Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Control System engineering

Fault Detection for Boolean Systems

Laureando:
Piero Simonetto

Relatore:
Prof. Maria Elena Valcher

ANNO ACCADEMICO 2021/2022

Data di laurea 10/10/2022

Abstract

This thesis aims at presenting a mathematical model (o environment) to describe the biological reactions inside a cell. In particular, we want to analyze the iterations between genes and proteins. A simplification of the dynamics is introduced in order to obtain a discrete and quantized system. In particular, we want to create a model based on Boolean functions. In order to capture more complex behaviors, models based on boolean functions are extended to account for some sort of probabilistic behavior, such as a noise or a non-constant update functions. Subsequently, the mathematical model is used to monitor the process in question, in order to identify any fault in it.

Analysing the biological world by means of the logical-Boolean approach is a compromise solution that tries to achieve a reasonable precision in describing certain dynamics, even when not all the describing parameters are known. Despite this, the use of a Boolean model is important in order to obtain useful information for the creation of more complex models.

Sommario

Questa tesi ha lo scopo di presentare un modello matematico al fine di descrivere le reazioni biologiche interne a una cellula. In particolare si vuole analizzare le iterazioni tra geni e proteine. Si introduce una semplificazione delle dinamiche al fine di ottenere un modello discreto e quantizzato. In particolare si vuole creare un modello basato su funzioni booleane. Al fine di integrare comportamenti più complessi, verrà integrato il concetto di rumore e di funzioni non constanti nel tempo. Successivamente, viene utilizzato il modello matematico per tenere sotto controllo il processo in questione, al fine di cercare di identificare eventuali guasti nello stesso.

Analizzare il mondo biologico con l'approccio logico-booleano è un compromesso tra il non dover conoscere tutti i parametri necessari a descrivere le iterazioni e la precisione con cui vengono descritte le dinamiche in questione. Nonostante questo, l'utilizzo di un modello booleano è importante al fine di ricavare informazioni utili alla creazione di modelli più complessi.

Contents

1 Introduction **1**

1.1 Framework 1

1.2 Real world data 2

1.3 Chemical Master Equation 3

1.4 Boolean Models 3

1.5 Relations with pathologies 4

1.6 Thesis organization 4

2 Boolean Models **5**

2.1 Semi Tensor Product and logic in matrix form 6

2.2 Boolean systems 10

2.3 Dynamical system 11

2.4 Trajectory set 23

3 Beyond deterministic boolean networks **25**

3.1 Stochastic model 25

3.2 Probabilistic Boolean Network 29

3.3 Asynchronous Step Update Scheme 33

4 Models for fault detection **39**

4.1 Output of the system 40

4.2 Types of fault 42

4.3 State estimator 43

4.4 Fault detection observer 45

4.5 Boolean Kalman filter 48

5 Simulation on a real model **53**

6 Conclusion **61**

Chapter 1

Introduction

In this chapter, we will present the key topics to get acquainted with the problem that the thesis attempts to face. We start by introducing a brief overview of the biological behaviors within a cell and of the techniques commonly used to collect real world data. Then, we present the mathematical techniques commonly used to describe the biological mechanism. We conclude by explaining how we want to use these notions to identify genome-related pathologies.

1.1 Framework

All living organisms are made of *cells*. Depending on the organism or the function inside the organism, each cell can vary in shape and size, but the chemistry and basic principles remain common [1]. The cell is composed of different parts, some of them are used to interact with the external world and others to regulate the internal behavior. For this thesis, it is important to cite the presence of two types of amino acid sequences, the DNA and RNA. The first is a "static" sequence that lasts throughout the life cycle of the cell, the RNA instead is only temporary. In addition to these sequences, the amino acids compose also some other actors of the cell life circle: the proteins. The proteins are used for different functions inside and outside the cell. For example, they catalyze metabolic reactions, encode specific signals and regulate the cell processes. The basic principle that connects these components is called *central dogma* [2]. This dogma states that the information needed to build proteins is encoded in some particular DNA subsections called genes. These pieces of information are transported outside the DNA through the RNA and are later used for other amino acid sequences. These new sequences fold into proteins, which regulate the cell's internal and external behaviors. In other words inside the genes all the information required for the life of the organism is available. It turns out that protecting the DNA is very important to avoid transcription errors, which can lead to malfunctioning or cell death. In some organisms, evolution has built a protected area that maintains the DNA stable. This area is called the cell nucleus. An organism with this feature is called eukaryote and is typical of an organism composed of multiple cells. Otherwise, it takes the name of prokaryote, typical of single-cell organisms as bacteria. The presence of the nucleus leads to a more complex transcription of the DNA to RNA for eukaryotes cells, but the dogma remains unchanged. It is evident that the relations between all of these components are not random: they follow some particular paths. Biology calls these relationships *gene regulator networks* (GRNs). The study of GRNs is the main topic of this thesis.

1.2 Real world data

The first problem that biologists have faced is how to observe and collect data from reactions inside a $1 - 100\mu m$ system. The second problem is how to acquire them systematically and automatically. With the help of techniques from bioinformatics, the following procedure was created: using microarrays or other technologies, take some snapshots of the cell composition at various time steps. Use these snapshots to feed particular algorithms that extract correlation between the objects of study. Based on this correlation, the biologists design some experiments to demonstrate that they are correlated to some chemical evolution and not casual events. In the end, a scheme that relates interactions and describes the various components is generated. As previously reported, this model takes the name of "gene regulator network". An example of this network is reported in Figure 1.1:

In this type of representation, the symbol $\rightarrow$ means that the first component induces an increased presence of the second, instead, $\dashv$ the first component induces a decreased presence of the second one. We need to generate a formal mathematical representation for this type of model. Different approaches have been applied, all with pros and cons. The main ones are the chemical master equation and booleans models.

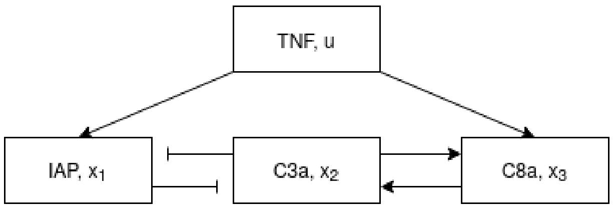


Figure 1.1: A simple GNR.

1.3 Chemical Master Equation

The chemical master equation approach transforms the graph into a system of first-order parametric differential equations. These equations explain the functionality of molecules, in particular how their concentration changes depending on the presence of other molecules. The variables describe the quantities of each reagent and each product at a given time, taking into account the probability that all reagents are available at the same time in a given space. Indeed the presence of all reagents is strictly necessary for the chemical reaction to take place.

Without going into too much detail, the chemical master equation approach has the advantage of precisely describing the evolution of the components inside the cell, but it requires to know every parameter internal to the system itself. These parameters are usually too many to be possible to obtain an adequate estimate for each of them.

1.4 Boolean Models

To mitigate these difficulties, in 1993 Kauffman proposed to simplify the modelling approach, moving to a discrete time boolean representations [3]. The details of this approach will be discussed in the next chapter, but they are based on the following consideration: the molecules generally exhibit simple dynamics. A gene that produces the corresponding mRNA is said to be an active gene, otherwise, it is said to be inactive. The activation is related to the presence or absence of biological signals usually in the form of proteins. All of these behaviors: active and inactive, presence and absence, can be encoded in a binary dynamics. Instead of modeling all the complex dynamics, as in the chemical master equation approach, Kauffman proposes to synthesize the internal dynamics as a logical circuit, where the logical function is the chemical reaction or transformation inside the cell, and the encoded variables that are the input-output of the circuit are the boolean states of the biological actor. This kind of model is also used to study how a cell relates with other cells or populations of cells.

1.5 Relations with pathologies

When a model of GRN has been generated and it has been proved to be consistent with the phenomenon it aims to represent, the focus switches to how to use this object? Lots of biological studies aim to understand how to relate certain pathologies to some behaviors of the cell. The goal is to determine if some configurations of the components' states are related to some pathologies or if a pathology can lead to a different time evolution with respect to the one described by the model.

Consider the first problem. According to [4], the insurgence of a pathology introduces a change in the internal cell configuration. Our focus is to deduce the conditions that lead to a stable cell configuration or the circumstances that lead to a significant change thereof. The second problem is described in literature as a fault detection problem. To detect a fault one needs to compare the real system evolution with the prediction of the model to verify when a discrepancy arises. The question is if it is possible to understand if and where the model stopped behaving as the real system that it tries to reproduce. Once the "faulty" part is found, it is used in specific studies to understand what happened and how to act to restore the normal behavior, if possible.

1.6 Thesis organization

The rest of the thesis is organized as follows: chapter 2 introduces boolean models and boolean systems. In chapter 3 there are the foundations to move from a deterministic boolean model to a stochastic approach, to encapsulate more complex dynamics. Chapter 4 is about fault detection models for boolean dynamics. In this chapter, we present the common faulty behaviors for GRNs and their analysis. Then in chapter 5 the previous techniques are applied, to explore a real complex problem.

Chapter 2

Boolean Models

Let us consider the boolean set $\mathcal{B} = \{0, 1\}$.

Definition 2.0.1. A variable α is a *boolean variable* if $\alpha \in \mathcal{B}$.

Definition 2.0.2. A map acting on n boolean variables: $F : \mathcal{B}^n \to \mathcal{B}$ is called a *boolean function*.

Usually a boolean function is a combination of boolean operators. The most common boolean operators are :

- NOT: the negation operator, indicated with $\neg\alpha$, returns the opposite value of the boolean variable α.
- OR: the conditional operator between two boolean variables α and β, indicated as $\alpha \vee \beta$, returns 1 if at least one of the two variables is 1.
- AND: the conjunction operator between two boolean variables α and β, indicated as $\alpha \wedge \beta$, returns 1 if both variables are 1.

These operators are also called *basic operations* of the boolean algebra.

Example 2.0.1. Given $\alpha, \beta, \gamma \in \mathcal{B}$ the expression:

$$\rho = \alpha \wedge (\neg\beta \vee \gamma)$$

is a boolean function: $\rho = F(\alpha, \beta, \gamma)$, $F : \mathcal{B}^3 \to \mathcal{B}$.

Alternatively, each boolean function $F : \mathcal{B}^n \to \mathcal{B}$ can also be expressed with a truth table. This table consists of 2^n lines. Each line corresponds to a different combination of the inputs with the corresponding function value. The truth table of the Example 2.0.1 is reported in the Table 2.1.

Definition 2.0.3. If two different boolean functions have the same truth table, they are said to be *equivalent.*

α	β	γ	ρ
1	1	1	1
1	1	0	0
1	0	1	1
1	0	0	1
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0

Table 2.1: Truth table of the Example 2.0.1

In addition to the basic operations, it is important to report some particular boolean functions called *secondary operations*, all of them *binary*, namely involving two logical variables α and β:

- XOR: The "exclusive or" denoted by $\alpha \bar{\vee} \beta$, returns 1 if only one of the two variables is 1
- Material conditional: The operation between α and β, denoted by $\alpha \to \beta$, returns the value of β if α is 1, 1 otherwise.
- Logical equivalence: The operation, denoted by $\alpha \equiv \beta$, returns 1 if and only if the two variables have the same state.

2.1 Semi Tensor Product and logic in matrix form

It is possible to introduce a matrix representation for the boolean functions. To do this it is necessary to present a new operation between matrices called the *semi tensor product* that is based on the Kronecker product.

The Kronecker product of two matrices $A \in \mathcal{B}^{m\times n}$ and $B \in \mathcal{B}^{p\times q}$ is defined as:

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix} \in \mathcal{B}^{(mp)\times(nq)} \tag{2.1}$$

where a_{ij} is the (i, j)th element of A.

