcal{B}$ ensures the first condition of the lemma on μ. Now, for each $(v'_1,v'_2)\in E'$ we define a path from $\mu(v'_1)$ to $\mu(v'_2)$ as follows: let $v\in B_{v'_1}\cap B_{v'_2}$ (first property of perfect brambles) then choose a path from $\mu(v'_1)$ to v inside $B_{v'_1}$ (which is connected) followed by a path from v to $\mu(v'_2)$ inside $B_{v'_2}$. The collection of paths $\mathcal{C}$ thus defined is such that there are at most 2Δ paths that start or end in $\mu(v')$ for any $v'\in E$. Moreover, for any $v\in V$, there are at most two elements of $\mathcal{B}$ that contain v, let's say $B_{v'_1}$ and $B_{v'_2}$. Then the only paths from $\mathcal{C}$ that can go through v are those starting or ending at either $\mu(v'_1)$ or $\mu(v'_2)$, so they are at most 4Δ in total. □

Theorem 6.19 *For any family $\mathcal{G}$ of constructible graphs of polynomial treewidth, the problem* nilpotency *is coNP-complete.*

Proof. First, by Lemma 6.2, a freezing automata networks with n nodes is nilpotent if and only if $F^{\lambda(n)}$ is constant where $\lambda(n)\in O(n)$ is the concrete computable bound from the lemma. The nilpotency problem is therefore clearly coNP.

We now describe a reduction from problem SAT. Given a formula with n variables seen as a Boolean circuit of maximum input/output degree 2 (of size polynomial in n), we first

construct $G = (V, E) \in \mathcal{G}$ such that the DAG $G' = (V', E')$ associated to the circuit is of size at most $p(|V|)$ where p is the polynomial map of Lemma 6.18. Then, using Lemma 6.18, we have a map $\mu : V' \to V$ and a collection $\mathcal{C}$ of paths in G that represent an embedding of G' inside G. The lemma gives a bound 8 on the number of paths visiting a given node $v \in V$. Then each node will hold 8 Boolean values, each one corresponding either a node $v' \in V'$ of the Boolean circuit or an intermediate node of a path from the collection $\mathcal{C}$. The alphabet is then $Q = \{0, 1\}^8 \cup \{\perp\}$ where $\perp$ is a special error state. In any configuration $c \in Q^V$, a node can be either in error state $\perp$, or it holds 8 Boolean components. We then construct the local rule at each node $v \in V$ that give a precise fixed role to each such component: it either represent a node $v' \in V'$ such that $\mu(v') = v$, or an intermediate node in one of the paths from $\mathcal{C}$, or is unused (because not all vertices of V have 8 paths from $\mathcal{C}$ visiting them). The local rule at $v \in V$ is as follows:

- if in state $\perp$ or if some neighbors is in state $\perp$, it stays in or changes to $\perp$;
- it then make the following checks and let the state unchanged if they all succeed or changes to $\perp$ if at least one test fails:
  1. check for any component corresponding to a node $v' \in V'$ that it holds the Boolean value $g(x, y)$ where g is the Boolean gate associated to v' in the the circuit and x and y are the Boolean values of the components corresponding to the vertex just before v in the two paths ρ_{e_1} and ρ_{e_2} in $\mathcal{C}$ that arrive at $\mu(v') = v$. In the case where g is a 'not' gate, there is only one input and in the case where v' is an input of the circuit, there is no input and nothing is checked;
  2. moreover, if the gate corresponding to v' is the output gate of the circuit, check that its Boolean value is 1;
  3. check for any component corresponding to an intermediate node in some path from $\mathcal{C}$ that the Boolean value it holds is the same as that of the component corresponding to the predecessor in the path.

We claim that F is not nilpotent if and only if the formula represented by the Boolean circuit is satisfiable. Indeed the configuration everywhere equal to $\perp$ is always a fixed point. It should be clear that if the formula is satisfiable then one can build a configuration corresponding to a valid computation of the circuit on a valid input which is a fixed point not containing state $\perp$. In this case we have two distinct fixed points and the automata network is not nilpotent. Conversely, suppose the the automata network is not nilpotent. Then it must possess a fixed point c distinct from the all $\perp$ one. Indeed, all configurations of $X = F^t(Q^V)$ are fixed points for t large enough (by the freezing condition) and if F^t is not a constant map then X must contain at least two elements. Moreover, the fixed point c do not contain state $\perp$, because otherwise it would contain a state from $Q \setminus \{\perp\}$ at some node which has a neighbor in state $\perp$, which would contradict the fact that it is a fixed point according to the local rule. Then c is a configuration where all checks made by the local rules are correct: said differently, c contains the simulation of a valid computation of the Boolean circuit that outputs 1. Therefore the Boolean formula is satisfiable and the reduction follows. □

When giving an automata network as input, the description of the local functions depends on the underlying graph (and in particular the degree of each node). However, some local functions are completely isotropic and blind to the number of neighbors and therefore can be

described once for all graphs. This is the case of local functions that only depends on the set of states present in the neighborhood. Indeed, given a map $\rho : Q \times 2^Q \to Q$ and any graph $G = (V, E)$, we define the automata network on G with local functions $F_v : Q^{N(v)} \to Q$ such that $F_v(c) = \rho\big(c(v), \{c(v_1), \ldots, c(v_k)\}\big)$ where $N(v) = \{v_1, \ldots, v_k\}$ is the neighborhood of v which includes v. We then say that the automata network is *set defined* by ρ. We will prove the next two hardness results with a fixed set defined rule, showing that there is a uniform and universally hard rule on graphs of polynomial treewidth for predecessor and asynchronous reachability problems. The proof below uses again Lemma 6.18 to embed arbitrary circuits like in theorems above, but the difference here is that the circuit embedding is written in the configuration and is not hardwired into the local rule. Moreover, the reduction also uses $L(1, 1)$ graph coloring [12] to deal with communication routing in a set defined rule in a similar way as in a radio network.

Theorem 6.20 *There exists a map $\rho : Q \times 2^Q \to Q$ such that for any family $\mathcal{G}$ of constructible graphs of polynomial treewidth and bounded degree, the problems* predecessor *and* asynchronous reachability *are both NP-complete when restricted to $\mathcal{G}$ and automata networks set-defined by ρ.*

Proof. These problems are clearly NP. For clarity of exposition we will construct a distinct map ρ for each of the two problems. Then, by taking the disjoint union of the alphabets and merging the two rules with the additional condition that any node that sees both alphabets is left unchanged, we obtain a single map ρ that is hard for both problems. Indeed, using the first alphabet only for predecessor problem inputs, we have the guarantee that the only possible pre-images must only use the first alphabet, hence the hardness follows for the combined rule. The same is true for asynchronous reachability using the second alphabet.

Let's now describe $\rho_1 : Q_1 \times 2^{Q_1} \to Q_1$ that set defines automata networks which have a NP-complete predecessor problem when restricted to $\mathcal{G}$. We describe it while showing the polynomial time reduction from SAT to the predecessor problem. Given a formula with n variables seen as a Boolean circuit of maximum input/output degree 2 (of size polynomial in n), we first construct $G = (V, E) \in \mathcal{G}$ such that the DAG $G' = (V', E')$ associated to the circuit is of size at most $p(|V|)$ where p is the polynomial map of Lemma 6.18. Then, using Lemma 6.18, we have a map $\mu : V' \to V$ and a collection $\mathcal{C}$ of paths in G that represent an embedding of G' inside G. The lemma gives a bound 8 on the number of paths visiting a given node $v \in V$. Let's compute a vertex coloring $\chi : V \to \{1, \ldots, k\}$ of the square of G with $k \leq \deg(G)^2 + 1$ colors, *i.e.* a vertex coloring of G such that no pair of neighbors of a given node has the same color (this can be done in polynomial time by a greedy algorithm). To implement the routing of information along paths of $\mathcal{C}$ and the circuit simulation by ρ_1, the alphabet Q_1 holds $8k$ state components, and we will use configurations where each node $v \in V$ uses only components $8\chi(v)$ to $8\chi(v) + 7$. These components can be seen as communication channels. Indeed, in such configurations, a node can distinguish the information going through up to 8 distinct paths coming from each neighbor individually just by looking at the set of states present in the neighborhood (because no pair of neighbors can use the same channel). Apart from the routing of information through paths, the rule ρ_1 implements each gate $v' \in V'$ of the Boolean circuits inside node $\mu(v')$ of G. We think of paths from $\mathcal{C}$ as being part of the circuit with nodes that implement the identity map. For that purpose

each state component in a node is associated to a descriptor that gives the type of gate to implement (input, identity, not, or, and, output) and the component numbers corresponding to input(s) of the gate (gates of type 'input' have no input). Formally, a state component is given by $S_1 = \{0, 1, \texttt{ok}, \texttt{off}\}$ where 0 and 1 are Boolean values, `off` means that the component is unused and `ok` is a special transitory state used to check correctness of computations (see below). A descriptor component is given by D_1, a finite set used to code any possible combination of gate type and input component numbers ($|D_1| = 6(8k)^2$ is enough). Then the state set of ρ_1 is $Q_1 = (S_1 \times D_1)^{8k}$. In a given configuration, we say that a given node $v \in V$ reads value $x \in \{0, 1\}$ on channel i if there is a unique state in the neighborhood with a state component i which is not `off`, and if this state component contains value x. In any other case, the value read on channel i is undefined. The rule ρ_1 does the following:

- the D_1 component are never changed;
- state components in `off` stay unchanged;
- any state component in `ok` becomes `off`;
- any state component in state $x \in \{0, 1\}$ checks that x is the correct output value of its gate type applied to the values read on the input channels given by its corresponding descriptor (in particular these input values must be defined). If it is the case, it becomes `ok`, otherwise `off`. The only exception to this rule is the case of the gate of type "output" where we only change state to `ok` if $x = 1$ and the computation check is correct, and change to `off` in any other case.

We then build configuration $c \in Q_1^V$ for the predecessor problem as follows:

- input component numbers and gate types in D components are set according to the Boolean circuit and the path collection $\mathcal{C}$;
- all unused state components are marked as `off`;
- all used state components are marked `ok`.

We claim that c has a predecessor in one step (*i.e.* $F_{\rho_1}(y) = c$ for some $y \in Q_1^V$) if and only if the SAT formula represented by the Boolean circuit is satisfiable. Indeed, the only possible predecessor configurations of c are such that all used state component hold a Boolean value equal to the output value of the gate they code applied to their corresponding input Boolean values, and that the output gate holds value 1.

We now describe $\rho_2 : Q_2 \times 2^{Q_2} \to Q_2$ that set defines automata networks which have a **NP**-complete asynchronous reachability problem when restricted to $\mathcal{G}$. The construction is almost identical to ρ_1 and the reduction is again from SAT problems, but with the following modifications:

- the state component is now $S_2 = \{\boxed{?}, 0, 1, \texttt{ok}, \texttt{off}\}$ where the new state $\boxed{?}$ represents a pre-update standby state; in each state component, the possible state sequences are subsequences of either $\boxed{?} \to \{0, 1\} \to \texttt{ok} \to \texttt{off}$ or $\boxed{?} \to \{0, 1\} \to \texttt{off}$;
- to each input gate of the Boolean circuit is attached a pre-input gate that serve as non-deterministic choice for input gates using asynchronous updates; the set D_2 is a modification of D_1 taking into account this new type of gates; the alphabet is then $Q_2 = (S_2 \times D_2)^{8k}$;

- the behavior of each state component depending on its type is as follows:
  - pre-input components become `ok` if previously in state $\boxed{?}$ and `off` in any other case;
  - input components in state $\boxed{?}$ become either 0 or 1 depending on whether there corresponding pre-input component is in state $\boxed{?}$ or not;
  - any other state component in state $\boxed{?}$ become $x \in \{0, 1\}$ the output value of its gate type applied to the values read on the input channels given by its corresponding descriptor (in particular these input values must be defined). If in a state from $\{0, 1\}$, it becomes `ok`, and in any other case it becomes `off`. The only exception to this rule is the case of gates of type "output" where we only change state to `ok` if the current value is 1, and change to `off` if the current value is 0.