Definition 2.1.1. Given two boolean matrices $A \in \mathcal{B}^{m\times n}$ and $B \in \mathcal{B}^{p\times q}$, their *semi tensor product* (STP) is defined as:

$$A \ltimes B = (A \otimes I_{\alpha/n})(B \otimes I_{\alpha/p}), \tag{2.2}$$

where α the least common multiple of n and p.

This operation is a generalization of the common matrix multiplication. Indeed, if $n = p$ then $\alpha = 1$ and $(A \otimes 1)(B \otimes 1) = AB$. In addition , it is possible to multiply matrices of different sizes.

Example 2.1.1. Given $A = [a_1\ a_2\ a_3]^T$ and $B = [b_1\ b_2]^T$, the STB between them is:

$$A \ltimes B = (A \otimes I_2)B = \begin{bmatrix} a_1 & 0 \\ 0 & a_1 \\ a_2 & 0 \\ 0 & a_2 \\ a_3 & 0 \\ 0 & a_3 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_1b_1 \\ a_1b_2 \\ a_2b_1 \\ a_2b_2 \\ a_3b_1 \\ a_4b_2 \end{bmatrix}$$

The STP has been defined for all kinds of vectors and matrices, but it encodes specific properties if used with boolean vectors. In that case it is possible to encode the information of the vector inside the position of the non-zero value inside the array. A boolean two-dimensional vector is an equivalent representation of a boolean scalar variable. In particular, the columns of the identity matrix are used to encode the boolean symbols:

$$1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad 0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.3}$$

This representation is called *algebraic form* of the boolean variables. Consider now n different boolean variables $a_1, a_2, \cdots, a_n$ expressed in algebraic form as in (2.3). It is possible to define a new cumulative vector as:

$$x = \ltimes_{i=1}^{n} a_1 \tag{2.4}$$

This vector has size 2^n and and it is canonical, namely all its entries are 0 except for one which is unitary. This position is unique for each configuration of the boolean variables a_i.

Example 2.1.2. Given the boolean variables: $a_1 = \begin{bmatrix} 1 & 0 \end{bmatrix}^T, a_2 = \begin{bmatrix} 0 & 1 \end{bmatrix}^T, a_3 = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$, the vector $x = a_1 \ltimes a_2 \ltimes a_3$ becomes:

$$(a_1 \otimes I_2)a_2 \ltimes a_3 = (\begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}) \begin{bmatrix} 0 \\ 1 \end{bmatrix} \ltimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \ltimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \ltimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} =$$

$$(\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \otimes I_2) \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Instead to report the full vector as in the Example 2.1.2 it is possible to use a standard representation for a canonical vector: $\delta_{2^n}^i$, where 2^n is the size of the vector and i the index of the unitary entry. If the number of variables is known, the size can be omitted. The symbol Δ^k denotes the set $\{\delta_k^i, i \in [1, k]\}$, where k is the size of the vectors. In this framework k will be equal to 2^n. The following lemma synthesise the previous discussion in a single statement:

Lemma 2.1.1. Given n boolean variables in algebraic form: $a_1, a_2, \cdots, a_n$; $a_i \in \Delta^2$, let $x \in \Delta^{2^n}$ be defined as in (2.4). The n a_i vectors can be uniquely determined from x.

Proof: Given $x \in \Delta^{2^n}$, proceed in two steps: split the vector x in two parts of identical size $x = \begin{bmatrix} x_1^T & x_2^T \end{bmatrix}^T$. Since x is canonical only one between x_1 and x_2 is a non-zero vector. If x_1 is non-zero, a_1 is δ_2^1 and the previous splitting procedures need to be iterated for $x = x_1$. Otherwise, a_1 is δ_2^2 and the process needs to be iterated with $x = x_2$. One can continue with this procedure until x becomes a two dimensional canonical vector.

Once we assume for the boolean variables the algebraic form, it is possible to introduce a third representation for a boolean function: the matrix form. The idea is the following one: a boolean function $F : \mathcal{B}^n \to \mathcal{B}$ is converted into a matrix $M^{2\times 2^n}$ based on the information of the truth table. Instead of mapping each single variable, the cumulative algebraic vector is used:

$$b = M \ltimes x \tag{2.5}$$

To generate M one can proceed as follows: starting from the matrix $\mathbf{0}^{2\times 2^n}$, where n is the number of boolean variables involved, insert as ith column the output vector associated with the input vector $\delta_{2^n}^i$. As for the truth table, this matrix is uniquely determined for each function.

Definition 2.1.2. A matrix $M \in \mathcal{B}^{n\times m}$ is said to be a *logical matrix* if each of its columns is a canonical vector.

A function represented as in (2.5) is said to be in *logical form.*

Example 2.1.3. Consider the boolean function of the Example 2.0.1:

$$\rho = \alpha \wedge (\neg\beta \vee \gamma)$$

First it is necessary to map each combination of values of α, β, γ to a canonical vector δ^i. For this example it is used the map reported in Table 2.2. The corresponding logical matrix is:

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{2.6}$$

α	β	γ	δ^i
1	1	1	δ^1
1	1	0	δ^2
1	0	1	δ^3
1	0	0	δ^4
0	1	1	δ^5
0	1	0	δ^6
0	0	1	δ^7
0	0	0	δ^8

Table 2.2: Map of canonical vector and variables configuration of the Example 2.1.3

2.2 Boolean systems

Given a set of boolean functions:

$$\begin{cases} \beta_1 & = F_1(\alpha_1, \cdots, \alpha_l) \\ \beta_2 & = F_2(\alpha_1, \cdots, \alpha_l) \\ & \vdots \\ \beta_n & = F_n(\alpha_1, \cdots, \alpha_l) \end{cases} \tag{2.7}$$

where each F_i is a boolean function $F : \mathcal{B}^l \to \mathcal{B}$, it is possible to rewrite each function of the previous set in logical form as in (2.5):

$$\begin{cases} b_1 & = M_1 \ltimes a_1 \ltimes \cdot \ldots a_l \\ b_2 & = M_2 \ltimes a_1 \ltimes \cdot \ldots a_l \\ & \vdots \\ b_n & = M_n \ltimes a_1 \ltimes \cdot \ldots a_l \end{cases} \tag{2.8}$$

where a_j is the algebraic form of α_j, b_k is the algebraic form of β_k and M_i is the logical form of f_i.

Consider the vectors:

$$y = \ltimes_{k=1}^{n} b_k \quad x = \ltimes_{j=1}^{l} a_j \qquad y \in \Delta^{2^n}, x \in \Delta^{2^l} \tag{2.9}$$

The set of logical equations reported in (2.7) admits also another representation:

$$y = Lx \tag{2.10}$$

As for the M matrix, one can generate L starting from the matrix $\mathbf{0}^{2^n \times 2^l}$ and insert as ith column the output vector y associated with the input vector $\delta^i_{2^l}$.

The matrix L is unique for a system described as in (2.7). This is always related to the Lemma 2.1.1. In fact, suppose there are two different matrices L' and L'' that aim to represent the same system of equations. Calling c'_i and c''_i the i-th column of the L' and L'' matrices, one has $c'_i \neq c''_i$ by hypothesis, but this implies that for the same condition $\delta^i = \ltimes^l_{j=i} a_j$ the system reported in (2.8) has two different results:

$$\ltimes^n_{j=i} M_j \delta^i = \ltimes^n_{j=i} b'_j = c'_i \neq c''_i = \ltimes^n_{j=i} b''_j = \ltimes^n_{j=i} M_j \delta^i \tag{2.11}$$

and this is a contradiction.

We will use the boolean system to formalize any dynamical system.

2.3 Dynamical system

Consider two systems, an autonomous case, where the dynamics are generated only by the system state variables, and a second one where the dynamics are also described with a set of input variables. Let us introduce $x_1(k), x_2(k), \cdots, x_n(k)$, the n boolean variables that describe the system state at time k. The system state at time 0 takes the name of initial condition: $x_1(0), x_2(0), \cdots, x_n(0)$. The m variables $u_1(k), u_2(k), \cdots, u_m(k)$ describe the system input at the time k, provided that it shows a dependency on them. The set of equations:

$$\begin{cases} x_1(k+1) &= F_1(x_1(k), x_2(k) \cdots, x_n(k)) \\ x_2(k+1) &= F_2(x_1(k), x_2(k) \cdots, x_n(k)) \\ & \vdots \\ x_n(k+1) &= F_n(x_1(k), x_2(k) \cdots, x_n(k)) \end{cases} \tag{2.12}$$

takes the name of *boolean network* or *boolean system* (BN), and

$$\begin{cases} x_1(k+1) & = F_1(x_1(k), x_2(k), \cdots, x_n(k), u_1(k), u_2(k), \cdots, u_m(k)) \\ x_2(k+1) & = F_2(x_1(k), x_2(k), \cdots, x_n(k), u_1(k), u_2(k), \cdots, u_m(k)) \\ & \vdots \\ x_n(k+1) & = F_n(x_1(k), x_2(k), \cdots, x_n(k), u_1(k), u_2(k), \cdots, u_m(k)) \end{cases} \tag{2.13}$$

take the name of *boolean control network* (BCN). Both BNs and BCNs can be represented by a logical matrix L. This representation of the system is named of *algebraic form.* The boolean state variables and the boolean input variables can be compacted into a matrix (here an abuse of notation has been used, the state is no longer a boolean variable, but in logical form):

$$x(k) = \ltimes_{i=1}^{n} x_i(k) \qquad u(k) = \ltimes_{i=1}^{m} u_i(k) \tag{2.14}$$

and so a BN can be represented as

$$x(k+1) = Lx(k) \tag{2.15}$$

while a BCN is usually formalized as:

$$x(k+1) = L \ltimes u(k) \ltimes x(k) = Lu(k)x(k) \tag{2.16}$$

The decision to define the algebraic form of a BCN as in (2.16) is not trivial. With respect to the inverse position $Lx(k)u(k)$, this representation comes in handy to visualize each input configuration as a network update function selector. Consider the following examples:

Example 2.3.1. Given a system described by two boolean states α_1 and α_2, with the following boolean network:

$$\begin{cases} \alpha_1(k+1) = \alpha_1(k) \vee \alpha_2(k) \\ \alpha_2(k+1) = \alpha_1(k) \end{cases}$$

with $x_1(k)$ the vector representation of $\alpha_1(k)$, $x_2(k)$ the vector representation of $\alpha_2(k)$, and $x(k) = x_1(k) \ltimes x_2(k)$ it is possible to rewrite the system with the construction matrix as reported in (2.8):

$$\begin{cases} x_1(k+1) = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x(k) \\ x_2(k+1) = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} x(k) \end{cases}$$

It is also possible to compact the notation in a single logical matrix as in 2.15, the result is:

$$x(k+1) = Lx(k) = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x(k)$$

Once the M matrices are generated, the column of the L matrix are generated doing the STP of the relative M columns.

Example 2.3.2. It is given a system with two states x_1 and x_2 and an input variable u. The logical matrix $L \in \mathcal{B}^{8\times 4}$ is also given. The vector $u \ltimes x_1 \ltimes x_2$ is the following map:

$$\begin{aligned} &ux_1x_2 = \delta^1 && \bar{u}x_1x_2 = \delta^5 \\ &ux_1\bar{x}_2 = \delta^2 && \bar{u}x_1\bar{x}_2 = \delta^6 \\ &u\bar{x}_1x_2 = \delta^3 && \bar{u}\bar{x}_1x_2 = \delta^7 \\ &u\bar{x}_1\bar{x}_2 = \delta^4 && \bar{u}\bar{x}_1\bar{x}_2 = \delta^8 \end{aligned}$$

It is evident that δ^i, $i \in \{1,2,3,4\}$, and δ^j, $j \in \{5,6,7,8\}$, describe the same set of system states, with the difference posed in the input variable. This allows to divide L into two halves:

$$L = \left[\begin{array}{c|c} L_1 & L_2 \end{array} \right]$$

The input u works as a selector: if $u = \delta^1$ the state vector evolves according to the logical matrix L_1. Instead, if $u = \delta^2$, the system evolves according to L_2. In Figure 2.1 a representation of the example is given.