When then define source configuration c_0 and destination configuration c_1 for the asynchronous reachability problem as follows. They both use the same circuit embedding like in c above but with a pre-input attached to each input. In c_0 all unused components are in state `off` and all used state components (including pre-inputs) are in state $\boxed{?}$. In c_1 all unused components are in state `off` and all used state components are in state `ok`. It should be clear that c_1 can be reached from c_0 if the formula associated to the Boolean circuit is satisfiable since either 0 or 1 can be produced at each input depending on whether the associated pre-input is update before the input update or not. Suppose now that c_1 can be reached from c_0 with some asynchronous update. First, all used state components except pre-inputs must follow either the sequence $\boxed{?} \to 0 \to$ `ok` or $\boxed{?} \to 1 \to$ `ok`. Therefore we can associate to each such component a unique Boolean value (0 or 1 respectively) and the rule ρ_2 ensures that the Boolean value of each such component is the output value of its corresponding circuit gate applied on the Boolean value of its corresponding inputs. Moreover the output gate must have Boolean value 1 so we deduce that the simulated circuit outputs 1 on the particular choice of Boolean values of inputs. The reduction from SAT follows. □

In the remaining of this section, we focus on the prediction problem for families of graphs with polynomial treewidth. In particular, we are interested in deriving an analogous of Theorem 6.20 for prediction problem. Nevertheless, as a log-space or a NC reduction of some **P**-complete problem is required, most of the latter results that worked for Theorem 6.20 are not necessarily valid in this context as we only know that there exist polynomial time algorithms that compute certain needed structures. In order to face this task, our approach is based in slightly modifying the input of our prediction problem and then show that we can efficiently compute paths in a polynomial treewidth graph G. The latter will allow us to show that we have an analogous of subgraph routing lemma (Lemma 6.18). In particular, as it is not clear if the perfect bramble structures used in order to obtain Lemma 6.18 are calculable in **NC** or in log-space, we need to modify the problem in order to show that there exists a log-space reduction or an **NC** reduction of circuit value problem (CVP) in this particular variation of prediction, and thus that it is **P**-complete. More precisely, we add a perfect bramble of polynomial size to the input of Prediction problem. We call this modified version of prediction Routed Prediction problem.

Problem (Routed prediction problem)

Parameters: alphabet Q, family of graphs $\mathcal{G}$ of max degree Δ

Input:

1. a deterministic freezing automata network $\mathcal{A} = (G, F)$ on alphabet Q, with set of nodes V with $n = |V|$ and $G \in \mathcal{G}$;
2. an initial configuration $c \in Q^V$
3. a node $v \in V$ and a $(\{v\}, Q, t)$-specification $\mathcal{S}_v$ of length $t \in \mathbb{N}$
4. A perfect bramble $\mathcal{B} = (B_1, \ldots, B_p)$ in with $p = n^{\mathcal{O}(1)}$ in G

Question: does the orbit of c restricted to v satisfies specification $\mathcal{S}_v$?

Now, having this latter problem in mind, we slightly modify the definition of a constructible familly of graphs $\mathcal{G}$ of polynomial treewidth introduced at the begining of this section: we define a routed collection of graphs of polynomial treewidth to the set $\mathcal{G} = \{(G_n, \mathcal{B}_n)\}_{n \in N}$ such that G_n is an undirected connected graph of order n and treewidth $\mathrm{tw}(G_n) \geq p(n)$ and $\mathcal{B}_n$ is a perfect bramble such that $|\mathcal{B}_n| \geq p'(n)$ where p and p' are polynomials. We say a that a routed collection of graphs of polynomial treewidth $\mathcal{G}$ is log-constructible if there is a log-space algorithm that given n produce the tuple $(G_n, \mathcal{B}_n) \in \mathcal{G}$. As we will be working with a log-constructible collection of routed graphs, we would like to say that we could have the result of Lemma 6.18 in order to show the main result of this section. Nevertheless, in order to do that, we need to have a log-space or a NC algorithm computing the paths that we will be using for the proof of the main result. More precisely, we need to compute the function μ and the collection of paths $\mathcal{C}$. Fortunately, in [68, Theorem 5.3] it is shown that there exist a log-space algorithm that accomplish this task. Finally, as in the proof of Theorem 6.20, we need a proper coloring of the square graph G^2 in order to broadcast information through the paths in the collection $\mathcal{C}$ without encountering problems in the nodes that are in different paths at the same time. Fortunately, we can do this in **NC** as it is stated in [28, Theorem 3].

We are now in condition of showing our main result concerning routed prediction problem:

Theorem 6.21 *There exists a map $\rho : Q \times 2^Q \to Q$ such that Routed Prediction problem is **P**-complete restricted to any family $\mathcal{G}$ of log-constructible routed collection of graphs of polynomial treewidth.*

Proof. We start by observing that prediction problem is in **P**. We also recall that in order to show the **P**-hardness of prediction problem, it suffices to show that there exist a **NC** reduction for the alternating monotone 2 fan-in 2 fan-out circuit value problem (AM2CVP), more precisely AM2CVP $\leq_m^{\mathbf{NC}^2}$ PRED$_{\mathcal{G}}$ (see [45] Theorem 4.2.2 and Lemma 6.1.2). Let $n, l \in \mathbb{N}$, $C : \{0,1\}^n \to \{0,1\}^l$ a monotone alternating 2 fan in 2 fan out circuit, $x \in \{0,1\}^n$ and $o \in \{0, \ldots, l-1\}$ a fixed output of C. We call $C' = (V', E')$ to the underlying DAG defining C and we fix $G \in \mathcal{G}$ where $\mathcal{G}$ is a log-constructible family of graphs with polynomial treewidth. We note that, by definition we can compute G and a perfect bramble of size $p = n^{\mathcal{O}(1)}$ in log-space and thus we can do the latter computations in $\mathbf{NC}^2$. Now, we use $\mathcal{B}$ and Proposition 1.20 in order to compute a mapping $\mu : V' \to V$ and a collection of

paths $\mathcal{C}$ as in Lemma 6.18. As we did in Theorem 6.20, we use Proposition 1.21 in order to compute a k-proper coloring $\chi : V \to \{1, \ldots, k\}$ of G^2 in $\mathbf{NC}^2$ with $k = \Delta^2 + 1$ for $\Delta \in \mathbb{N}$ such that $\Delta(G) = \Delta$. From here we construct ρ analogously as we did for ρ_1 and ρ_2 in the proof of Theorem 6.20 but observing that now we have only 5 type of gates as the circuit is monotone. We also consider state component $S = \{0, 1, \textbf{wait}, \textbf{off}\}$. Remember that the descriptor component assures that there won't be overlappings of the channels during broadcasting. We map x into a configuration $y \in Q = \{S \times D\}^{8k}$ in the following way:

- The D component is assigned according to the structure of C'.
- For every input we assign a boolean value given by x.
- For every unused node we assign the state **off**.
- For every other node we assign the state **wait**.

The rule ρ is defined in the following way:

- Every node in state **off**, 0 or 1 is fixed and does not change its state.
- Every node in state **wait** reads the information of its neighbors and do the following depending on its type of gate:
  - identity will take the value of its input
  - AND will read its inputs: if both inputs are in 1 it will change to 1 and it will change to 0 if it reads one neighbor in 0. In any other case it will remain in **wait**
  - OR will read its inputs: if both inputs are in 0 it will change to 0 and it will change to 1 if it reads one neighbor in 1. In any other case it will remain in **wait**

In order to show the desired result, it will suffices to show the following simulation property: there exists $t \in \mathbb{N}$, $t = n^{\mathcal{O}(1)}$ such that for every output $o \in V'$ we have $C(x)_o = (F^t(y)_{\mu(o)})|_S$ where $y \in Q^V$ is the configuration computed from x as explained above. In fact, if we have the latter property, for some fixed output o, we define $v = \mu(o)$ and $\mathcal{S}_v$ be a $(\{v\}, Q, t)$-specification such that $\mathcal{S}_v = \{z \in \{0, 1\} : z \neq y_v \text{ and } (F_\rho(y)^t|_v)|_S = z\}$ and then we can answer if the orbit of y in time t given by $F^t_\rho(y)$ satisfies $\mathcal{S}_v$ if and only if we can answer if $C(x)_o = 1$ (and thus AM2CVP $\leq_m^{\mathbf{NC}^2}$ PRED). We now show that the latter simulation property holds. In order to do that, we inductively check, that eventually, the orbit of y will evaluate every layer of the circuit. We start by the input. Note that in one time step all the information is broadcasted through the different channels and through the paths given by $\mathcal{C}$. In a maximum of $L = n^{\mathcal{O}(1)}$ time steps (given by the longest path of C') the last signal will arrive to a gate in the first layer. Note that, with the gates described above, signals arriving at different times do not change its output value as gates have a monotone behaviors on states with order **wait** $\leq 0 \leq 1$. Iteratively, we have maximum arriving times for signals of L time steps for each layer and then, defining $t = L \times \text{deph}(C) = n^{\mathcal{O}(1)}$ and observing that output nodes will will remain constant once they have done a computation (when they change to a boolean value), we get the desire result. Therefore, AM2CVP $\leq_m^{\mathbf{NC}^2}$ PRED holds and then, $\text{PRED}_\mathcal{G}$ is **P**-complete. □

6.2 Counting complexity on freezing automata networks: a case study.

In this subsection, we address the problem of computing the exact probability that a node reaches a given state, in the Bootstrap Percolation model under a random sweep updating scheme. In the Bootstrap Percolation model, nodes in the network have two possible states, namely 0 or 1, that evolve according to the following rule: (1) when a node in state 0 is updated, it evolves taking the state of the strict majority of its neighbors; (2) when a node in state 1 is updated, it remain in state 1. Bootstrap Percolation models are well-studied within the framework of modelling many physical, social and biological phenomena such as magnetic properties of some materials [14] crystal growth [43, 75], alert spreading in distributed networks [74], sand pile formation and disease spreading [7].

Asynchronous bootstrap percolation Given a graph $G = ([n], E)$, we consider that each node has one of two possible *states*, that we denote 0 and 1 and call *healthy* and *infected*, respectively. A *configuration* of G is a vector $x \in \{0,1\}^n$ assigning a state to each node. We define the following dynamic over a configuration x (that in the following is called *initial condition*), that we call *asynchronous bootstrap percolation*. First, we fix a positive integer $t \leq n$, that we call *time-span*. Second, we fix an injective function $\sigma : [t] \to [n]$, that we call *updating sequence*. Then, the *trajectory* of (x, σ, t) is the sequence of configurations $x^\sigma(0), \ldots, x^\sigma(t)$ such that $x^\sigma(0) = x$ and, for each $0 \leq k < t$ and $\mathrm{i} \in [n]$ we define:

$$x_{\mathrm{i}}^\sigma(k+1) = \begin{cases} x_{\mathrm{i}}^\sigma(k) & \text{if } \mathrm{i} \neq \sigma(k+1), \\ 1 & \text{if } \mathrm{i} = \sigma(k+1) \text{ and } x_{\mathrm{i}}^\sigma(k) = 1, \\ 1 & \text{if } \mathrm{i} = \sigma(k+1) \text{ and } x_{\mathrm{i}}^\sigma(k) = 0 \text{ and } \sum_{j \in N(\mathrm{i})} x_j^\sigma(k) > \lfloor \mathrm{d}(\mathrm{i})/2 \rfloor, \\ 0 & \text{if } \mathrm{i} = \sigma(k+1) \text{ and } x_{\mathrm{i}}^\sigma(k) = 0 \text{ and } \sum_{j \in N(\mathrm{i})} x_j^\sigma(k) \leq \lfloor \mathrm{d}(\mathrm{i})/2 \rfloor, \end{cases}$$

In words, in the k-th configuration, the state of all nodes except $\sigma(k)$ remain in the same state than in the previous configuration. Node $\sigma(k)$ remains in state 1 if it was in state 1 in the previous configuration, and otherwise it takes the state of the strict majority of its neighbors.