The Example 2.3.2 can be generalized to include m inputs: considering the algebraic form of the array of input $u(k) = \delta^i_{2^m}$, $i \in \{1, 2, \cdots, 2^m\}$, the operation $L \ltimes u(k)$ can be revisited as a BN selector, with the correspondence of i in $u = \delta^i$ and the i in the L

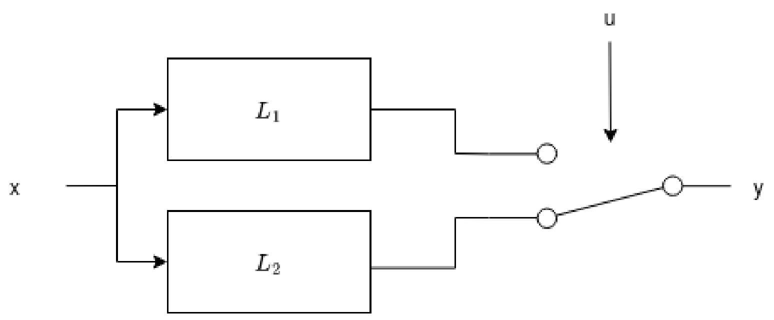


Figure 2.1: A representation of the input as a selector.

sub-matrix:

$$L = \left[\begin{array}{c|c|c|c} L_1 & L_2 & \cdots & L_{2^m} \end{array} \right] \tag{2.17}$$

leading the operation $Lu(k)x(k)$ to be the same as the multiplication

$$L_i x(k) \tag{2.18}$$

To summarize, the following theorem describes a general fact about a boolean system evolution:

Theorem 2.3.1. The evolution of a boolean network with $x(0)$ as initial condition until a time K is uniquely identified by the logical matrix L reported in (2.15)

Proof: Consider the reverse procedure evolution:

$$x(K) = Lx(K-1) = L^2x(K-2) = \cdots = L^K x(0)$$

and since the vector $x(k)$ uniquely identifies each state variable $x_i(k)$ it is possible to construct the evolution of the system starting form $x(0)$.

For a boolean control network in addition to a known L the input sequence $u(k)$ $k \in \{1, 2, \cdots, K\}$ is needed to reconstruct the evolution of the state:

$$\begin{aligned} x(K) &= Lu(K-1)x(K-1) = Lu(K-1)Lu(K-2)x(K-2) = \\ &= (\Pi_{i=1}^{K} Lu(K-i))x(0) \end{aligned}$$

Definition 2.3.1. The set of states $\{x(0), x(1), \cdots, x(K)\}$ that are obtained by Theorem

2.3.1 is called *trajectory* of the boolean network (or boolean control network)

Given a state $x(k)$, what can be said about its future evolution?The answer is to be sought in the study of L, in particular its columns. Starting from the autonomous case let us recall the definition of attractor:

Definition 2.3.2. Given a system in state space representation, where $\mathbf{x}$ is the state vector, $\mathbf{f}()$ is the set of functions that describe the dynamics of the system and $\mathcal{X}$ the phase space. An *attractor* A is a subset of $\mathcal{X}$ that has the following properties :

- It is invariant for $\mathbf{f}$, namely $\mathbf{f}(\mathbf{a}) \in A \ \forall \mathbf{a} \in A$,
- There exists a neighborhood of A, called the *basin of attraction* and denoted as $B(A)$, which consists of the subset of $\mathcal{X}$ where the trajectory of each point $b \in B$ converge to A
- There is no proper subset of A that maintains the previous properties.

Note also that the set of possible states is finite. That being stated, in a sufficient large time window it is impossible not to reach a repetition of a given state. By theorem 2.3.1 the system evolution is uniquely determined by L. Therefore, should one observe the repetition of a configuration, all the following ones would be observed again. Following the previous definition, it is possible to categorize each state x in three groups:

- If $x = Lx$, x is said to be a *fixed point*
- If $x = L^k x$ and the set $\{x, Lx, L^2x, L^{k-1}x\}$ does not contain repetition, $\{x, Lx, L^2x, L^{k-1}x\}$ x is said to be a *limit circle*
- The set $S_i = \{x | x(0) = x \to x(t) \in C_i, t \geq T_t\}$ is said to be the *domain of attraction* of the set C_i, where C_i is the set that describes an attractor, that can be either a fixed point or a limit circle.

Example 2.3.3. Reconsider the Example 2.3.1. From matrix L:

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

it is possible to extract the following information: the first and last column have 1's on the diagonal, which implies that $L\delta^1 = \delta^1$ and $L\delta^4 = \delta^4$ are fixed points. The layout of the other two columns imply that if $x(0) = \delta^3$ or $x(0) = \delta^2$, the following trajectory is the only one allowed:

$$\delta^2 = L\delta^3 \qquad \delta^1 = L\delta^2$$

In short, with different initial conditions, the possible trajectories are the following:

$$\begin{gathered} \delta^3 \rightarrow \delta^2 \rightarrow \{\delta^1\} \\ \{\delta^4\} \end{gathered}$$

where δ^1 and δ^4 are fixed points and $\{\delta^2, \delta^3\}$ is the domain of attraction of δ^1.

Definition 2.3.3. A state of a boolean network is said to be *stable* if it is a fixed point.

Definition 2.3.4. A boolean network is said to be *globally stable* if the set of its attractor is composed by a unique fixed point δ^i. If this is the case, it is possible to define a time T_i called *absorption time*. The absorption time is the minimum k value such that, for any initial condition $x(0)$, the attractor is reached: $T_i = \min_T$ s.t. $k > T_i \implies x(k) = L^k x(0) = \delta^i \, \forall x(0) \in \Delta$

If the boolean network is globally stable, the L^{T_i} matrix needs to be composed of the vector δ^i in each column:

$$L^{T_i} = \begin{bmatrix} \delta^i & \delta^i & \cdots & \delta^i \end{bmatrix} \tag{2.19}$$

If this does not apply, there exists an initial condition $x(0) = \delta^j$ that in T_i step does not reach the fixed point δ^i that does not reach the fixed point at step T_i. This observation emphasizes the importance of the power operation of the L matrix, in order to identify fixed points or limit circles.

Example 2.3.4. Reconsider the L matrix of the example 2.3.1, where the powers of L are:

$$L = \begin{bmatrix} \delta^1 & \delta^1 & \delta^2 & \delta^4 \end{bmatrix} \qquad L^2 = L^3 = \begin{bmatrix} \delta^1 & \delta^1 & \delta^1 & \delta^4 \end{bmatrix} \tag{2.20}$$

Since $L^2 = L^3$, it is impossible to have a different matrix if another step is taken in action. Looking at the columns of L^2, it turns out that δ^1 and δ^4 are actually fixed points.

It is possible to extract an additional behavior looking at the evolution of the columns in the sequence of L^k, $k \in \{1, 2, \cdot, K\}$. Indeed, considering the set of the evolution of the j columns:

$$\{\mathrm{Col}_j(L), \mathrm{Col}_j(L^2), \cdots, \mathrm{Col}_j(L^K)\} \tag{2.21}$$

and k, the exponent of matrix L as index of the element in the set, the previous result can be expressed as:

- If $Col_j(L^k) = Col_j(L^{k+1})$, then $Col_j(L^k)$ is a fixed point

- If $Col_j(L^k) = Col_j(L^{k+a})$, $a > 1$, and all the elements in the subset between the index k and $k + a$: $\{\mathrm{Col}_j(L^k), \mathrm{Col}_j(L^{k+1}), \cdots, \mathrm{Col}_j(L^{k+a-1})\}$ are distinct, then the previous subset is a limit circle.

The step one takes to generalize these concepts to a boolean control network is comparable to a classical study of a control affine system:

$$x(k+1) = f(x(k)) + g(x(k))u(k) \tag{2.22}$$

In this case there is no drift dynamics $f()$ and the $g(x(k))u(k)$ are rewritten as $L_i x(k)$ as reported in equation (2.18). It makes sense to convert the concepts of reachability, controllability and stabilazability in a boolean fashion. In this regard, it is better to introduce a new object:

$$L_{tot} = \sum_{i=1}^{2^m} L_i \tag{2.23}$$

Where L_i is the submatrix defined in (2.17). This object compresses all the possible future states in a single square matrix without specifying what input is needed to reach the specific output configuration.

Example 2.3.5. Given a boolean control network described with the matrix:

$$\left[\begin{array}{cccccccc|cccccccc}
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&1&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
1&1&0&0&0&1&0&0&1&1&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&1
\end{array}\right]$$

The L_{tot} turns out to be:

$$L_{tot} = \begin{bmatrix}
0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0\\
0&0&1&1&0&0&0&1\\
0&0&0&0&0&0&0&0\\
0&0&0&0&2&0&0&0\\
0&0&0&0&0&0&1&0\\
2&2&0&0&0&2&0&0\\
0&0&1&1&0&0&0&1
\end{bmatrix}$$

For example, consider a state $x = \delta^3$. From the third column it is possible to observe that the next state is or δ^3 either δ^8, even without knowing u.

Before moving on to the complete definition of the problem, let us introduce the following lemma, that explains the utility of L_{tot}:

Lemma 2.3.1. The number of input sequences that lead the system from the state x_a to x_b in k steps is:

$$l(k; x_a, x_b) = x_b L_{tot}^k x_a \tag{2.24}$$

Proof: call $u^1, u^2, \ldots, u^{l(k,x_a,x_b)}$ the sequences that carry the system from x_a to x_b. This implies that there exists a sequence $t(k, x_a, x_b) = 2^{mk} - l(k, x_a, x_b)$ that does not carry the system from x_a to x_b denoted by w:

$$1 = x_b^T L \ltimes u^i(k-1) \ltimes \cdots \ltimes L \ltimes u^i(0) x_a \quad i \in \{1, 2, \dots, l(k, x_a, x_b)\} \tag{2.25}$$

$$0 = x_b^T L \ltimes w^j(k-1) \ltimes \cdots \ltimes L \ltimes w^j(0) x_a \quad j \in \{1, 2, \dots, t(k, x_a, x_b) \tag{2.26}$$

Summing up all equation it results that:

$$l(k, x_a, x_b) = x_b^T L \ltimes 1_{2^m} \ltimes \cdots \ltimes L \ltimes 1_{2^m} x_a = x_b^T L_{tot}^k x_a \tag{2.27}$$

where 1_{2^m} is a column vector filled with ones and long 2^m and $L \ltimes 1_{2^m}$ is another way to define the L_{tot} matrix.

With this lemma it is easy to understand that a state x_b is reachable from x_a in k steps if and only if $l(k; x_a, x_b) \neq 0$, but if it is reachable in k steps there is no guarantee that is also possible to reach it in $k+1$ steps, this property is called stabilizability. Let's formalize all these properties:

Theorem 2.3.2. Considering a BCN with n states as reported in (2.13) and its algebraic form (2.16) from which the L_{tot} matrix is calculated. It turns out that:

a) A state $x_b = \delta^b$ is reachable from $x_a = \delta^a$ in k steps if and only if

$$l(k, x_a, x_b) > 0 \tag{2.28}$$

b) A state $x_b = \delta^b$ is reachable from $x_a = \delta^a$ if and only if

$$\sum_{k=1}^{2^n} l(k, x_a, x_b) > 0 \tag{2.29}$$

c) The BCN is said to be globally reachable from x_a if all the states are reachable from x_a

d) A BCN is said to be globally controllable to x_b if x_b is reachable from each other initial condition x_a

d) A BCN is said to be globally controllable if each state x_b can be reached from each state x_a, or alternatively

$$\sum_{k=1}^{2^n} \det\left(L_{tot}^k\right) > 0 \tag{2.30}$$

The proof of this theorem is based on the demonstration that there exists a path from a state x_a to a state x_b and this leads to $l(k; x_a, x_b) \neq 0$

To conclude the summary of the nomenclature of a boolean control network we report the concept of stabilizability. Starting from the definition of equilibrium point for a BCN, it is possible to simplify the concept of a stabilazability of a boolean control network to a configuration x_s as the sum of two properties.

Definition 2.3.5. A state x_e of a BCN with m inputs is said to be an *equilibrium point with constant input* (or simply *equilibrium point*) if there exists an input $\bar{u} \in \Delta^{2^m}$ such that $x_e = L\bar{u}x_e$

To be an equilibrium for a BCN, a state $x_e = \delta^i$ requires the existence of a possible stabilizing input: $x_e = L\bar{u}x_e$. Such condition is the same as requiring that the element in position (i, i) of L_{tot} is not zero. With this definition it is possible to state the probability of a stabilizability as:

Definition 2.3.6. A BCN is said to be *stabilizable* if exists a state x_s that is globally reachable form any other state and at the same time it is an equilibrium point.

Example 2.3.6. Consider the following matrix L, associated with a system with 3 system states and a single input:

$$L = \left[\begin{array}{cccccccc|cccccccc}
1&0&0&0&0&0&0&0&1&0&0&1&0&0&1&0\\
0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&1&0&0&1&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&1\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0
\end{array}\right]$$

The corresponding matrix L_{tot} is:

$$L_{tot} = \begin{bmatrix} 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

From L_{tot} it is possible to state that δ^1 and δ^2 are equilibrium points since the corresponding entries on the diagonal $[L_{tot}]_{1,1}$ and $[L_{tot}]_{2,2}$ are different from zero. They are good candidates to be points where the system can be stabilized. To this end we consider the powers of the L_{tot}:

$$L_{tot}^2 = \begin{bmatrix} 4 & 0 & 1 & 2 & 1 & 0 & 2 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 0 & 0 & 4 \\ 0 & 0 & 1 & 0 & 0 & 2 & 2 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \qquad L_{tot}^3 = \begin{bmatrix} 8 & 1 & 3 & 4 & 2 & 2 & 4 & 0 \\ 0 & 2 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 4 & 4 & 0 \\ 0 & 1 & 0 & 0 & 4 & 0 & 0 & 4 \\ 0 & 1 & 0 & 0 & 2 & 0 & 0 & 4 \\ 0 & 0 & 1 & 0 & 0 & 4 & 0 & 0 \end{bmatrix}$$

$$L_{tot}^4 = \begin{bmatrix} 16 & 4 & 6 & 9 & 6 & 4 & 8 & 4 \\ 0 & 3 & 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 8 & 0 & 0 & 8 \\ 0 & 1 & 4 & 1 & 0 & 8 & 4 & 0 \\ 0 & 1 & 2 & 1 & 0 & 4 & 4 & 0 \\ 0 & 1 & 0 & 0 & 4 & 0 & 0 & 4 \end{bmatrix}$$

looking at the first row of L_{tot}^4 it is possible to notice that each element is different from zero. As reported in Theorem 2.3.2 this means that δ^1 is reachable from every other initial condition. This fulfills the requirement of Definition 2.3.6. The second line has some zero entries. To understand if something more can be said, let us consider higher powers of the matrix:

$$L_{tot}^5 = \begin{bmatrix} 32 & 10 & 15 & 20 & 12 & 12 & 20 & 8 \\ 0 & 4 & 2 & 3 & 0 & 0 & 0 & 0 \\ 0 & 3 & 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 4 & 9 & 3 & 0 & 16 & 8 & 0 \\ 0 & 5 & 1 & 1 & 12 & 0 & 0 & 16 \\ 0 & 3 & 1 & 1 & 8 & 0 & 0 & 8 \\ 0 & 1 & 2 & 1 & 0 & 4 & 4 & 0 \end{bmatrix} \quad L_{tot}^6 = \begin{bmatrix} 64 & 25 & 32 & 42 & 32 & 24 & 40 & 24 \\ 0 & 6 & 3 & 4 & 0 & 0 & 0 & 0 \\ 0 & 4 & 2 & 3 & 0 & 0 & 0 & 0 \\ 0 & 3 & 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 13 & 3 & 4 & 24 & 0 & 0 & 32 \\ 0 & 6 & 13 & 5 & 0 & 24 & 16 & 0 \\ 0 & 4 & 9 & 3 & 0 & 16 & 8 & 0 \\ 0 & 3 & 1 & 1 & 8 & 0 & 0 & 8 \end{bmatrix}$$

$$L_{tot}^7 = \begin{bmatrix} 128 & 57 & 74 & 89 & 64 & 64 & 88 & 48 \\ 0 & 9 & 4 & 6 & 0 & 0 & 0 & 0 \\ 0 & 6 & 3 & 4 & 0 & 0 & 0 & 0 \\ 0 & 4 & 2 & 3 & 0 & 0 & 0 & 0 \\ 0 & 16 & 18 & 13 & 0 & 48 & 32 & 0 \\ 0 & 19 & 5 & 6 & 40 & 0 & 0 & 48 \\ 0 & 13 & 3 & 4 & 24 & 0 & 0 & 32 \\ 0 & 4 & 9 & 3 & 0 & 16 & 8 & 0 \end{bmatrix} \quad L_{tot}^8 = \begin{bmatrix} 256 & 131 & 153 & 185 & 152 & 128 & 176 & 128 \\ 0 & 13 & 6 & 9 & 0 & 0 & 0 & 0 \\ 0 & 9 & 4 & 6 & 0 & 0 & 0 & 0 \\ 0 & 6 & 3 & 4 & 0 & 0 & 0 & 0 \\ 0 & 44 & 13 & 16 & 80 & 0 & 0 & 96 \\ 0 & 24 & 46 & 19 & 0 & 80 & 48 & 0 \\ 0 & 16 & 28 & 13 & 0 & 48 & 32 & 0 \\ 0 & 13 & 3 & 4 & 24 & 0 & 0 & 32 \end{bmatrix}$$

L_{tot}^5 has the same zero and nonzero pattern as L_{tot}^7 and the same holds for L_{tot}^6 and L_{tot}^8. This implies that all future powers of that matrix will keep that same zero/nonzero pattern. The constant presence of zeros in the second row implies that the δ^2 configuration is not a stable state. Looking at the columns of the powers of L_{tot} it can be said that in five steps any other configuration can be reached from the states δ^2, δ^3 and δ^4. This implies that the BCN is globally reachable from those states.

2.4 Trajectory set

Consider a boolean network as in (2.15) in a time window $0, 1, \cdot, K$, and all the possible 2^n initial conditions $x(0) \in \delta^{2^n}$. It is possible to create a set that contains all the possible trajectories:

$$\mathcal{T}_f = \{(x(0), Lx(0), L^2x(0), \cdots, L^Kx(0)); \forall x(0)\Delta^{2^n}\} \tag{2.31}$$

It is also possible to introduce the same object for a boolean control network as in (2.16), the difference is that it is required to consider all the possible input sequences $u^1, u^2, \cdots, u^{2^{Km}}$:

$$\mathcal{T}_f = \{(x(0), Lu^i(1)x(0), \cdots, L^Ku^i(K)x(0)); \forall x(0)\Delta^{2^n}, \forall u^i \in \{u^1, u^2, \cdots, u^{2^{Km}}\}\} \tag{2.32}$$

This kind of object easily becomes huge and difficult to fully express as time and the dimension of the states increase. In practice the common use of this object is to verify if a model is coherent with the real world data without the introduction of a state estimator or an observer.

Example 2.4.1. Consider a general gene regulator network composed by n different genes with no inputs and a boolean model with its trajectory set $\mathcal{T}_f$. An experiment is conducted and the evolution of the genes state are measured at different time steps: $t_m = (x(0), x(1), \cdots, x(K))$. If $t_m \in \mathcal{T}_f$ the trajectory is said to be *compatible* with the model, if not or the measurements are affected with some kind of error, otherwise either the measurements are affected by some kind of error or the chosen model is wrong.

One may wonder whether the model is actually wrong. As reported by Kauffman, this approach of reducing a complex environment to a set of boolean actors is useful to describe the general evolution of the system, but we must keep in mind the fact that it is a huge simplification. Some modifications can be done to the general boolean model (2.7) in order to encapsulate more trajectory on the $\mathcal{T}_f$, in particular the next chapter will cover the following changes that can be made:

- Add noise to the state update function.
- Move from a deterministic boolean update function to a probabilistic one.
- Move from a synchronous update scheme to an asynchronous one.

Chapter 3

Beyond deterministic boolean networks

As reported at the end of the previous chapter, the use of a boolean model to describe a complex system as a gene regulator network can lead to some discrepancies between the predicted state evolution and the effective trajectory. Let us discuss the proposed changes to the general boolean model (2.13). The following discussion revolves around BCNs, but it holds also for a BN, provided one removes the dependence on u.

3.1 Stochastic model

The first change proposed is to move from a deterministic update function to a stochastic one. The main idea is to take the update function as described in (2.12) or in (2.13) and use the property of the XOR operator to add noise. The noise is described by n Bernoulli processes that generate at the time k a boolean value $w_i(k)$. The value is 1 with probability p_i and 0 with probability $1 - p_i$. By associating this process to the boolean update function, one obtains a stochastic model, in which the symbol $\oplus$ represents the

XOR operator:

$$\begin{cases} x_1(k+1) & = f_1(x_1(k), \cdots, x_n(k), u_1(k), \cdots, u_m(k)) \oplus w_1(k) \\ x_2(k+1) & = f_2(x_1(k), \cdots, x_n(k), u_1(k), \cdots, u_m(k)) \oplus w_2(k) \\ & \vdots \\ x_n(k+1) & = f_n(x_1(k), \cdots, x_n(k), u_1(k), \cdots, u_m(k)) \oplus w_n(k) \end{cases} \tag{3.1}$$

Recalling the truth table reported in Table 3.1, considering x as the state vector variable

x	w	$x \oplus w$
1	1	0
1	0	1
0	1	1
0	0	0

Table 3.1: XOR truth table

and w as the noise, it is possible to observe that the variable x affected by the noise w will change its status if and only if the noise value is 1. Indeed

$$x \oplus w \neq x \text{ if } w = 0 \tag{3.2}$$

$$x \oplus w = x \text{ if } w = 1 \tag{3.3}$$

This implies that the probability $1 - p_i$ can be seen as a "stability" measure of the state x_i: in fact, p_i is the probability that $w_i = 1$ and hence the state x_i changes to its complementary value $\bar{x}_i$.

A second observation is the fact that the probabilities p_i may be different from each other. This can be motivated by the information that the particular state variable encodes. Consider the following example for a better view of a gene regulator network environment:

Example 3.1.1. In [5] the following network is proposed to describe the epithelial to mesenchymal transition (EMT) of a cell. EMT is the process that makes an epithelial cell[1] lose its polarity and promotes the transformation into a mesenchymal stem cell.

[1]Roughly speaking a cell that composes the animal tissue

These cells are not “specialized”. Instead, they evolve into other kinds of cells, which can be bones, muscles, tissues, or cartilage cells. The EMT usually happens in three different scenarios: growth of the embryo, wound repair and cancer development. The authors discovered the existence of an intermediate phase, where the cell presents some properties of the epithelial cells and others of a mesenchymal cell. When the cell shows this characteristic it takes the name of hybrid cell.