Let us fix a graph $G = ([n], E)$, a vertex $v \in [n]$, a configuration x, and a time-span t. We say that an updating sequence σ *activates* v if $x_v^\sigma(t) = 1$. Moreover, in that case we say that σ is a *good-sequence* for G, x, v, and t. We denote by $\mathsf{Good}(G, x, v, t)$ the set of good-sequences for G, x, v and t. When the context is clear, we omit the specifications of the graph, configuration, node and time-span. We say that v is *stable* if $\mathsf{Good} = \emptyset$, and otherwise we say that the node is *unstable*.

We define a computational task which, inspired by the relations with the dynamics of Bootstrap Percolation and disease spreading, we call Contagion-Probability. The input of problem Contagion-Probability is a simple undirected n-node graph G with vertex set $[n]$, an initial condition $x \in \{0,1\}^n$ which assigns a state to each node of the graph, a node $v \in [n]$ that we call *objective node* and a *time-span* $t \leq n$. The task consists in computing the probability that node v reaches state 1 in at most t time-steps under a random sweep updating scheme.

Observe that in the definition of Contagion-Probability we restricted the time-span to

be bounded by the number of nodes n. There are two reasons for considering this restriction. The first reason is that in many percolation processes (such as rumor or disease spreading) is unnatural to consider a time-spans larger than the number of nodes, as the later number could be extremely large. Second, given an initial condition, these processes reach the same fixed point on every updating scheme, as it was shown on [38, Theorem 1]. Moreover, it is possible to reach the fixed point updating at most n nodes.

Contagion-Probability We formally define the problem Contagion-Probability. As we explained at the start of this chapter, this problem asks for the probability of choosing a good-sequence, when an update sequence is picked uniformly at random. Formally, this problem receives as input a graph $G = ([n], E)$, a vertex $v \in [n]$ of G, an initial condition $x \in \{0,1\}^n$ and a time-span $t \leq n$. The task consists in computing $|\mathsf{Good}(G, x, v, t)|$ divided by $n!/(n-t)!$, which is the fraction of good-sequences from the total number of possible updating sequences with time-span t.

As we have stated before, the number $n!/(n-t)!$ can be computed in time polynomial in n, and also the division of two integers can be computed in a running time that is linear in the number of bits required for their binary representation (see Proposition 1.22). Therefore, the difficulty of Contagion-Probability rests in the computation of $|\mathsf{Good}(G, x, v, t)|$. For that reason, we abuse notation and refer as Contagion-Probability also the problem of computing the latter number.

Problem (Contagion-Probability)

Input:

1. an n-node graph $G = ([n], E)$.
2. a node $v \in [n]$
3. an initial condition $x \in \{0,1\}^n$.
4. a natural number $t \leq n$.

Output: $|\mathsf{Good}(G, x, v, t)|$

In order to study how difficult is to solve one of latter problem, we use computational complexity theory. Roughly, we are interested in measuring how difficult is to solve the problem, measuring the amount of resources required to solve in a Turing machine it in the worst case. These resources are measured with respect to to the size of the input, in this case this corresponds to the number of bits required to encode G, vertex v, the configuration x and the time-span t. This quantity is $\Theta(n + m + \log n)$, as the graph can be encoded in $\Theta(n + m)$ bits, the name of v and the value of t can be encoded in $\lceil \log n \rceil$ bits, and the configuration x is encoded in n bits.

The complexity of Contagion-Probability

Observe that Contagion-Probability is not a decision problem, as in consists in computing a number in $[0, 1]$. The natural generalization of classes of decision problems are the

functional problems. These problems are defined by a function $f : \{0,1\}^* \to \{0,1\}^*$ (or equivalently $f : \{0,1\}^* \to \mathbb{N}^*$), and the task is to compute $f(x)$ for a given input $x \in \{0,1\}^*$.

Then, our goal is to answer in which context Contagion-Probability is in **FP**. Unfortunately, the brute-force algorithm that test all possible permutations of the set of vertices in unfeasible, as this number is exponential in the size of the input. A natural question is whether the brute-force algorithm is optimal, or there exist properties of the dynamic (perhaps restricting the input graph to some class), that can be algorithmically exploited in order to obtain a polynomial-time algorithm.

Observe that the counting version of an **NP**-Complete problem can be considered as a good candidate for a #**P**-Complete problem. In fact, up to our knowledge, there are no examples of a **NP**-Complete problems whose counting version is not #**P**-Complete [58]. Interestingly, Valiant shows in [76] that there are examples of problems that their decision version is solvable in polynomial-time, while their counting version is #**P**-Complete. For instance, the problem consisting in computing a maximum matching of a given graph is solvable in polynomial time, while the problem of counting all matchings is #**P**-Complete [76].

Let (G, x, v, t) be an instance of Contagion-Probability. Remember that an injective function $\sigma : [t] \to V$ is a *good-sequence* if the objective node v reaches state 1 in the dynamics of Bootstrap Percolation under the updating sequence σ. Now consider the following decision problem, that we call Good-Sequence. This problem receives the same instances that Contagion-Probability, but output *yes* when there exist at least one-good sequence of the given instance. Clearly this problem is in **NP**, as a good sequence is a witness of polynomial size that can be verified in polynomial time by simply simulation of the dynamics of Bootrstap Percolation.

Observe that the output of Contagion-Probability corresponds to the number of good sequences of the input instance, divided by $n!/(n-t)!$, which corresponds to the number of possible injective functions $\sigma : [t] \to V$. As the latter number can be computed in a running time that is polynomial in n (see the preliminaries section for more details), the difficulty of Contagion-Probability is reduced to the computation of the number of good-sequences. In the following when we refer to Contagion-Probability, we do not distinguish between the computation of the number of good-sequences and the computation of the actual probability. Note that previous observations imply that Contagion-Probability is in #**P**.

We show that Contagion-Probability is #**P**-Complete. Roughly this result implies that in order to compute the output, there is no better algorithm that simply simulate the dynamics on every possible injective function $\sigma : [t] \to [n]$, and keep the count of the ones that are good-sequences.

Our result is obtained constructing a polynomial-time Turing reduction from a version of #Sat to problem Contagion-Probability. The functional problem #Sat is the counting version of Boolean Satisfability, i.e., given a Boolean formula, the task is to count the number of truth-assignments satisfying it. In fact, our reduction is not from #Sat directly, but instead from a restriction of the problem called #Mon-2-Sat, where the input Boolean formula is in a 2-CNF form and it is monotonic (it does not have any negations of variables).

This result turns to be quite natural, as in [38] it is shown that GOOD-SEQUENCE (i.e. simply decide if the set of good sequences is non empty) is **NP**-Complete. As we said above, up to our knowledge, there are no examples of a **NP**-Complete problems whose counting version is not #**P**-Complete. In the same reference, it is shown that GOOD-SEQUENCE remain **NP**-Complete even when the problem is restricted to graphs of maximum degree 5.

Interestingly, when the input graph has maximum degree 4, GOOD-SEQUENCE is solvable in polynomial time [38]. This leads us to ask for the complexity of CONTAGION-PROBABILITY restricted to that family of graphs. As we mentioned before, there are problems solvable in polynomial time with #**P**-Complete counting versions. We remark that the restriction to graphs of maximum degree 4 contains important instances, as the ones where the dynamics occur on a two-dimensional grid with periodic boundary conditions.

Our second result is that CONTAGION-PROBABILITY restricted to input graphs of maximum degree 4 is solvable in polynomial time, hence belongs to **FP**. Our algorithm is based on a characterization of the initial conditions with at least one good-sequence, given in [38]. This characterization states that when a good-sequence exists, the objective node belong to some tree T of the input graph, such that in every good sequence, a *pruning sequence* of its nodes is induced. Generally speaking, a *prunning* sequence for T is a sequence in which every node of the tree must change its state to 1 before the objective node. In this regard, CONTAGION-PROBABILITY solution uses an algorithm that counts the number of pruning sequences of a given tree as a subroutine.

This subsection is organized as follows: in Section 6.2.1 we show that CONTAGION-PROBABILITY problem is #**P**-complete by showing #MON-2-SAT is Turing reducible to CONTAGION-PROBABILITY, i.e. #MON-2-SAT $\in \mathbf{FP}^{\text{CONTAGION-PROBABILITY}}$ however, in Section 6.2.2 we show, based on previous results in the characterization of good sequences [38], that problem CONTAGION-PROBABILITY is in **FP** when restricted to graphs of maximum degree 4.

6.2.1 CONTAGION-PROBABILITY is #P-Complete

In this section, we show that CONTAGION-PROBABILITY is #**P**-Complete. In order to do that, we reduce #MON-2-SAT to CONTAGION-PROBABILITY by a polynomial-time Turing reduction. Roughly, we show that given an instance of #MON-2-SAT, we can produce a series of instances of problem CONTAGION-PROBABILITY that represent $\mathcal{F}$, in the sense that the number of truth-assignment satisfying $\mathcal{F}$ can be computed from the number of good sequences of these instances.

Let $\mathcal{F}$ be a monotone 2-CNF formula with n variables and m clauses. We call $Z(\mathcal{F}) = z_1, \ldots, z_n$ and $C(\mathcal{F}) = \{C_1, \ldots, C_m\}$, respectively the sets of variables and clauses of $\mathcal{F}$. Also, for each $j \in [m]$, we call c_1^j and c_2^j the two variables that participate in clause C_j. In other words

$$\mathcal{F}(z_1, \ldots, z_n) = \bigwedge_{j \in [m]} (c_1^j \vee c_2^j), \quad \text{with } c_j^1, c_j^2 \in Z(\mathcal{F}).$$

We call $\varphi(\mathcal{F})$ the set of truth-assignments satisfying $\mathcal{F}$. The *weight* of a truth-assignment corresponds to the number of variables that are assigned *true*. For $k \in [n]$ we call $\varphi(\mathcal{F}, k)$

the set of truth-assignments of *weight* k satisfying $\mathcal{F}$. Clearly $|\varphi(\mathcal{F})| = \sum_{k=1}^{n} |\varphi(\mathcal{F}, k)|$.

For $k \in [n]$, we define a graph $G[\mathcal{F}, k]$ on $N = (m+3)n + 8m + 2k + 2$ nodes, and an initial configuration $x[\mathcal{F}, k] \in \{0,1\}^N$ as follows.

- First, for each variable $z_{\mathrm{i}} \in Z(\mathcal{F})$ graph $G[\mathcal{F}, k]$ contains a *variable gadget*. This gadget consists in a node z_{i}, called *variable node*, with $m+1$ pending nodes denoted $z_{\mathrm{i},1}, \ldots z_{\mathrm{i},m+1}$ and called *auxiliary variable nodes*. In the initial configuration we assign the variable node to be inactive and the auxiliary variable nodes to be active.

  Intuitively, the variable nodes are going to simulate the variables of $\mathcal{F}$. They are initially inactive, and are adjacent to a large enough number of auxiliary variable nodes to become active if they are updated, with the aim of associating an updating sequence to a choice of a truth-assignments.