To determine the state of the cell, it is necessary to look at four internal molecules and an external one that works as an input [5]. Since the target of this thesis is not biologists, details are omitted. However, such molecules are represented by the system state variables x_i. The network describing the process takes the form:

$$\begin{cases} x_1(k+1) = (x_1(k) \vee u(k)) \wedge (\neg x_2(k)) \\ x_2(k+1) = x_1(k) \wedge x_3(k) \\ x_3(k+1) = (x_1(k) \vee x_3(k)) \wedge (\neg x_4(k)) \\ x_4(k+1) = x_4 \wedge (\neg x_1(k) \vee x_3(k)) \end{cases} \tag{3.4}$$

According to [5] it is possible to subdivide the cells internal state as in Figure 3.1

It is evident that for any state of the epithelial cell, it is sufficient to change the input value to make the transition either hybrid or mesenchymal. If the input is fixed, the sets of states are divided into three domains of attraction each of them with a single fixed point. The authors have linked each domain to a specific type of cell. In this way, the model allows a change of type only by changing the input. How is it possible to describe the transition from epithelial to hybrid and at the end to mesenchymal state by changing the input only once? To be able to describe this behavior, one can consider each state as an unstable value, similar to (3.1). Consider for example an epithelial cell in the configuration $x = 0010$. At the time k u goes from 0 to 1, and the cell became a hybrid cell. At time $k + 1$, the cell reaches the equilibrium configuration $x = 1011$. In order to complete the transition to become mesenchymal, the cell needs a perturbation, for example, $x = 1011 \rightarrow x = 1011$. The noise can also be used to justify the fact that a mesenchymal cell that becomes a hybrid one.

Since this type of model is based on (2.13), one can ask himself which of the properties of the deterministic model can be converted into a stochastic version. The key to this

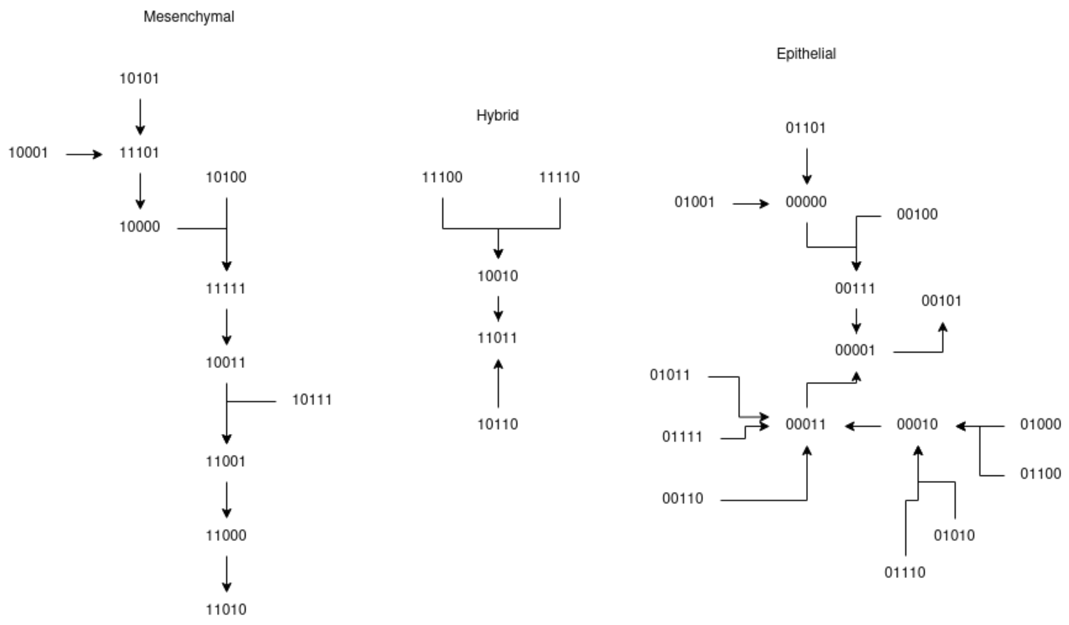


Figure 3.1: Evolution of the EMT network, where the numbers are the state of the cell in the following order: u, x_1, x_2, x_3, x_4.

is a change of viewpoint: instead of looking at a single state configuration, it is worth looking at a set of them. In this way even if the single configuration does change in time, it can be categorized as a single entity provided that the evolution of the system starting from an element of a set is fully contained in the same set. The second step is to assign a description to the sets, which will be related to the evolutions that they incorporate. Based on these descriptions it is worth evaluating the desired property with a measure. We will focus only on stability because controllability and reachability are not interesting for the scope of this thesis. In order to generate a stability index, one can make the following consideration: the study of a stochastic network needs to start from the deterministic part. An attractor is time invariant, and it makes no sense to study the stability of a state that changes even in the deterministic representation. Hence, the attractor and the basin of attraction will be the same for the deterministic model and for the one affected by the noise. As a consequence the interesting set to look at is the *1-degree neighbor* of an attractor. It is defined as the set of states whose vector representation differs by one bit (0/1) from the vector representation of the attractor state. Note that states belonging to the 1-degree neighbor of an attractor may be part of its domain of attraction or belong to the domain of attraction of a different attractor. Based on this, once all the attractors with the corresponding basin of attraction have been defined, the stability index of the attractor x_s is defined as:

$$S_{x_s} = \sum_i O_i^{x_s} - \sum_{j \neq x_s} O_{x_s}^j \tag{3.5}$$

where O_i^j is the ratio between the 1-degree neighbors of the attractor i in the basin of attractor j and the total 1-degree neighbors of i. In other words, the stability index provides information on what is the probability that a perturbation of a single bit results in a state that belongs to the domain of attraction of x_s. Indeed, $O_i^{x_s}$ gives the probability that a single bit perturbation of the attractor i results in a state that is attracted by x_s, while $O_{x_s}^j$ represents the probability that a single bit perturbation of the attractor X_s results in a state that is attracted by $j \neq x_s$.

The larger is the set S_{x_s} the more stable is the system. More specifically, a large value of S_{x_s} highlights the robustness of the system and its capability to endure small perturbations. This index cannot be regarded as a guarantee of stability, since it is possible to move to a state that can lead to a different domain of attraction.

For this type of model a problem with the trajectory set $\mathcal{T}_f$ arises. Due to the additive noise, the system evolves from one state x_a to any other state x_b. As a direct consequence, $\mathcal{T}_f$ is composed of all the possible state sequence combinations. Therefore, it is impossible to distinguish two models only by looking at the trajectories they generate.

3.2 Probabilistic Boolean Network

Instead of using the same update function at each time step, one can consider a model where there is more than a possible deterministic function that can be used to update the state vector. To do so let us introduce F_i, the set that contains all the possible update functions for the state x_i:

$$F_i = \{f_{1,i}(x_1, \cdots, x_n, u_1, \cdots, u_n), f_{2,i}(x_1, \cdots, x_n, u_1, \cdots, u_n), \ldots, f_{k_i,i}(x_1, \cdots, x_n, u_1, \cdots, u_n)\} \tag{3.6}$$

The single function $f_{a,i}$ is chosen with a probability $r_{a,i}$:

$$R_i = \{r_{1,i}, r_{2,i}, \ldots, r_{k_i,i}\} \tag{3.7}$$

Since it is a set of probability values it must be:

$$\sum_{a=1}^{k_i} r_{a,i} = 1 \qquad \forall i \tag{3.8}$$

From these update functions it is possible to generate $k_1 \cdot k_2 \cdot \cdots \cdot k_n$different BCNs. Each network is described by the set of indexes $\alpha_1, \alpha_2, \cdots, \alpha_n$. Each index is associated to a different update function in the set F_i. It is convenient to create a map from each n-tuple of indexes to a single natural number:

$$\begin{aligned}
(1, 1, \cdots, 1, 1) &\rightarrow 1 \\
(1, 1, \cdots, 1, 2) &\rightarrow 2 \\
&\vdots \\
(k_1, k_2, \ldots, k_n) &\rightarrow k_1 \cdot k_2 \cdot \cdots \cdot k_n
\end{aligned}$$

In this way it is possible to identify each BCN only through its map index l:

$$f_l^{PBN} : \begin{cases} x_1 &= f_{\alpha_1}(x_1, x_2, \cdots, x_n, u_1, u_2 \cdots, u_m) \\ x_2 &= f_{\alpha_2}(x_1, x_2, \cdots, x_n, u_1, u_2 \cdots, u_m) \\ \vdots \\ x_n &= f_{\alpha_n}(x_1, x_2, \cdots, x_n, u_1, u_2 \cdots, u_m) \end{cases} \tag{3.9}$$

The probability w_l associated to that network is

$$w_l = r_{\alpha_1,1} * r_{\alpha_2,2} * \cdots * r_{\alpha_n,n} \qquad \forall l \tag{3.10}$$

It is possible to introduce a discrete random variable W with the following probability mass function:

w	$P(W = w)$
1	w_1
2	w_2
$\vdots$	

and use it to update the system:

$$\begin{bmatrix} \mathbf{x}(k+1) \\ W(k+1) \end{bmatrix} = \begin{bmatrix} f_{W(k)}(\mathbf{x}(k), \mathbf{u}(k)) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ w \end{bmatrix} \tag{3.11}$$

where $f_{W(k)}(\mathbf{x}(k), \mathbf{u}(k))$ is the same as in (3.9).

Each network f_l can be rewritten in algebraic form by resorting to a logical matrix $L(l)$. As for L_{tot} it is possible to compress all the net information into a single matrix:

$$L = \sum_{i=1}^{l} w_i L(i) \tag{3.12}$$

With this notation the dynamical system can be written as:

$$x(k+1) = L(w)u(k)x(k) \tag{3.13}$$

and the expected vector state is expressed as:

$$E[x(k+1)] = Lu(k)E[x(k)] \tag{3.14}$$

These matrices can be used to check if the system has attractors, these are defined similarly to the ones for BNs and BCNs:

Definition 3.2.1. A state x_s of a PBN is said to be a *fixed point* if $x_s = L(w)x_s \ \forall w$

In practice, the point needs to be an equilibrium for each described network.

There are different possible definitions of equilibrium point for a PBCN, depending on the assumptions we introduce. If the network used to update the system at the time k is known before applying the input $u(k)$, it is possible to use the following definition:

Definition 3.2.2. A state x_e of a PBCN with m inputs is said to be an *equilibrium point* if for each w there exists an input u_w such that $x_e = L(w)u_w x_e \ \forall w$.

Otherwise, if the network is unknown until the transition at time k takes place, it is necessary to impose the same input for all possible $L(w)$:

Definition 3.2.3. A state x_e of a PBCN with m inputs is said to be an *equilibrium point* if there exist an input $\bar{u}$ such that $x_e = L(w)\bar{u}x_e \ \forall w$.

The decision about which definition should be used is taken at the time of the problem statement and it depends on the set-up: this problem can be ported as a request that $W(k)$ is accessible in (3.11). If this is the case it is possible to use 3.2.2. Otherwise, one needs to adopt 3.2.3.

Since the update matrix $L(w)$ is not known when the input is given to the system, it is requested that $\bar{u}$ is the same for each update matrix. Otherwise, even if the configuration is in equilibrium, it is impossible to determine the input sequence that leads to the expected behavior.

The concept of limit circle is more complex since it needs to be shared across all the networks. It is possible to revise this definition by imposing that this invariant set is shared across all the networks. To check these properties it is sufficient to look at the L matrix reported in (3.12). If a unitary entry appears on the diagonal (or in the diagonal of the submatrix corresponding to a specific input value) the evolution certainly is stationary. If a state is not an equilibrium point, it is interesting to investigate what are the configurations that are reachable from it. Recall that a state is said to be *reachable* from another state if the former belongs to one of the trajectories starting from the latter. Since the update function is not constant, it is necessary to keep track of it when the reachability of a certain state is discussed:

Definition 3.2.4. A state x_b is said to be *definitely reachable* from x_a in k steps if it is certain that in k steps the system state will move from x_a to x_b. Otherwise, it is *reachable* with probability p in k steps if the probability to move from x_a to x_b in k is p.

Starting from the PBN matrix L and comparing it with the deterministic one, we notice that the generic jth column of L is no longer a canonical vector, instead its ith entry will represent the probability to "jump" to the state δ^i starting from the selected initial condition δ^j. Knowing this, it is possible to employ the techniques devised for classical boolean control networks. The main problem is the fact that Lemma 2.3.1 does not apply anymore. One possibility is to normalize each column and verify if $x_b^T L_{tot}^k x_a$ is different from zero. Should this event occur, it is possible to reach x_b from x_a in k steps.

In the following example, we discuss how the matrix L defined in (3.12) allows deriving some conclusions on the trajectory behavior of the boolean control network.

Example 3.2.1. Consider the PBCN with two states x_1 and x_2 and the following update equations together with their probabilities:

$$\begin{cases} x_1(k+1) = \begin{cases} u_1(k) \vee (x_1(k) \vee x_2(k)), & p = 0.3 \\ u_1(k) \vee x_1(k), & p = 0.7 \end{cases} \\ x_2(k+1) = \begin{cases} x_1(k), & p = 0.2 \\ x_1(k) \wedge x_2(k), & p = 0.8 \end{cases} \end{cases}$$

There are only four possible configurations that correspond to the following matrices:

$$\begin{aligned} L(1) &= \delta[1,1,2,2,1,1,2,4], \ p = 0.06 \\ L(2) &= \delta[1,2,2,2,1,2,2,4], \ p = 0.24 \\ L(3) &= \delta[1,1,2,2,1,1,4,4], \ p = 0.14 \\ L(4) &= \delta[1,2,2,2,1,2,4,4], \ p = 0.56 \end{aligned}$$

The matrix L turns out to be:

$$L = \sum_{i=1}^{4} L(i) = \begin{bmatrix} 1 & 0.2 & 0 & 0 & 1 & 0.2 & 0 & 0 \\ 0 & 0.8 & 1 & 1 & 0 & 0.8 & 0.3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 1 \end{bmatrix}$$

From which it is possible to calculate compute the L_{tot} matrix:

$$L_{tot} = \begin{bmatrix} 2 & 0.4 & 0 & 0 \\ 0 & 1.6 & 1.3 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0.7 & 1 \end{bmatrix}$$

From L and L_{tot} it is possible to extract the following dynamics: δ^1 and δ^4 are equilibrium points, while δ^3 is in the basin of attraction of both equilibrium points. Instead, δ^2 may either be stable, thus preserving its status, or switch to δ^1. Consider, without loss of generality, a constant input $u = \delta^1$. At each time step the probability to preserve a given configuration is: $\Pi_{i=1}^{k} 0.8$. In other words, one may legitimately expect δ^2 to converge to δ^1 provided that the time frame is large enough.

3.3 Asynchronous Step Update Scheme

Up to now, we have described systems whose update functions are synchronous. In this section, by referring to [6] and [7] we will discuss some problems of this approach. When dealing with a general GRN, the variables of the BCN represent different biological objects such as DNA, RNA and proteins. These objects have different time scales, which are not taken into consideration when the model is generated. A second problem is determined by the simplification of the real world provided by boolean models. In the real world, an event can not be triggered by the presence of a single molecule, but a threshold needs to be reached. A synchronous boolean scheme is not always the best option for this type of behavior. An argument against the synchronous update is how a biological function works. Kalman reported that it is the changes in the state of a variable that induces the activation of the update function. For this reason, a "cascade" updating scheme is introduced, where the state variable x_i updates through its update function f_i if and only the latter depends on at least a variable that has changed at the previous time step. There are other possible asynchronous updating schemes. Each one has particular relevance in a specific scientific environment. It turns out that a cascade update is a suitable type of updating scheme when we deal with the update function of a genetic network. To formalize this approach let us introduce a function that, given two state vectors x_1 and x_2, returns the state variable that has changed. With this function it is possible to express the asynchronous update in compact form as:

$$\begin{bmatrix} \mathbf{x}(k+1) \\ \mathbf{c}(k+1) \end{bmatrix} = \begin{bmatrix} f_c(\mathbf{x}(k), \mathbf{u}(k), \mathbf{c}(k)) \\ f_d(\mathbf{x}(k), \mathbf{u}(k), \mathbf{c}(k)) \end{bmatrix} \tag{3.15}$$

The vector $\mathbf{c}(k)$ contains the indexes of the variables that have changed at the previous update. For each boolean function (2.15), f_c checks whether f_i depends on a variable in $\mathbf{c}$. If not, then it replaces such variable update function with $x_i(k-1)$.

This update scheme can be also used for a probabilistic network as (3.11):

$$\begin{bmatrix} \mathbf{x}(k+1) \\ W(k+1) \\ \mathbf{c}(k+1) \end{bmatrix} = \begin{bmatrix} f_c(\mathbf{x}(k), W(k), \mathbf{c}(k)) \\ 0 \\ f_d(\mathbf{x}(k), W(k), \mathbf{c}(k)) \end{bmatrix} + \begin{bmatrix} 0 \\ w \\ 0 \end{bmatrix} \tag{3.16}$$

Where the update functions are based on the same philosophy as for (3.11). If c constantly contains all possible values, the asynchronous cascade updating model collapses to the synchronous one. As a consequence, as reported in [6], the synchronous updating mechanism is a specific case of the cascade update model. The biological reason for the introduction of this approach is that in a boolean network the concept of time does not make any sense, and usually, it is the change of status of a node that triggers the update of all the other nodes whose evolution depends on it. The initial conditions need to account also for this: if $c(0)$ is void the system is in equilibrium.

As in the previous section, we will study what is the expected behavior of an asynchronous model. The first step is the concept of stability of a BN. In the synchronous deterministic model, to determine if a state δ^i is an equilibrium it is only necessary to look at the algebraic matrix L and see if its (i,i) entry is unitary. Since the update function depends on the variable that was previously updated, this simplicity cannot be transferred directly into the cascade model. The same state can have multiple possible evolutions depending on the c vector that is associated with it. If c is void the state will not change in time. This is compatible with the concept of attractor and leads to the conclusion that in addition to checking the update functions, one has to look at c to determine when a configuration is in equilibrium or not. This also implies that each configuration can be an equilibrium, depending on how the system reached it. The focus needs to be posed on how the system will change the update function. This task is very operations-specific, there is no general result, and most papers come to the aforementioned conclusion using numerical analysis. Here is presented an example where a GNR has been studied to better visualize how difficult is to study an asynchronous model in comparison to a synchronous one.

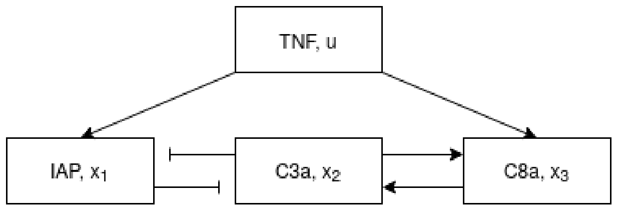


Figure 3.2: A simple GNR.

Example 3.3.1. Consider the GNR reported in Figure 3.2. This network is a simplification of the life circle of a cell as reported in [4]. TNF is the acronym for Tumor Necrosis

Factor, a protein that the body uses to notify the cell of a request to kill itself. In this model, there is an external input u. IAP is a set of proteins that act as inhibitors of apoptosis[2]. C3a and C8a are two cascades of internal cell processes[3] that interacts with TNF and IAP. Starting from the figure above, the presence of IAP will be modeled by the boolean variable x_1, the state of the cascade C3a by the boolean variable x_2 and C8a by x_3. The corresponding model is the following:

$$\begin{cases} x_1(k+1) = (\neg x_2(k)) \wedge u(k) \\ x_2(k+1) = x_3(k) \wedge (\neg x_1(k)) \\ x_3(k+1) = x_2(k) \vee u(k) \end{cases}$$

It turns out that there are two special configurations: $x_1 = 0 \wedge x_2 = 1$ that correspond to the dead cell, and all the other configurations correspond to a living cell. The question we want to answer is the following: is it always possible to determine an input sequence u that makes the cell move from an alive configuration to a dead one? The expected answer is negative. Pathologies like cancer inhibit the apoptotic process. The two matrices L and L_{tot} turn out to be:

$$L = \left[\begin{array}{cccccccc|cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1
\end{array}\right]$$

[2]The apoptosis is the process by which the cell kills itself.

[3]"A cascade of internal cell process" is a sequence of interactions between different components or molecules inside the cell. This is usually an independent process that can be considered as a single entity.

$$L_{tot} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 2 & 2 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The initial request is that δ^5 and δ^6 are the only equilibrium points when $u = 1$, while δ^3,δ^4,δ^7 and δ^8 are transient points. In this way, it is guaranteed that each configuration leads to the desired one when the input takes that specific value.

Looking at L_{tot} it is immediate to notice that $[L_{tot}]_{5,5} = 2$. This implies that δ^5 is an equilibrium point for every $u \in \{0, 1\}$. δ^6 is not an equilibrium point, but δ^3 and δ^8 are. Since it is impossible to determine the input configuration that leads the states δ^3 and δ^8 to be equilibrium points?, it is worth to look or at L or at the trajectories. From L we deduce the possible one-step evolutions:

$$\begin{aligned}
\delta^1 &\to \delta^7 && \forall u \\
\delta^2 &\to \delta^7 && \forall u \\
\delta^3 &\to \delta^3 && \text{if } u = \delta^1 \\
\delta^3 &\to \delta^8 && \text{if } u = \delta^2 \\
\delta^4 &\to \delta^3 && \text{if } u = \delta^1 \\
\delta^4 &\to \delta^8 && \text{if } u = \delta^2 \\
\delta^5 &\to \delta^5 && \forall u \\
\delta^6 &\to \delta^7 && \forall u \\
\delta^7 &\to \delta^1 && \text{if } u = \delta^1 \\
\delta^7 &\to \delta^6 && \text{if } u = \delta^2 \\
\delta^8 &\to \delta^3 && \text{if } u = \delta^1 \\
\delta^8 &\to \delta^8 && \text{if } u = \delta^2
\end{aligned}$$

From these evolutions, it is possible to draw the following conclusions. There exists a set of states that act as traps, i.e., once the system status enters the set it is impossible to find an input sequence to exit the trap set. A deeper analysis of the evolution configuration leads to identify as possible the following cases:

- The trap set $\{\delta^3, \delta^8\}$, that can be accessed by the external state δ^4 independently of the input configuration.
- The trap set $\{\delta^1, \delta^6, \delta^7\}$, that can be accessed by the external state δ^2 independently of the input configuration.
- The state $\{\delta^5\}$, which is an equilibrium for all the inputs and is isolated.

This analysis emphasizes that it is impossible to move from one set to another. The initial request was to check if the system is stabilizable at δ^5 or δ^6. Given that they are not present in the first row the answer to our request is negative. If we switch to an asynchronous update scheme the possible trajectories are reported in Figure 3.3 [4]

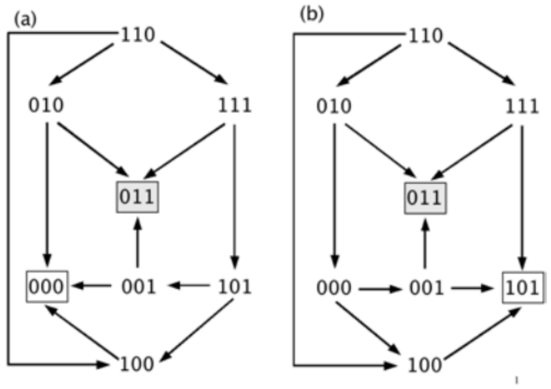


Figure 3.3: Asynchronous transition graph for the system. The nodes are the states of the system's boolean variables. (a) shows the updates with $u = 0$ and (b) with $u = 1$. The common state 011 represents the dead cell the corrispettive of δ^5. 000 (δ^8) and 101 (δ^3) represent the cell that will survive.