- Second, for each clause $C_j \in C(\mathcal{F})$, graph $G[\mathcal{F}, k]$ contains a *clause gadget*. This gadget consists in a node v^j, called *clause node*, and four more nodes that are adjacent to the clause node. Two of these nodes are denoted u_1^j, u_2^j and called *clause variable nodes*; the two remaining nodes are called *clause auxiliary nodes*. Node u_1^j is adjacent to the variable node associated to c_1^j, and node u_2^j is adjacent to the variable node associated to c_2^j. Finally, the initial configuration fixes v^j, u_1^j, u_2^j as inactive, and the auxiliary clause nodes as active.

- Third, graph $G[\mathcal{F}, k]$ contains one *threshold gadget*. This gadget consists in a node θ, called *threshold node*, and $n + 2m + 2k + 1$ other nodes called *auxiliary threshold nodes*, that are adjacent to the threshold node. The threshold node is also adjacent to all variable nodes, and all clause variable nodes. In the initial configuration, the threshold node and $2k$ of auxiliary threshold nodes are inactive, and the remaining $n + 2m + 1$ auxiliary threshold nodes are initially active. Observe that the threshold node has degree $2n + 4m + 2k + 1$.

- Finally, graph $G[\mathcal{F}, k]$ contains one *output gadget*. This gadget consists in a node *out* called *output node* and $m - 1$ other nodes called *auxiliary output nodes*. The output node is also adjacent to every clause node. All nodes in this gadget are initially inactive.

See Figure 6.2 for a graphical representation of each gadget. Observe that there are n variable gadgets, each containing $(m+2)$ nodes; there are m clause gadgets, each containing 5 nodes; one threshold gadget containing $n + 2m + 2k + 2$ nodes; and one output gadget containing m nodes. This sums up a total of $(n+3)m + 8m + 2k + 2$ nodes in graph $G[\mathcal{F}, k]$. See Figure 6.23 for more details on the structure of $G[\mathcal{F}, k]$.

Let us define the timespan $t(\mathcal{F}, k) = k + 2m + 2$ and call $v[\mathcal{F}, k]$ the output node of $G[\mathcal{F}, k]$. Given a sequence σ of $v[\mathcal{F}, k]$ with timespan $t[\mathcal{F}, k]$, we say that σ induces a truth-assignment z of $\mathcal{F}$ if z is such that $z_{\mathrm{i}} = true$ if and only if the corresponding variable node is activated in σ.

Let us call $\rho(\mathcal{F}, k)$ the set of good-sequences for $v[\mathcal{F}, k]$ with timespan $t[\mathcal{F}, k]$, i.e. $|\rho(\mathcal{F}, k)|$ is the output of problem Contagion-Probability on input $(G[\mathcal{F}, k], x[\mathcal{F}, k], v[\mathcal{F}, k], t[\mathcal{F}, k])$.

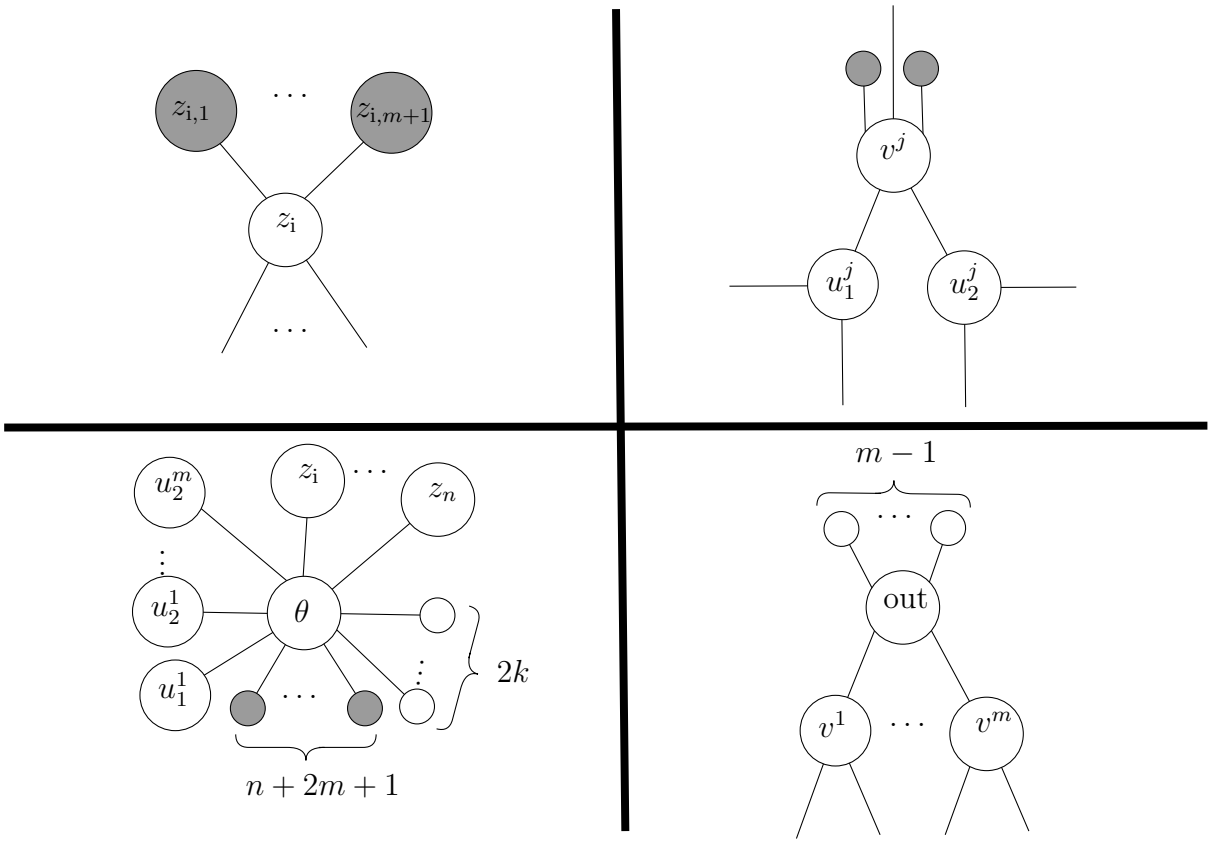


Figure 6.2: Gadgets used to construct graph $G[\mathcal{F}, k]$. (Left upper panel) variable gadget. (Right upper panel) clause gadget. (Left bottom panel) threshold gadget.(Right bottom panel) output gadget. Gray nodes are active and white nodes are inactive.

Lemma 6.22 *Suppose that $\sigma \in \rho(\mathcal{F}, k)$. Then, σ satisfies that:*

- *For every* $\mathrm{i} \in \{1, \ldots, k\}$, $\sigma(\mathrm{i})$ *is a variable node,*
- $\sigma(k+1)$ *is the threshold node,*
- *For every* $\mathrm{i} \in \{k+2, \ldots, 2m+k+1\}$, $\sigma(\mathrm{i})$ *is a node in a clause gadget,*
- $\sigma(2m+k+2)$ *is the output node.*

Proof. First, observe that the output node of $G[\mathcal{F}, k]$ has degree $2m-1$, with every neighbor initially inactive. Therefore, by the strict majority rule it is necessary to activate m of its neighbors before activating it. As $m-1$ of these neighbors are auxiliary output nodes, it is necessary to activate the m clause nodes before activating the output node. Each clause node has degree 5, where one of the neighbors is the output node, and two neighbors are auxiliary clause nodes, which are initially active. Therefore, the clause node requires that at least one of the corresponding clause variable nodes is activated before it. Remember that the clause variable nodes have degree three, where one neighbor is a clause node, another neighbor is the threshold node, and the remaining one is a variable node. Consider now the time-step on which for the first time a clause-variable node is updated. Then, this node requires that both the threshold node and the adjacent variable node are activated before it. Therefore, threshold must be changed to active during sequence σ, and must become active before all the clause variable nodes. The threshold node has degree $2n+4m+2k+1$, with $n+2m+1$ adjacent auxiliary threshold nodes initially active and the remaining $n+2m+2k$ neighbors initially inactive. As we explained, $2m$ of its inactive neighbors are still inactive when the threshold node is activated. Moreover, the $2k$ inactive auxiliary threshold nodes are also inactive when the threshold node is activated. Observe that the threshold node requires at least k more active neighbors to become active. More precisely let p the amount of active neighbors that are required to active threshold node. Then, we have that p must be chosen as the minimum number such the amount of active is more than the number of inactive neighbors. Summarizing previous calculations we have $(n+2m+1+p)$ active neighbors and $(n+2m+2k-p)$ inactive neighbors. Then, by asking p to satisfy $(n+2m+1+p)-(n+2m+2k-p)>0$ we deduce $p > k-\frac{1}{2}$ and thus, $p \geq k$. We conclude that the only option is to choose at least k variable nodes.

Wrapping up, we deduce the following order in every sequence activating the output node in $k+2m+2$ steps: first, k variable nodes are activated. Then, the threshold node is activated. Then, all the m clause nodes must be activated, which requires $2m$ steps, one for the clause variable node, and one for the clause node. Finally the output node is activated. We deduce that σ satisfies the statements of the lemma. □

Lemma 6.23 *Every $\sigma \in \rho(\mathcal{F}, k)$ induces a truth-assignment in $y \in \varphi(k, \mathcal{F})$. For every $y \in \varphi(k, \mathcal{F})$ there is a $\sigma \in \rho(\mathcal{F}, k)$ that induces y.*

Proof. Let σ be a good sequence in $\rho(\mathcal{F}, k)$. Lemma 6.22 implies that in a good sequence every clause node is activated, implying that at least one variable of each clause is chosen in the k first-steps. We deduce that σ induces a truth-assignment of weight k satisfying $\mathcal{F}$. Conversely, let y be a truth assignment in $\varphi(k, \mathcal{F})$. As this truth-assignment has weight k, we

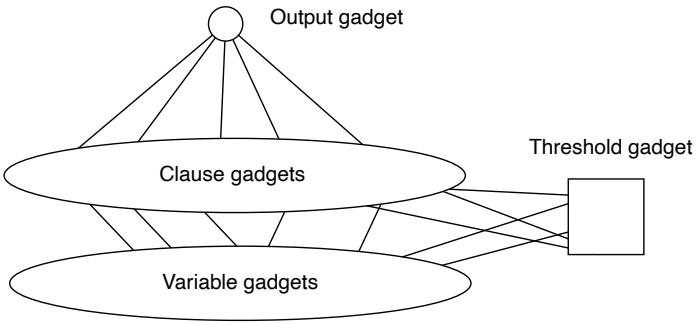


Figure 6.3: Scheme representing graph $G[\mathcal{F}, k]$. Each type of gadget is detailed in Figure 6.2.

can update the corresponding k variable nodes, and update the clause gadgets accordingly. This produces a sequence in $\rho(\mathcal{F}, k)$. □

Observe that Lemma 6.22 implies that for each truth-assignment $y \in \varphi(\mathcal{F}, k)$, there are several good sequences for $v[\mathcal{F}, k]$ with timespan $t[\mathcal{F}, k]$. Indeed, we know that in the first k time-steps the variable nodes that are *true* in y are updated. They can be updated in any order, so there are $k!$ ways of updating them. Then, the threshold node is updated. Then, $2m$ nodes are updated to activate all clause nodes. Let us call γ the number of ways that the m clause nodes can be activated, once the k variable nodes and the threshold node were activated according to y. Unfortunately, the value of γ is not easy to compute, since it depends each the truth-assignment y. We say that a truth-assignment y *fully satisfies* clause C if $y_{c_1} = y_{c_2} = true$, in other words, when both variables in the clause are true in y. Indeed, the exact value γ depends on the number of clauses that are fully satisfied by a given truth assignment. A clause that is fully satisfied has two ways to be updated (one for each clause variable node), while a clause that is not fully satisfied has only one.

Therefore, to be able to compute $|\varphi(\mathcal{F}, k)|$ we will require to count with more detail. Let $C = (c_1 \vee c_2)$ be a clause of $\mathcal{F}$. For $\mathrm{d} \in [m]$ we call $\varphi(\mathcal{F}, k, \mathrm{d})$ the set of truth-assignments of weight k satisfying $\mathcal{F}$, and such that exactly d clauses of $\mathcal{F}$ are fully satisfied. Clearly $|\varphi(\mathcal{F}, k)| = \sum_{\mathrm{d}=0}^{m} |\varphi(\mathcal{F}, k, \mathrm{d})|$. Similarly, we define $\rho(\mathcal{F}, k, \mathrm{d})$ as the number of good sequences of $v[\mathcal{F}, k]$ with timespan $t[\mathcal{F}, k]$ that induce truth-assignments in $\varphi(\mathcal{F}, k, \mathrm{d})$. Obviously $|\rho(\mathcal{F}, k)| = \sum_{\mathrm{d}=0}^{m} |\rho(\mathcal{F}, k, \mathrm{d})|$.

Lemma 6.24 $|\rho(\mathcal{F}, k, \mathrm{d})| = |\varphi(\mathcal{F}, k, \mathrm{d})| \cdot k! \cdot \dfrac{(2m)!}{2^m} \cdot 2^{\mathrm{d}}$

Proof. Let us fix $y \in \varphi(\mathcal{F}, k, \mathrm{d})$. From Lemma 6.23 we know that y induces a good sequence in $\rho(\mathcal{F}, k)$, and then by definition this sequence also belongs to $\rho(\mathcal{F}, k, \mathrm{d})$. From Lemma 6.22 we know that in the first k steps the variable nodes are updated. As they are independent, there are $k!$ ways of updating the variables that are *true* in y. Once the k variable nodes are activated, we have to continue by activating the threshold node. After that, we know from Lemma 6.22 that in the next $2m$ steps all the clause nodes must be activated.

For each clause, exactly two nodes are updated in the corresponding clause gadget. Indeed we need 2 time steps to activate each clause node, and if we spend more than two steps in any clause gadget we are not going to be able to activate the output node in the timespan. Let C be the first clause. There are $\binom{2m}{2}$ ways of choosing steps to update one of its clause variable nodes, and the clause node. To update the second clause, we have $\binom{2m-2}{2}$ ways of choosing steps to update the corresponding pair. Repeating this argument we deduce that there are

$$\prod_{\mathrm{i}=0}^{m} \binom{2m-2\mathrm{i}}{2} = \frac{2m!}{2^m}$$

ways of choosing steps for updating a pair of nodes of each clause. Finally, for each fully-satisfied clause there are two possible choices of a clause-variable node to update, giving a total of 2^{d} choices.

Previous calculations imply that there are $k! \cdot \frac{(2m)!}{2^m} \cdot 2^{\mathrm{d}}$ possible good sequences that induce y. We deduce that there are $|\varphi(\mathcal{F}, k, \mathrm{d})| \cdot k! \cdot \frac{(2m)!}{2^m} \cdot 2^{\mathrm{d}}$ good-sequences inducing truth-assignments in $\varphi(\mathcal{F}, k, \mathrm{d})$. □

Now let us call $P(s)$ the degree m polynomial defined by

$$P(s) = \sum_{\mathrm{d}=0}^{m} |\varphi(\mathcal{F}, k, \mathrm{d})| s^{\mathrm{d}},$$

and observe that $|\rho(\mathcal{F}, k)| = \sum_{\mathrm{d}=0}^{m} |\rho(\mathcal{F}, k, \mathrm{d})| = \frac{(2m)!}{2^m} \cdot k! \cdot P(2)$. We would like to compute the coefficients of $P(x)$. This is possible when we know an evaluation of the polynomial in a large enough point, as notices by Valiant in [76]. For sake of completeness we give the result and the full proof.

Lemma 6.25 *Let $P(x) = \sum_{\mathrm{i}=0}^{n} a_{\mathrm{i}} x^{\mathrm{i}}$ be a polynomial with integer coefficients upper bounded by a $A > 2$. Suppose that we know a pair (x_0, y_0) such that $y_0 = P(x_0)$ and $x_0 > A^2$. Then, there exists an algorithm that outputs the coefficients $a_0, \ldots, a_n$ of P in a time that is polynomial in $n(\log(x_0) + \log(y_0) + \log n)$.*

Proof. First, observe that $\sum_{\mathrm{i}=0}^{j-1} a_{\mathrm{i}} x_0^{\mathrm{i}} < x_0^j$. for every $j \in [n]$. Indeed,

$$\begin{aligned}\sum_{\mathrm{i}=0}^{j-1} a_{\mathrm{i}} x^{\mathrm{i}} < A \sum_{\mathrm{i}=0}^{j-1} x^{\mathrm{i}} = A x^{j-1} \sum_{\mathrm{i}=0}^{j-1} x^{\mathrm{i}-j+1} = A x^{j-1} \sum_{\mathrm{i}=0}^{j-1} \frac{1}{x^{j-1-\mathrm{i}}} \\ < A x^{j-1} \sum_{\mathrm{i}=0}^{j-1} \frac{1}{A^{2\mathrm{i}}} < \frac{3}{2} A x^{j-1} < x_0^j\end{aligned}$$

Then, since $a_n = \frac{y_0 - \sum_{\mathrm{i}=0}^{n-1} a_{\mathrm{i}} x_0^{\mathrm{i}}}{x_0^n}$ we deduce that, $\frac{y_0}{x_0^n} - 1 < a_n \leq \frac{y_0}{x_0^n}$ implying that $a_n = \left\lfloor \frac{y_0}{x_0^n} \right\rfloor$.

We sequentailly obtain the other coefficients taking $(x_0, y_0 - a_n x_0^n)$ as a pair for $P'(x) = \sum_{\mathrm{i}=0}^{n-1} a_{\mathrm{i}} x^{\mathrm{i}}$. From from Proposition 1.22 we deduce that each iteration can be done in time $\log(x_0) + \log(y_0) + \log n$. □

Note that, unfortunately, the expression $|\rho(\mathcal{F}, k)| = \frac{(2m)!}{2^m} \cdot k! \cdot P(2)$ only allows us to get the value of the latter polynomial in $s = 2$ which is not a large-enough value to apply the previous lemma. However, we can use a technique that is also inspired in the same paper of Valiant [76], used to show the #**P**-Completeness of a variant of SAT. Let p be a positive integer to be fixed later. Let $\mathcal{F}^p$ be the 2-CNF formula with n variables and mp clauses defined as $\mathcal{F}^p = \mathcal{F} \wedge \mathcal{F} \wedge \cdots \wedge \mathcal{F}$ (repeated p times).

Lemma 6.26 $|\rho(\mathcal{F}^p, k)| = k! \cdot \dfrac{(2mp)!}{2^{mp}} \cdot P(2^p)$

Proof. Observe that $\mathcal{F}$ and $\mathcal{F}^p$ have the same set of variables, and each clause of $\mathcal{F}$ is repeated p times on $\mathcal{F}^p$. Therefore, $\varphi(\mathcal{F}^p, k, p\mathrm{d}) = \varphi(\mathcal{F}, k, \mathrm{d})$. Moreover, if d is not a multiple of p, then $\varphi(\mathcal{F}^p, k, \mathrm{d}) = \emptyset$. Then,

$$\begin{aligned}
|\rho(\mathcal{F}^p, k)| &= \sum_{\mathrm{d}=0}^{mp} |\rho(F^p, k, \mathrm{d})| \\
&= \sum_{\mathrm{d}=0}^{mp} |\varphi(\mathcal{F}^p, k, \mathrm{d})| \cdot k! \cdot \frac{(2mp)!}{2^{mp}} \cdot 2^{\mathrm{d}} \\
&= \sum_{\mathrm{d}=0}^{m} |\varphi(\mathcal{F}^p, k, p\mathrm{d})| \cdot k! \cdot \frac{(2mp)!}{2^{mp}} \cdot 2^{p\mathrm{d}} \\
&= \sum_{\mathrm{d}=0}^{m} |\varphi(\mathcal{F}, k, \mathrm{d})| \cdot k! \cdot \frac{(2mp)!}{2^{mp}} \cdot 2^{p\mathrm{d}} \\
&= k! \cdot \frac{(2mp)!}{2^{mp}} \cdot P(2^p)
\end{aligned}$$

□

Theorem 6.27 Contagion-Probability *is #**P**-complete.*

Proof. Let $\mathcal{F}$ be an instance of #Mon-2-Sat and consider the following algorithm computing $\varphi(\mathcal{F})$ in polynomial time on a machine with an oracle for Contagion-Probability. For each $k \in [n]$, the algorithm picks $p = 2n + 1$ and constructs the input $(G[\mathcal{F}^p, k], x[\mathcal{F}^p, k], v[\mathcal{F}^p, k], t[F^p, k])$ of Contagion-Probability and queries the oracle on it, obtaining $|\rho(\mathcal{F}^p, k)|$. Then, the algorithm computes $k! \cdot \dfrac{(2mp)!}{2^{mp}}$ and divides $|\rho(\mathcal{F}^p, k)|$ by it in order to obtain the value $P(2^p)$ according to Lemma 6.26. Then, the algorithm uses as

a subroutine the algorithm given by Lemma 6.25 to obtain all the coefficients of P, which correspond to $\{|\varphi(\mathcal{F}, k, \mathrm{d})|\}_{\mathrm{d}\in\{0,\dots,m\}}$. Finally, the algorithm outputs

$$\varphi(\mathcal{F}) = \sum_{k=1}^{n}\sum_{\mathrm{d}=0}^{m} |\varphi(\mathcal{F}, k, \mathrm{d})|$$

From Proposition 1.22 and Lemma 6.25, all previous calculations can be done in polynomial time. We deduce that #Mon-2-Sat $\in FP^{\text{Contagion-Probability}}$, implying that Contagion-Probability is #**P**-Complete.

□

6.2.2 Polynomial time algorithm for maximum degree 4

In this section, we restrict ourselves to the of Bootstrap Percolation in graphs of maximum degree four. We show that in this case, unlike the general case studied in previous section, we can compute the exact probability of infecting some node in polynomial time. Roughly, this means, that we are able to efficiently count the sequences of nodes, such that, if we update them we change the state of objective node from 0 to 1 in some fixed time t. In other words, we show that problem Contagion-Probability is in **FP**. To show this result, we use a characterization given in [38], for the configurations where the objective node is unstable. This characterization involves a topological structure that can be exploited to design an efficient algorithm counting all the good sequences.

In the following, $G = ([n], E)$ is a graph of degree at most 4, x is a configuration of $\{0,1\}^n$ and v is a node such that $x_v = 0$. We call $G[0]$ the subgraph of G induced by the nodes that are healthy, i.e. $G[0] = G[\{u \in V : x_u = 0\}]$. Observe that $v \in V(G[0])$ and hence we call $G[0, v]$ the connected component of $G[0]$ containing v. The following proposition characterizes the stable configurations.

Proposition 6.28 ([41]) *Node v is stable in G if and only if v belongs to a path P in $G[0]$ such that an endpoint w of P belongs to a cycle in $G[0]$ or $\mathrm{d}_G(w) \leq 2$.*

Suppose that v is a site that is not stable in G. For each neighbor w of v in G such that $x_w = 0$, we call D_w the connected component of $G[v, 0] - v$ containing w. When $x_w = 1$, we fix $D_w = \emptyset$. The following lemma, characterizes the structure of the configuration around unstable sites.