Here the desired states δ^5 can be reached from δ^6. So, a switching input fulfills the request, as shows in the image switching between a and b. The different updates determine different scenarios, and it is necessary to select the one that is relevant to the real data.

The lack of a rigorous procedure to determine what is the behavior of an asynchronous system leads to a more complex analysis. In the next chapter we will focus on a general way to detect if there is the system is faulty, both in case the system is a simple BCN and in case it is a more complex network of the type discussed in this chapter.

[4]The figure is taken from [4].

Chapter 4

Models for fault detection

Up to now, the reference system has been assumed to be perfectly described by its mathematical representation. The question addressed in this chapter is the following: What happens if an error occurs? The error can regard the state vector, due to a wrong initial condition or a wrong state estimation from the real world data. Otherwise, there is the possibility that the system changes its internal behavior and the mathematical representation is no longer accurate. The latter problem can happen if a component breaks inside the system, which could happen for reasons internal or external to the system. The goal of this chapter is to verify if some kind of fault has happened to the system and if it is possible to locate it.

According to [8], in a gene regulatory network represented by a BCN, the fault can be viewed as a corruption of its boolean operators. This can happen in two ways:

- "Stuck-at" fault: this fault happens when the operator has the same output for each input configuration. In particular, it is referred to as 'stuck-at-1' and 'stuck-at-0' to better explain what type of fault occurred.
- "Bridging" fault: this type of fault happens when the operator accepts as input a different component from the one that was initially configured. This can happen if the process can be also activated by a different molecule that has a small error in its production somewhere else in the system.

A pathology, such as cancer, can be a combination of multiple faults in the internal cell network, that leads to an evolution that differs from the normal one. The system states remain the same, but they are linked by a different set of update functions. We denote by L_{fault} the update matrix of the faulty system, i.e., the update function of the pathology, by $\{u^1, u^2, \cdots, u^{2^{Km}}\}$ the set of all possible inputs in the time window $\{0, 1, \cdots, K\}$, and by Δ^{2n} the set of all the possible initial conditions. If it is possible to directly observe the system state (this hypothesis is not true in most cases, and we will relax it a bit later), it is possible to define the set trajectories of the faulty system in the time window $\{0, 1, \cdots, K\}$:

$$\mathcal{T}_{fault} = \{(x(0), L_{fault}u^i(1)x(0), \cdots, L^K_{fault}u^i(K)x(0)); \forall x(0) \in \Delta^{2^n}, \forall u^i \in \{u^1, u^2, \cdots, u^{2^{Km}}\}\} \tag{4.1}$$

The previous set, by construction, describes only the dynamics for $t = 0, 1, \ldots, K+1$ of a system that is faulty already at $t = 0$. This is a strong hypothesis because in general a fault can happen inside the time window, namely the system behaves normally until the j^{th} step, when the fault occurs, and the faulty update function is used to generate the rest of the trajectory. If the exact moment of the fault is known the resulting set of trajectories, corresponding to all possible initial conditions is:

$$\begin{aligned}\mathcal{T}^j_{f,fault} = \{(x(0), Lu^i(1)x(0), \cdots, x(0), L^{j-1}u^i(j-1)x(0), x(0), L_{fault}L^{j-1}u^i(j)x(0),\\ \cdots, L^{K-j}_{fault}L^{j-1}u^i(K)x(0)); \forall x(0) \in \Delta^{2^n}, \forall u^i \in \{u^1, u^2, \cdots, u^{2^{Km}}\}\}\end{aligned} \tag{4.2}$$

There are now two strong hypotheses that have been introduced to derive this set and need to be relaxed a bit: the direct measure of the state vector and the knowledge of the exact time when the fault occurs. To measure the state of the system a set of experiments needs to be done. Each experiment can be modeled as a function $h(x_1, \cdots, x_n)$ called *output function.*

4.1 Output of the system

Consider a boolean system described as in (2.12) or in (2.13). Up to now, we have assumed that all system variables are accessible. This is not the general case. In classical system theory, this has been fixed by assuming that at least some output measurements, depending on the state variables, are available. In the boolean context this is equivalent to introduce a set of p boolean function $F : \mathcal{B}^{n+m} \to \mathcal{B}$:

$$\begin{cases} y_1(k) & = h_1(x_1(k), \cdots x_n(k), u_1(k), \cdots u_m(k)) \\ y_2(k) & = h_2(x_1(k), \cdots x_n(k), u_1(k), \cdots u_m(k)) \\ & \vdots \\ y_p(k) & = h_p(x_1(k), \cdots x_n(k), u_1(k), \cdots u_m(k)) \end{cases} \tag{4.3}$$

Being a set of boolean equations, it admits a logical representation:

$$y(k) = Hu(k)x(k) \tag{4.4}$$

where $y(k) = \ltimes_{i=1}^{p} y_i$.

As in the classical nomenclature, a boolean system is said to be *observable* if it is always possible to the initial condition from the input sequence and the corresponding output measurements in a finite time window. To do so we calculate the state with:

$$x(k) = (Hu(k))^T y(k) \tag{4.5}$$

If $x(k)$ is a canonical vector, this means that there exists a bijective correspondence between this state of the system and the corresponding output.

Otherwise, the state of the system is a boolean vector that can thought of the sum of all the vectors δ^i that generate the given output. In this case, the analysis needs to be performed by looking also at the previous time steps and checking all the possible scenarios, as in the classical approach. As a result, we can modify a bit the concept of trajectory set. We introduce the *output trajectory set* :

$$\mathcal{T}_{o,f} = \{Hx(0), HLx(0), HL^2x(0), \ldots, \forall x(0) \in \Delta^{2n}\} \tag{4.6}$$

Considering the system output and therefore the output trajectories, relaxes the first hypothesis, as it is no longer necessary to know the complete state of the system. To do this, it is sufficient to consider $y(k) = Hx(k)$ instead of $x(k)$ without further modifying the discussion.

4.2 Types of fault

To relax the hypothesis of knowing when the fault occurs we will create a new set that is the union of the trajectories that correspond to the different locations of the fault in time:

$$\mathcal{T}_{f,fault} = \bigcup_{j=1}^{K} \mathcal{T}^{j}_{o,f,fault} \tag{4.7}$$

Note: in this definition, we group the output trajectories, but we drop the o-subscript since in real scenarios direct access to the system state is not guaranteed.

To recap, so far we have described how a breakdown of the model affects its trajectories. The new set of trajectories is strictly dependent on the fault time, in the sense that two different fault times correspond to two different sets of trajectories. By analyzing these sets, we can classify the fault. In particular, we are interested in knowing if it is possible to identify the fault time by looking at the trajectory (or rather the output trajectory) of the system[1].

Definition 4.2.1. A fault is said to be:

- *Detectable* if the intersection between $\mathcal{T}_f$ and $\mathcal{T}_{f,fault}$ is void.
- *Not detectable* if $\mathcal{T}_{f,fault} \subset \mathcal{T}_f$
- *Possibly detectable* if there exists some trajectories in $\mathcal{T}_{f,fault}$ that are not shared with $\mathcal{T}_f$, namely $\mathcal{T}_{f,fault} \cap \mathcal{T}_f \neq \emptyset$ but $\mathcal{T}_{f,fault} \not\subset \mathcal{T}_f$.

An absolutely observable fault is the best situation that can occur since it is always possible to understand if the system is faulty or not. Usually, the more complex the model becomes, for example as reported in Chapter 3, the more unlikely is to fall in this case. It is also important to remark on the assumption that, until now, the system may be affected only by a single kind of fault. If this is not the case, it is necessary to build a set $\mathcal{T}_{f,fault_i}$ for each known possible fault, and check if there exist some detectable trajectories. If there is no knowledge of the fault type, it is only possible to detect if the evolution does not belong to the set of expected ones $\mathcal{T}_f$. Checking the presence of a trajectory inside the previous set is elegant, but it requires the knowledge of the entire trajectory

[1]Keeping into account that the trajectories also depend on the type of model, as discussed in Chapter 3.

set. This is costly for complex models. Another issue is the size of the time windows of the trajectories. The researcher needs to figure out how big the experiment needs to be in order to detect the possible fault, taking into account the relative cost.

The first approach to fault detection is to use a boolean model to build a state estimator as in the classical system theory literature, meaning that an estimator is used to obtain an estimate, $\bar{x}(k)$, of the state at the time k. Then it is possible to assume as error measure the Hamming distance between $x(k)$ and $\bar{x}(k)$. The Hamming distance of two Boolean vectors is the number of positions in which the two vectors differ. As in [6] and [9], this distance should remain contained in a specific interval of numbers if the system is fault free (or the fault is not detectable). This interval is specific for each model, but the procedure for obtaining it is not trivial. If a fracture occurs, this measurement is out of its range. In practice, it is necessary to study the evolution of the error

$$e(k) = Hamm(x(k), \bar{x}(k)). \tag{4.8}$$

In the rest of the thesis we will not address this case, but if the state variables are not accessible and hence only observable from an output equation $y(k) = h(x(k))$, the distance can be assumed:

$$e(k) = Hamm(y(k), \bar{y}(k)) = Hamm(h(x(k)), h(\bar{x})(k)) \tag{4.9}$$

Now it is possible to proceed in two ways. The first method is to set a threshold: when $e(\cdot)$ exceeds this threshold, an error in the system will be notified. Otherwise, a more complex study can be done by treating $e(\cdot)$ as a stochastic process. If $e(\cdot)$ does not match the normal expectation, the system needs to be treated as a faulty one. From now on we will focus on this second approach.

4.3 State estimator

To estimate the current state of a boolean network it is possible to use a classical open loop observer. This is an additional system that shares the same boolean update function of the desired system. This state estimator needs to be initialized with the same initial condition and fed with the same input that goes to the real system. The knowledge of the initial condition is a strong assumption, because if we are able to know them we should be able also to know the state of the system at each moment. We will discuss how integrate the output of the system to remove this hypothesis. Figure 4.1 is a graphical representation of this approach.

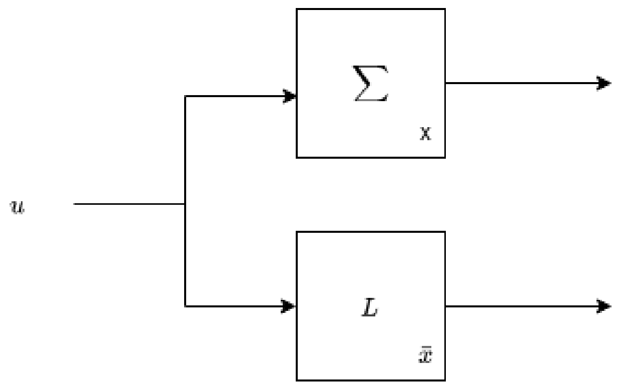


Figure 4.1: Classical state estimator scheme in open loop

This estimator is classically called *open loop full observer* and has some drawbacks. Once initialized, there is no guarantee to keep the estimated state aligned with the real one. The reduction of the gap between the estimated and the real state is addressed with the introduction of a feedback; this will addressed in the next section. This approach can be transferred to BNs and BCNs, but some problems emerge. They depend on the characteristics of the update function, in particular, if the update function is no more deterministic the construction of a proper feedback loop is not trivial. Consider a PBN as in (3.9), recalling that the state update can be represented as:

$$x(k+1) = L(\omega)u(k)x(k) \tag{4.10}$$

Where $L(\omega)$ is the current update matrix. Since it is not possible to know in advance what is this matrix before the update, it is possible to study the expected evolution of the state using L as reported in [9]:

$$\hat{x}_e(k+1) = E[L(\omega)]u(k)\hat{x}_e(k) \tag{4.11}$$

The expected $E[L(\omega)]$ is the same reported in (3.13). As previously discussed, $\hat{x}_e(k)$ is not a more canonical. Instead, each entry i will represent the probability that the system is in the δ^i configuration. It is worth noticing that the evolution reported in (4.11) keeps track of the previous probability when the new state is calculated. The absence of feedback leads each i-probability to depend on all the previously estimated ones.