Proposition 6.29 ([38]) *Let w be a neighbor of v such that $x_w = 0$ and w becomes active before v for some good sequence. Then D_w induces a tree of $G[0]$ where all nodes have degree at least 3 in G.*

A component D_w which is a tree where all the nodes are of degree at least 3 in G is called a *good tree*. As convention, an empty set is a good tree. Let $\mathsf{GoodNeighbors}(v)$ the set of neighbors of v that induce good trees. Observe that v admits a good sequence if the strict

majority of its neighbors induce good trees. A tree of $G[0]$ rooted at v is called a *good tree* for v, and denoted T_v, if $T_v - v$ has $\left\lfloor \frac{\mathrm{d}_G(v)}{2} \right\rfloor + 1$ components, each of them being good trees.

Definition 6.30 *Given a tree T of size n rooted at r, a* pruning sequence *of T is a bijective mapping $\rho : [n] \to V(T)$ such that, if u is an ancestor of v, then $\rho^{-1}(u) > \rho^{-1}(v)$. In particular $\rho^{-1}(r) = n$. The number of pruning sequences of T is denoted $\#\textsc{Prune}(T, r)$.*

The following lemma links up the good sequences of v with the existence of pruning subsequences of the good trees in the neighborhood of v.

Lemma 6.31 *A sequence σ is a good sequence for v if and only there is a succession $s_1 < \cdots < s_{|T_v|}$ such that $\rho(\mathrm{i}) := \sigma(s_\mathrm{i})$ is a pruning sequence a good tree T_v of v.*

Proof. Suppose first that $\sigma : [t] \to V$ is a good sequence for v and let $\{x(s)\}_{s\in[t]}$ be the succession of configurations such that $x(s) = F(x, \sigma, s)$. Since σ is a good sequence, there must exist a step $s^* \in [t]$ in which v becomes infected, i.e. such that $x(s^* - 1)_v = 0$ and $x(s^*)_v = 1$. In step $s^* - 1$ the strict majority of the neighbors of v must be infected.

From Proposition 6.29, we know that there exist a set $\{w_1, \ldots, w_k\}$ of $k = \left\lfloor \frac{\mathrm{d}_G(v)}{2} \right\rfloor + 1$ neighbors of v such that, for all $\mathrm{i} \in [k]$, node w_i induces a good tree D_i and $x(s^* - 1)_{w_\mathrm{i}} = 1$. Define $T_v = \{v\} \cup \bigcup_{\mathrm{i}\in[k]} D_\mathrm{i}$, and observe that T_v is a good tree for v. Let us call $s_1 < \cdots < s_\ell = s^*$ the sequence on which the nodes of T_v are updated, with $\ell = |T_v|$. More precisely, for each $j \in [\ell]$ we have that $\sigma(s_j) \in D_\mathrm{i}$, $x(s_j)_{\sigma(s_j)} = 1$ and $x(s_j - 1)_{\sigma(s_j)} = 0$. Observe that $\sigma(s_j)$ is a leaf of $T_v \setminus \sigma([s_j - 1])$, because for each $s \in \{s_1 < \cdots < s_\ell = s^*\}$ state of node $\sigma(s)$ is updated, meaning that the majority of its children are infected or in state 1 in time $s - 1$ and also $\sigma(s)$ must be an ancestor of $\sigma(s)$ as any node in T_v was initially set to 0. Thus, if s_j is not a leaf of $T_v \setminus \sigma([s_j - 1])$ then, it would have more than one ancestor in T_v which is not possible. We deduce that $\rho(\mathrm{i}) := \sigma(s_\mathrm{i})$ is a pruning sequence of T_v.

Conversely, if ρ is a pruning sequence of T_v, then in step $s_{|T_v|}$ vertex v becomes infected, implying that σ is a good. sequence. □

Now we introduce the main lemma of the section in which we compute Contagion-Probability for an unstable node in a graphs of maximum degree 4. Observe that, as we are considering the strict majority rule, a node that changes its state from 0 to 1 needs at least $\left\lfloor \frac{\mathrm{d}_G(v)}{2} \right\rfloor + 1$ neighbors in state one. Thus, we need to study two different cases: the case in which a node changes from 0 to 1 with exactly $\left\lfloor \frac{\mathrm{d}_G(v)}{2} \right\rfloor + 1$ neighbors in state one and the case in which there are more than that.

Lemma 6.32 *Let (G, x, v, t) be an instance of* Contagion-Probability *such that G is a graph of maximum degree 4 and v is unstable. Then,*

- *If* $|\mathsf{GoodNeighbors}(v)| = \left\lfloor \frac{\mathrm{d}_G(v)}{2} \right\rfloor + 1$, *then*

$$\textsc{Contagion-Probability}(G, x, v, t) = \alpha(T^*, v, t)$$

- *If* $|\mathsf{GoodNeighbors}(v)| > \left\lfloor \frac{\mathrm{d}_G(v)}{2} \right\rfloor + 1$, *then*

$$\textsc{Contagion-Probability}(G, x, v, t) = \alpha(T^*, v, t) + \beta(T^*, v, t)$$

where

$$\alpha(T, r, t) = \begin{cases} \binom{t}{|T|} \#\textsc{Prune}(T, r) \frac{(n-|T|)!}{(n-|T|-t)!} & \text{if } |T| \leq t \\ 0 & \text{otherwise} \end{cases},$$

$$\beta(T, r, t) = \sum_{S \subset N(v) s.t |S| = \mathrm{d}_G(v) - 1} (\alpha(T_S, v, t) - \alpha(T^*, v, t)),$$

$$T^* = \{v\} \cup \bigcup_{w \in \mathsf{GoodNeighbors}(v)} D_w \text{ and } T_S = \{v\} \cup \bigcup_{w \in S} D_w$$

Proof. First, for a given a rooted tree T, observe that $\alpha(T, r, t)$ is exactly the number of sequences of length t that contain a pruning sequence of T. Indeed, if $|T| > t$ then this number is zero. Otherwise, a sequences containing a pruning of T is constructed picking $|T|$ steps over the t possible choices, and prune T in the chosen steps. In the remaining steps any other node can be updated. The number of ways that $|T|$ steps can be picked over a total of t steps is $\binom{t}{|T|}$. The possible ways of pruning the T on those steps is $\#\textsc{Prune}(T, v)$. Finally, the number of possible choices for updating other vertices in the remaining steps is $\frac{(n-|T|)!}{(n-|T|-t)!}$. We deduce that $\alpha(T, r, v)$ is the product of previous quantities.

Lemma 6.31 implies that every good sequence for v contain a subsequence that can be mapped into a pruning sequence of a good tree for v. When $|\mathsf{GoodNeighbors}(v)| = \left\lfloor \frac{\mathrm{d}_G(v)}{2} \right\rfloor + 1$, there is only one possible choice of good tree for v, which is precisely $T^* = \{v\} \cup \bigcup_{w \in \mathsf{GoodNeighbors}(v)} D_w$. We deduce that the number of good sequences for v is $\alpha(T^*, v, t)$.

When $|\mathsf{GoodNeighbors}(v)| > \left\lfloor \frac{\mathrm{d}_G(v)}{2} \right\rfloor + 1$, then necessarily the degree of v is 3 or 4. In either case $|Good(v)| = \mathrm{d}(v)$ and any good tree for v contains $\mathrm{d}(v) - 1$ of its neighbors. Therefore, the choices for good sequences of v is the number of sequences that update contain a prune of T^*, plus all the sequences that update some good tree of v, but not all T^*. The number of sequences that contain a prune of T^* is $\alpha(T^*, v, t)$. For a given set S of three neighbors of v, the number of sequences that contain a pruning of the good trees induced by the nodes in S, but not a pruning of T^* equals $\alpha(T_S, v, t) - \alpha(T^*, v, t)$. We deduce that $\textsc{Contagion-Probability}(G, x, v, t)$ equals

$$\alpha(T^*, v, t) + \sum_{S \subset N(v) \text{s.t} |S| = 3} (\alpha(T_S, v, t) - \alpha(T^*, v, t)).$$

□

Previous lemma implies that, in order to obtain a polynomial-time algorithm solving problem CONTAGION-PROBABILITY, it is enough to have a polynomial-time algorithm computing the value of $\#\textsc{Prune}(T, r, t)$, for a given rooted tree T. The following lemma states that this is the case.

Lemma 6.33 *Given tree T rooted in vertex r with maximum degree 4, there is an algorithm computing $\#\textsc{Prune}(T, r)$ in deterministic time polynomial in $|T|$.*

PROOF. For a node u in T, let us define the *depth* of u, denoted $D(u)$, is the distance of u to the root r. We also call M the maximum depth of a node in T. The function $D(\cdot)$ can be computed in polynomial time simply running a BFS starting at node r. The *level* of a node u, denoted $L(u)$, equals $M - D(u)$. In other words, vertices at depth M are at level 0. The root is at level M.

Observe that all the nodes in level 0 are leafs, but not necessarily all leaves are at that level. For a given $u \in T$, we call T_u the subtree rooted at u containing u and all it descendants. Observe that if u is a leaf of T, then $\#\textsc{Prune}(T_u, u) = 1$. If u is not a leaf, let us call $w_1, \ldots, w_k$ the descendants of u in T, with $k = \mathrm{d}(u) - 1$. Then,

$$\#\textsc{Prune}(T_u, u) = \frac{(|T_{w_1}| + \cdots + |T_{w_k}|)!}{|T_{w_1}|! \cdots |T_{w_k}|!} \#\textsc{Prune}(T_{w_1}, w_1) \cdots \#\textsc{Prune}(T_{w_k}, w_k) \qquad (6.1)$$

Indeed, to $\#\textsc{Prune}(T_u, u)$ we have to prune all the descendants of u before u. Observe that a pruning of any two decendants of u are independent. Therefore, a pruning of T_u consist in choosing $|T_w|$ steps and a prune of T_w for each descendant w of u. For each $j \in [k]$ let $s_j = |T_{w_j}|$ and $S = s_1 + \cdots + s_k$. Then, we have $\binom{S}{s_1}\binom{S-s_1}{s_2} \cdots \binom{S-\sum_{j=1}^{k-1} s_j}{s_k} = \binom{S}{s_k} \prod_{k=2}^{n} \binom{S-\sum_{j=1}^{k-1} s_j}{s_k} = \frac{(|T_{w_1}| + \cdots + |T_{w_k}|)!}{|T_{w_1}|!, \ldots, |T_{w_k}|!}$ ways of choosing $|T_w|$ steps for every descendant w of u. For each such choice, there are

$$\#\textsc{Prune}(T_{w_1}, w_1) \cdots \#\textsc{Prune}(T_{w_k}, w_k)$$

ways of choosing a pruning of each subtree T_{w_i}.

Therefore, our algorithm computing $\#\textsc{Prune}(T, r)$ consists in a dynamic programming scheme over the levels of T. In level 0 all nodes u are leafs, and then $\#\textsc{Prune}(T_u, u) = 1$. If all the nodes al level i are already computed, the value of $\#\textsc{Prune}(T_u, u)$ for a node in level $\mathrm{i} + 1$ can is computed using Equation 6.1.

Observe that for each $u \in T$, the quantity $\#\textsc{Prune}(T_u, u)$ is at most $|T|! = \mathcal{O}(2^{|T| \log |T|})$. Therefore, from Proposition 1.22 we deduce that the expression of Equation 6.1 can be computed in time $\mathcal{O}(|T|^2)$. We conclude that $\#\textsc{Prune}(T, r)$ can be computed in time $\mathcal{O}(|T|^3)$. □

Theorem 6.34 *There is a polynomial-time algorithm solving* Contagion-Probability *restricted to the graphs of maximum degree* 4.

Proof. Let (G, x, v, t) be an input of Contagion-Probability, where G an n-node graph of maximum degree 4, x is a configuration of G, v is a node such that $x_v = 0$ and $t \leq n$.

The algorithm computes $G[0, v]$, which is the component of $G[0]$ that contains v. Let $C_1, \ldots, C_k$, with $k \leq 4$ the connected components of $G[0, v] - v$. Then, it computes $\mathsf{GoodNeighbors}(v)$ verifying which components induce good trees. If $|\mathsf{GoodNeighbors}| < \lfloor \frac{\mathrm{d}(v)}{2} \rfloor + 1$ then by Proposition 6.28 and Proposition 6.29, the algorithm outputs 0 because v is stable. Now suppose $|\mathsf{GoodNeighbors}| \geq \lfloor \frac{\mathrm{d}(v)}{2} \rfloor + 1$ then, for each $w \in \mathsf{GoodNeighbors}(v)$, compute $\#\textsc{Prune}(T_w, w)$, where T_w is the component (good tree) of $G[0, v] - v$ that contains w. Finally, the output is computed according to the expressions given in Lemma 6.32. From Proposition 1.22 and Lemma 6.33, we deduce that our algorithm runs in time $\mathcal{O}(n^3)$. □

Discussion

During this thesis work, we have developed a theoretical background to study the dynamics of automata networks from a computational complexity viewpoint. In order to achieve this task, we have studied in Chapter 2 how to precisely represent an automata network family and how to manage different representations. In addition, we have proposed different variants for prediction problem and we have formalized the study of the computational and dynamical properties of abstract families of automata networks. In addition, we have shown that both computational and dynamical properties are related thanks to the concept of simulation and particularly thanks to the study of (strongly) universal networks.

Then, in Chapter 3 we have focused on how localized behavior in automata networks can be analyzed in order to deduce global properties for a family of automata networks. We have identified the structure of a gadget and we have provided sufficient conditions for gadgets to form networks capable of coherently combine, from a dynamical viewpoint, its particular properties. Moreover, we have stated these conditions in the context of a framework which also opens new perspectives for future work. Additionally, we have presented concrete examples of useful families such as $\mathcal{G}_m$-networks and $\mathcal{G}_{m,2}$-networks which allow us to define a concrete approach in order to study automata networks consisting, on the one hand, in analyzing the period of attractors in order to find global bounds on this parameter and, on the other hand, in searching for coherent $\mathcal{G}_{m,2}$ (or $\mathcal{G}_m$)-gadgets.

In Chapter 4, we have addressed the problem of asynchronism in the dynamics by developing a framework based on the concept of projection systems and asynchronous extensions. We have presented a way to harmonize asynchronous dynamics by seeing it as a projection of a synchronous dynamics defined over a large alphabet which contains the information on how to updated different nodes at each time step.

Then, in Chapter 5, we have applied the theoretical framework developed on the latter chapters in order to systematically study a set of concrete symmetric automata network (CSAN) families under different update schemes. By constructing coherent gadgets and analyzing dynamical properties, we have found some kind of *marker* for universality based on an interesting trade-off between constraints in the update scheme and the simplicity of rules. This observation combined with the description and properties of CSAN rules, suggest that update schemes play an impactful role in changing the interaction graph by affecting its symmetry and thus, symmetry plays an important role on the global properties of automata networks.

Finally, on chapter 6, we turned our attention to the structure of the interaction graph in

the context of the study of freezing dynamics.

In first place, we have established how alphabet size, treewidth and maximum degree of the underlying graph are key parameters which have an impact on the overall computational complexity of finite freezing automata networks. We have accomplished this by defining the Specification Checking Problem, that captures many classical decision problems such as prediction, nilpotency, predecessor, asynchronous reachability. We have presented a fast-parallel algorithm that solves this latter problem when the three parameters are bounded, hence showing that the problem is in **NC**. Moreover, we have shown that the problem is in **XP** on the parameters tree-width and maximum degree. Finally, we have hown that these problems are hard from two different perspectives. First, the general problem is **W[2]**-hard when taking either treewidth or alphabet as single parameter and fixing the others. Second, the classical problems are hard in their respective classes when restricted to families of graph with sufficiently large treewidth. Moreover, for prediction, predecessor and asynchronous reachability, we establish the hardness result with a fixed set-defined update rule that is universally hard on any input graph of such families.

In the second part of this chapter we have studied the computational complexity of the problem Contagion-Probability. In general, we have shown that Contagion-Probability is #**P**-complete. Roughly speaking, this means there is no better strategy for computing the probability of an inactive node to change to state 1 than simply simulate the system for each possible updating sequence σ in order to verify how many of them actually change the state of objective node. However, when we consider networks with maximum degree 4, we have shown, based on previous results that characterize good sequences in terms of underlying graph topology [38], that latter computation can be made in polynomial time.

Perspectives

Regarding our framework for glueing of automata networks it would be relevant to understand the properties of a glueing process defined generally and see what kinds of properties can be deduced from it when considered as a process beyond the given conditions assuring coherence of dynamics. Particularly, given two abstract automata networks belonging to some particular families it would interesting to study at which point dynamical properties are preserved when two networks are glued. In addition, it would be very interesting to explore if latter process can be seen in the opposite way, i.e., given an automata network, determine if it is possible to decompose the network in blocks satisfying some particular properties as gadgets do. Additionally, Proposition 3.19 provides an interesting starting point to explore the link between different gate sets from the viewpoint of analyzing the richness of their synchronous closure. This latter observation could lead to establish some sort of hierarchy between sets of gates. In this sense, it would be interesting to study, for example, reversible gate sets such as Toffoli or Fredkin gates.

On the other hand, as we have pointed out in Chapter 4, it would be interesting to study other types of update schemes which are not captured by general periodic update schemes such as reaction-diffusion systems (see, for instance, [30]) or the most permissive semantics introduced in [15]. In both cases the orbits are not given by a choice in the possible nodes

that can be updated according to local rule. In that sense, it would be interesting to adapt the concept of asynchronous extension in order to capture the latter update schemes.

Regarding our results on freezing automata networks, we observe that our algorithm for the general model checking problem is not as efficient as known algorithms for specific subproblems [11] and it would be interesting to establish hardness results in the **NC** hierarchy to make this gap more precise. In the same vein, our algorithm does not yield fixed parameter tractability results for any of the parameters (treewidth, degree, alphabet), and we wonder whether our hardness results in the framework of parameterized complexity [23] could be improved. We could also consider intermediate treewidth classes (non-constant but subpolynomial). Concerning these complexity questions, we think that considering other (more restrictive) parameters like pathwidth could definitely help to obtain better bounds.

Besides, one might wonder whether the set of dynamical properties that are efficiently decidable on graphs of bounded degree and treewidth could be in fact much larger than what gives our model checking formalism. This question remains largely open, but we can already add ingredients in our formalism (for instance, a relational predicate representing the input graph structure). However, we conjecture that there are **NP**-hard properties for freezing automata network on trees of bounded degree that can be expressed in the following language: first order quantification on configurations together with a reachability predicate (configuration y can be reached from x in the system).

In addition, we think that we can push our algorithm further and partly release the constraint on maximum degree (for instance allowing a bounded number of nodes of unbounded degree). This can however not work in the general model checking setting as shown in Remark 6.4.

Concerning asynchronous bootstrap percolation, the first natural question that arises is whether this threshold in the maximum degree of the network is tight, or in other words: is Contagion-Probability still #**P**-complete even when restricted to instances of maximum degree 5? Observe that gadgets in section 3 strongly use high connectivity in graph $G[\mathcal{F}, k]$ in order to assure that good sequences will respect a fixed order while updating different nodes in the network (first we update variable nodes, then we update threshold gadget, etc.). This is a key aspect of the proof since it helps us to keep control in the amount different sequences associated to one particular assignation satisfying the Boolean formula. This shows that even when one could conjecture a way to implement constant degree gadgets (by for example considering more copies of smaller gadgets) there should also be a way to keep track in all possible good sequences that implement the same assignation satisfying the original Boolean formula.

Additionally, it could be also interesting to study a different version of problem Contagion-Probability but now considering some other model having a different family of automta networks such as CSAN networks, particularly, conjunctive or disjuntive networks, algebraic networks or totalistic networks. In this regard, we recall that for some of the latter automata networks, there exist efficient algorithms (polynomial or even fast parallel algorithms) in order to compute the dynamics of the system (for example, in the case of algebraic networks or even conjunctive networks, it suffices to compute powers of some matrix). However, this does not necessarily implies that counting version of Good-Sequence

would be efficiently solvable.

In addition, we would like to note that the previous results are also interesting in the context of counting decision problems. Within this framework, we are no longer interested in computing the number of all possible polynomial size certificates but to determine whether, for some fixed input, *the majority* of polynomial size strings we give to the verifier along with the input are actually certificates. This notion defines the complexity class **PP**. More precisely a language $L \subseteq \{0,1\}^*$ is in **PP** if and only if there exists a polynomial function $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time verifier V such that $x \in L \iff |y \in \{0,1\}^{p(|x|)} : V(x,y) = 1| \geq \frac{1}{2} \cdot 2^{p(|x|)}$. By considering this formalism, a natural question arises: is the decision version of Contagion-Probability which consists in deciding if the probability of transmission for objective node is at least $\frac{1}{2}$, **PP**-complete? (observe that by definition it is in **PP**). In this context, it is known that problem MAJ-SAT, which consist on deciding if the majority of all assignments of some Boolean formula will satisfy it, is **PP**-complete and thus, it could be very interesting to explore as future work if an application of the same techniques we have used in Section 3 might produce a polynomial-time reduction from latter problem to decision version of Contagion-Probability.

Finally, as we have shown that Contagion-Probability is #**P**-complete, it could be interesting to study approximations. In particular, within the framework of applications one could ask whether this problem admits a polynomial randomized approximation scheme.

Bibliography

[1] Hamed Amini and Nikolaos Fountoulakis. Bootstrap percolation in power-law random graphs. *Journal of Statistical Physics*, 155(1):72–92, feb 2014.

[2] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, April 1987.

[3] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[4] Per Bak, Kan Chen, and Chao Tang. A forest-fire model and some thoughts on turbulence. *Physics Letters A*, 147(5):297 – 300, 1990.

[5] József Balogh and Béla Bollobás. Bootstrap percolation on the hypercube. *Probability Theory and Related Fields*, 134(4):624–648, jul 2005.

[6] József Balogh, Béla Bollobás, Hugo Duminil-Copin, and Robert Morris. The sharp threshold for bootstrap percolation in all dimensions. *Transactions of the American Mathematical Society*, 364(5):2667–2701, may 2012.

[7] József Balogh and Gábor Pete. Random disease on the square grid. *Random Structures & Algorithms*, 13(3-4):409–422, 1998.

[8] Edwin Roger Banks. Universality in cellular automata. In *11th Annual Symposium on Switching and Automata Theory (swat 1970)*, pages 194–215. IEEE, 1970.

[9] Florent Becker, Diego Maldonado, Nicolas Ollinger, and Guillaume Theyssier. Universality in freezing cellular automata. In *Sailing Routes in the World of Computation - 14th Conference on Computability in Europe, CiE 2018, Kiel, Germany, July 30 - August 3, 2018, Proceedings*, pages 50–59, 2018.

[10] Hans L. Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM Journal on Computing*, 27(6):1725–1746, December 1998.

[11] S. R. Buss. The boolean formula value problem is in ALOGTIME. In *Proceedings of the nineteenth annual ACM conference on Theory of computing - STOC '87*. ACM Press, 1987.

[12] T. Calamoneri. The l(h,k)-labelling problem: A survey and annotated bibliography. *The Computer Journal*, 49(5):585–608, February 2006.

[13] Olivier Carton, Bruno Guillon, and Fabian Reiter. Counter machines and distributed automata. In *Cellular Automata and Discrete Complex Systems*, pages 13–28. Springer International Publishing, 2018.

[14] John Chalupa, Paul L Leath, and Gary R Reich. Bootstrap percolation on a bethe lattice. *Journal of Physics C: Solid State Physics*, 12(1):L31, 1979.

[15] Thomas Chatain, Stefan Haar, Loïc Paulevé, et al. Most permissive semantics of boolean networks. *arXiv preprint arXiv:1808.10240*, 2018.

[16] Chandra Chekuri and Julia Chuzhoy. Polynomial bounds for the grid-minor theorem. *Journal of the ACM*, 63(5):1–65, dec 2016.

[17] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, mar 1990.

[18] Víctor Dalmau, Phokion G Kolaitis, and Moshe Y Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *International Conference on Principles and Practice of Constraint Programming*, pages 310–326. Springer, 2002.

[19] Bart De Schutter and Bart De Moor. On the sequence of consecutive powers of a matrix in a boolean algebra. *SIAM Journal on Matrix Analysis and Applications*, 21(1):328–354, 1999.

[20] Jacques Demongeot and Sylvain Sené. About block-parallel boolean networks: a position paper. *Natural Computing*, 19(1):5–13, 2020.

[21] Alberto Dennunzio, Enrico Formenti, Luca Manzoni, Giancarlo Mauri, and Antonio E. Porreca. Computational complexity of finite asynchronous cellular automata. *Theoretical Computer Science*, 664:131–143, February 2017.

[22] Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness i: Basic results. *SIAM Journal on Computing*, 24(4):873–921, aug 1995.

[23] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer London, 2013.

[24] Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of bodlaender and courcelle. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*. IEEE, October 2010.

[25] M.A. Fuentes and M.N. Kuperman. Cellular automata and epidemiological models with spatial dependence. *Physica A: Statistical Mechanics and its Applications*, 267(3–4):471 – 486, 1999.

[26] Maximilien Gadouleau. On the influence of the interaction graph on a finite dynamical

system. *Natural Computing*, to appear.

[27] Maximilien Gadouleau and Adrien Richard. Simple dynamics on graphs. *Theoretical Computer Science*, 628:62–77, may 2016.

[28] Andrew Goldberg, Serge Plotkin, and Gregory Shannon. Parallel symmetry-breaking in sparse graphs. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 315–324, 1987.

[29] Leslie M Goldschlager. The monotone and planar circuit value problems are log space complete for p. *ACM sigact news*, 9(2):25–29, 1977.

[30] E Goles and Martin Matamala. Reaction-diffusion automata: Three states implies universality. *Theory of Computing Systems*, 30(3):223–229, 1997.

[31] E. Goles, N. Ollinger, and G. Theyssier. Introducing freezing cellular automata. In *Exploratory Papers of Cellular Automata and Discrete Complex Systems (AUTOMATA 2015)*, pages 65–73, 2015.

[32] Eric Goles, Fabiola Lobos, Gonzalo A Ruz, and Sylvain Sené. Attractor landscapes in boolean networks with firing memory: a theoretical study applied to genetic networks. *Natural Computing*, 19:295–319, 2020.

[33] Eric Goles, Diego Maldonado, Pedro Montealegre, and Nicolas Ollinger. On the computational complexity of the freezing non-strict majority automata. In *Cellular Automata and Discrete Complex Systems - 23rd IFIP WG 1.5 International Workshop, AUTOMATA 2017, Milan, Italy, June 7-9, 2017, Proceedings*, pages 109–119, 2017.

[34] Eric Goles, Diego Maldonado, Pedro Montealegre, and Martín Ríos-Wilson. On the complexity of asynchronous freezing cellular automata. *arXiv preprint arXiv:1910.10882*, 2019.

[35] Eric Goles, Diego Maldonado, Pedro Montealegre-Barba, and Nicolas Ollinger. Fast-parallel algorithms for freezing totalistic asynchronous cellular automata. In *Cellular Automata - 13th International Conference on Cellular Automata for Research and Industry, ACRI 2018, Como, Italy, September 17-21, 2018, Proceedings*, volume 11115 of *Lecture Notes in Computer Science*, pages 406–415. Springer, 2018.

[36] Eric Goles and Servet Martínez. *Neural and Automata Networks: Dynamical Behavior and Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1990.

[37] Eric Goles and Pedro Montealegre. Computational complexity of threshold automata networks under different updating schemes. *Theoretical Computer Science*, 559:3–19, 2014.

[38] Eric Goles and Pedro Montealegre. The complexity of the asynchronous prediction of the majority automata. *Information and Computation*, page 104537, 2020.

[39] Eric Goles, Pedro Montealegre, and Martín Ríos-Wilson. On the effects of firing memory

in the dynamics of conjunctive networks. *Discrete & Continuous Dynamical Systems - A*, 40(10):5765–5793, 2020.

[40] Eric Goles, Pedro Montealegre, Ville Salo, and Ilkka Törmä. Pspace-completeness of majority automata networks. *Theoretical Computer Science*, 609:118–128, 2016.

[41] Eric Goles, Pedro Montealegre-Barba, and Ioan Todinca. The complexity of the bootstraping percolation and other problems. *Theoretical Computer Science*, 504:73–82, 2013.

[42] Eric Goles-Chacc, Françoise Fogelman-Soulié, and Didier Pellegrin. Decreasing energy functions as a tool for studying threshold networks. *Discrete Applied Mathematics*, 12(3):261–277, 1985.

[43] Janko Gravner and David Griffeath. Cellular automaton growth on z2: Theorems, examples, and problems. *Advances in Applied Mathematics*, 21(2):241 – 304, 1998.

[44] Frederic Green. NP-complete problems in cellular automata. *Complex Systems*, 1, 01 1987.

[45] Raymond Greenlaw, H James Hoover, Walter L Ruzzo, et al. *Limits to parallel computation: P-completeness theory*. Oxford University Press on Demand, 1995.

[46] D. Griffeath and C. Moore. Life without death is P-complete. *Complex Systems*, 10, 1996.

[47] Godfrey Harold Hardy, Edward Maitland Wright, et al. *An introduction to the theory of numbers*. Oxford university press, 1979.

[48] Alexander E. Holroyd. Sharp metastability threshold for two-dimensional bootstrap percolation. *Probability Theory and Related Fields*, 125(2):195–224, 2003.

[49] J Iirgen Albert and Karel Culik II. A simple universal cellular automaton and its one-way and totalistic version. *Complex Systems*, 1:1–16, 1987.

[50] Joseph JáJá. *An Introduction to Parallel Algorithms*. Addison Wesley Longman Publishing Co., Inc., USA, 1992.

[51] J. Kari. The Nilpotency Problem of One-dimensional Cellular Automata. *SIAM Journal on Computing*, 21:571–586, 1992.

[52] Akinori Kawachi, Mitsunori Ogihara, and Kei Uchizawa. Generalized predecessor existence problems for boolean finite dynamical systems on directed graphs. *Theoretical Computer Science*, 762:25–40, March 2019.

[53] William Ogilvy Kermack and A. G. McKendrick. A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 115(772):700–721, aug 1927.

[54] D. KNOP, M. KOUTECKÝ, T. MASAŘÍK, and T. TOUFAR. Simplified algorithmic metatheorems beyond mso: Treewidth and neighborhood diversity. *Logical Methods in Computer Science ; Volume 15*, pages Issue 4 ; 1860–5974, 2019.

[55] Stephan Kreutzer and Siamak Tazari. On brambles, grid-like minors, and parameterized intractability of monadic second-order logic. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 354–364. SIAM, 2010.

[56] Martin Kutrib and Andreas Malcher. Cellular automata with sparse communication. *Theor. Comput. Sci.*, 411(38-39):3516–3526, 2010.

[57] Kristian Lindgren and Mats G Nordahl. Universal computation in simple one-dimensional cellular automata. *Complex Systems*, 4(3):299–318, 1990.

[58] Noam Livne. A note on -completeness of NP-witnessing relations. *Information Processing Letters*, 109(5):259–261, February 2009.

[59] Dániel Marx. Can you beat treewidth? In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 169–179. IEEE, 2007.

[60] Jacques Mazoyer and Ivan Rapaport. Inducing an order on cellular automata by a grouping operation. *Discrete Applied Mathematics*, 91(1-3):177–196, 1999.

[61] Cristopher Moore. Majority-vote cellular automata, ising dynamics, and p-completeness. *Journal of Statistical Physics*, 88(3/4):795–805, August 1997.

[62] Mathilde Noual. *Updating automata networks*. PhD thesis, École normale supérieure de Lyon, 2012.

[63] Mathilde Noual and Sylvain Sené. Synchronism versus asynchronism in monotonic Boolean automata networks. *Natural Computing*, 17:393–402, 2018.

[64] Nicolas Ollinger. Toward an algorithmic classification of cellular automata dynamics. *Reasearch report*, 10, 2001.

[65] Nicolas Ollinger. The intrinsic universality problem of one-dimensional cellular automata. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 632–641. Springer, 2003.

[66] Nicolas Ollinger and Guillaume Theyssier. Freezing, bounded-change and convergent cellular automata. *CoRR*, abs/1908.06751, 2019.

[67] Loïc Paulevé and Sylvain Sené. Non-deterministic updates of Boolean networks. 2021. Submitted.

[68] Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4):1–24, September 2008.

[69] Élisabeth Remy, Paul Ruet, and Denis Thieffry. Graphic requirements for multistability and attractive cycles in a boolean dynamical framework. *Advances in Applied Mathematics*, 41(3):335–350, 2008.

[70] Adrien Richard. Nilpotent dynamics on signed interaction graphs and weak converses of thomas' rules. *Discrete Applied Mathematics*, 267:160–175, aug 2019.

[71] Neil Robertson and P.D Seymour. Graph minors. v. excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92–114, aug 1986.

[72] Marko Samer and Stefan Szeider. Constraint satisfaction with bounded treewidth revisited. *Journal of Computer and System Sciences*, 76(2):103–114, 2010.

[73] Stefan Szeider. On fixed-parameter tractable parameterizations of SAT. In *Theory and Applications of Satisfiability Testing*, pages 188–202. Springer Berlin Heidelberg, 2004.

[74] Michael Treaster, William Conner, Indranil Gupta, and Klara Nahrstedt. Contagalert: using contagion theory for adaptive, distributed alert propagation. In *Fifth IEEE International Symposium on Network Computing and Applications (NCA'06)*, pages 126–136. IEEE, 2006.

[75] S. M. Ulam. On some mathematical problems connected with patterns of growth of figures. In A. W. Bukrs, editor, *Essays on Cellular Automata*, pages 219–231. U. of Illinois Press, 1970.

[76] Leslie G Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[77] R. Vollmar. On cellular automata with a finite number of state changes. In *Parallel Processes and Related Automata / Parallele Prozesse und damit zusammenhängende Automaten*, volume 3 of *Computing Supplementum*, pages 181–191. Springer Vienna, 1981.

[78] Andrew Winslow. A brief tour of theoretical tile self-assembly. In *Cellular Automata and Discrete Complex Systems - 22nd IFIP WG 1.5 International Workshop, AUTOMATA 2016, Zurich, Switzerland, June 15-17, 2016, Proceedings*, pages 26–31, 2016.

[79] Angela Wu and Azriel Rosenfeld. Cellular graph automata. i. basic concepts, graph property measurement, closure properties. *Information and Control*, 42(3):305 – 329, 1979.

[80] Angela Wu and Azriel Rosenfeld. Cellular graph automata. ii. graph and subgraph isomorphism, graph structure recognition. *Information and Control*, 42:330–353, 09 1979.