An asynchronous system requires a more complex approach. It is necessary to keep track of the state's history and change the update function accordingly. If the open loop observer is combined with a stochastic update function, it is necessary to treat each possibility separately. Such an approach requires more attention and becomes confusing after a few steps. In these cases, a mere observer is not recommended. As an alternative, an estimator based on the Kalman filter is very promising as reported in [10], [11], and [12]. But first, let's see how this observer is useful to identify the faults.

4.4 Fault detection observer

Consider the output function of a boolean dynamical system as reported in (4.4). An observer can be built as:

$$\hat{x}_o = (Hu(k))^T y(k) \tag{4.12}$$

Where the vector $\hat{x}_o$ is a probability vector as $\hat{x}_e(k)$. The difference is that instead of being evaluated based on the evolution function, these probabilities are evaluated from the observation function. It is also important to report the fact that these probabilities depend on the current system state. The estimated state at the time k requires the knowledge of the system input and the output at the time k. $\hat{x}_o$ can be combined with the estimated state gained in (4.11) to obtain a better state estimator:

$$\hat{x}(k)' = \hat{x}_o(k) \circ \hat{x}_e(k) \tag{4.13}$$

where $\circ$ is the element wise multiplication. This operation combines each specific state probability, obtained from the expected evolution, with the probability to be in that state given the output. Consequently, $\hat{x}(k)'$ is a state vector whose i^{th} entry is $p_e^i(k)$, the probability of the system being in the δ^i configuration after the last update, multiplied with $p_o^i(k)$, the probability of the system to be in the δ^i state given the output of the system. The entries of $\hat{x}(k)'$ are not ensured to sum to one anymore. The vector contains all possible configurations estimated in $\hat{x}_e(k)$ that are compatible with the output. This sum is represented with the symbol α as reported in [9]:

$$\alpha(k) = \sum_{i=1}^{n} \hat{x}'_i(k). \tag{4.14}$$

If we assume that there is no fault in the system, α can be used to normalize the state vector, to obtain a distribution that sums to one:

$$\hat{x}(k) = \frac{1}{\alpha(k)} \hat{x}'(k) \tag{4.15}$$

Given this description of α, it is interesting to note that $1-\alpha$ corresponds to evolutions that are not compatible with the model. This leads to the next construction: assume a fault free system. At time k the state vector is $x(k) = \delta^i$. The corresponding estimated state $\hat{x}_e(k)$ has in the i^{th} position a positive number that corresponds to the probability to have reached that configuration given the system model. The entry is zero if that configuration is not compatible, given the update function and the previous state vector $\hat{x}(k)$.

At the same time, the output $\delta_v^j = Hu(k)\delta^i$ is compatible with the state δ^i as far as $[\hat{x}_o(k)]_i > 0$.

In this way, α is ensured to be the sum of zeros and at least a positive number. Otherwise, if the model has a fault the update $\hat{x}(k) = Lu(k-1)\hat{x}(k-1)$ could have a zero in the i^{th} position of $\hat{x}(k)$. This can be used to determine the presence of an observable fault.

Since $\hat{x}_e$ is the vector with the probabilities to reach each particular configuration, the element-wise multiplication with the observed state acts as a "selector" leading $\hat{x}'$ to contain only the accepted states.

Example 4.4.1. Consider a network and its fault detector observer. This system has some internal states that lead to the same output. One of these states cannot be reached by the faulty free system. As a consequence it is possible to be in the case where $HLx(0) = HL_{faulty}x(0)$. In this case, α remains greater than zero, since the output is compatible with the faulty free one. It remains so until $HL^k x(0) \neq HL^k_{faulty}x(0)$. When there is the case of inequality, α drops to zero and it is certainly possible to say that the system had a fault.

Looking only if $\alpha(k)$ reaches zero, limits this approach as it only occurs when the observed trajectory is not included in $\mathcal{T}_f$. In the other cases, namely when the intersection between $\mathcal{T}_f$ and $\mathcal{T}_{f,fault}$ is not empty, α remains greater than zero. It is possible to mitigate this problem at the cost of the possibility of some false alarms. To do so we require knowledge of the faulty system model. The basic idea is simple, we assume that the system is affected by a fault at time k and evolves according with the faulty model L_{faulty}. We build an estimator on this evolution, and we evaluate through alpha the probability that the current step is compatible with the given faulty model.

$$\hat{x}_{f,e}(k+1) = L_{fautl}u(k)\hat{x}(k) \tag{4.16}$$

$$\hat{x}'_f(k) = \hat{x}_o(k) \circ \hat{x}_{f,e}(k) \tag{4.17}$$

$$\alpha_{faulty}(k) = \sum_{i=1}^{n} \hat{x}'_f(k) \tag{4.18}$$

$$\hat{x}_f(k) = \frac{1}{\alpha(k)}\hat{x}'_f(k) \tag{4.19}$$

In this way, α_{faulty} will be the sum of states compatible with the given output weighted by the probabilities determined by the faulty model. Such an estimator needs to work in parallel to the one designed for normal behavior.

We want to point out that (4.11) shares the same state vector $\hat{x}(k)$ as in (4.19) before performing the update. Given that the two estimators need to share the same initial state $\hat{x}(k)$, we need to choose between (4.15) and (4.19) This is equivalent to asking whether the model is faulty or not. In order to answer the usual question of weather or not the model is faulty, we need to check the following inequality:

$$\beta\alpha(k) < \alpha_{faulty}(k) \tag{4.20}$$

Whether the case is correct the system functions normally. Otherwise, the system needs to be marked as faulty. By making use of β one can tune the probability of false alarms against detection rate.

This approach can also be modified to integrate more than a single faulty model, checking which one is the most probable. To do so, we reproduce an estimator for each known fault, and calculate its $\alpha_{faulty,j}$ measure. The second step remains to look at which one is the most probable.

$$i(k) = \arg\max(\beta\alpha(k), \alpha_{faulty,1}(k), \cdots, \alpha_{faulty,n}(k)) \tag{4.21}$$

This approach requires knowledge of how the fault affects the system. Moreover, such an approach can be suitable in a biological environment, if the request is not to understand how the system broke down but to check if it is affected by a specific pathology.

4.5 Boolean Kalman filter

In the last part of this chapter, we present the Boolean Kalman Filter as an alternative to the state estimator. The base idea is to create 2^n cases, where n is the number of variables in the state vector. Each case will be a different simulation of the system evolution. Associate at each case a possible initial condition of the system[2]. After this initialization phase, for each time step in the time window we repeat the following: Update the case with the system update function f and use the output function to estimate the corresponding probability. Use this probability to make a weighted sum of the cases. This would be the estimated current state. The estimated state will not interfere with the various simulations, which will remain independent processes. Using all the possible states at the same time is the key to the correctness of this algorithm since it ensures keeping traces of all possible outcomes. However, this solution is quite demanding and becomes impossible to use with large systems. This can be mitigated by adding a particle filter. The probability of each state is used to generate a probability mass function within the scope of sampling the possible states. Only this selection will be used to estimate the current state and the next initial cases. In the algorithm 1 it is reported the pseudocode of the particle filter as discussed in [11].

This is a Monte Carlo approach, where the state is estimated with a span of simulations. The more simulations are taken into account, the more precise the estimation becomes. It can be seen as a rough calculation of the possible evolution, and the particle filter allows for a reduction in the amounts of particles requested. If the initial condition of the system is known, it is possible to encapsulate this information in the initialization phase.

[2]Since the state vector is Δ^{2^n} there will be one case for each possible condition.

Algorithm 1 The pseudocode for the boolean kalman filter with the particle filter subsample

```
# Initialize a set of $N$ particle and give an initial probability of $\frac{1}{N}$
$\mathbf{x}(0)_i \sim \Pi_{0|0}, W(0)_i = \frac{1}{N}$, for $i = 1, \cdots, N$
for $k = 1, 2, \cdots$ do
    # Update each particle with the corresponding update function $f$
    for $i = 1$ to $N$ do
        $\mu(k)_i = \mathbf{f}(\mathbf{x}(k-1)_i)$
        # Update the corresponding probability of each particle with the probability gained looking at the output of the system
        $V(k)_i = P(\mathbf{Y}(K)|\mu(k)_i)W(k-1)_i$
    end for
    # Subsample the possible particle relating to the new probability distribution
    $\{j(k)_i\}_{i=1}^{N}$ = Sample $\{V(k)_{i=1}^{N}\}$
    # For each sample integrate a noise and estimate the weight to the possibility to reach the output with the integrated noise.
    for $i = 1$ to $N$ do
        $\mathbf{x}(k)_i) = \mu(k)_{j(k)_i} \otimes \mathbf{n}_k$
        $\tilde{W}(k)_i = \frac{P(\mathbf{Y}(K)|\mathbf{x}(k)_i)}{P(\mathbf{Y}(K)|\mu(k)_{j(k)_i})}$
    end for
    # Normalize the weights
    $W(k)_i = \frac{\tilde{W}(k)_i}{\sum_{i=1}^{N} \tilde{W}(k)_i}$
    # The estimated next state is the weighted sum of the subsampled states
    $\mathbf{X}^{MS}(k) = \sum_{i=1}^{N} W(k)_i \mathbf{x}(k)_i$
end for
```

In the algorithm we add a noise n_k to affect a candidate state vector (namely a particle). This is for the same reasons that have been taken into consideration when dealing with models with noise. Using an appropriate noise could lead to a reduction of the number of particles requested to reach an appropriate state estimation, adding some random perturbation.

The probability $P(\mathbf{y}(K)|x(k))$ depends on the experiment done in order to obtain the observation. In the simulation reported in Chapter 5 we use the pure expectation of the given output. This leads to great results but is only possible since the data are simulated. Once the state is estimated it can be used to determine if the network is faulty. The procedure is similar to what was done with the faulty observer in (4.13). First we replace $\hat{x}_e$ with the state estimated with the particle filter. Then we continue in order to obtain α as in (4.14). The conclusions must be drawn by looking at when α goes to zero, as reported in the previous discussion.

This method works also when dealing with PBNs and asynchronous networks. It only required some small tweaks. In particular, it needs to take two distinct probabilistic distributions into account: one describing the possible initial state and the second that describes the network used at the current time to update the system. We can operate by increasing the number of particles and keep the algorithm unchanged. The number of particles needs to be big enough to ensure an appropriate representation of the update function. This can be resource intensive also for very small networks. It is sufficient to have a very small probability associated with an update function to increase drastically the number of particles requested. In alternative, it is possible to modify a bit the code of the filter. Instead of randomly choosing the update function, let us create a set of new particles. These will be the result of the original one updated with each possible function. The final particle will weigh as the original one multiplied by the probability of the function used. This "dynamical" approach reduces the request of particles and ensures that each update function has been taken into consideration at each time step. If the system is asynchronous it is necessary to modify the structure depending on the type of the update function. Since the cascade update depends on the consequences of the past update, one cannot treat all particles equally. But as shown in (3.15) it is also necessary to keep track of the individual history of each particle. Then we update each particle with the relative update function. The rest of the algorithm remains unaltered, but the estimated state is returned without the history since it makes no sense to average the changes on different processes. It is possible to combine both tweaks in order to generate an estimation of more complex systems.

In the next chapter, we will present the difference between the results simulating a real dynamical system.

Chapter 5

Simulation on a real model

In this chapter we focus on a simulation of a real pathology. It is based on what is reported in the paper [13]. By following up the reasoning made in Chapter 1, the authors use the data obtained from multiple cells to generate the boolean models reported in Table 5.1. The data has been used to group the target genes and proteins into twelve subgroups. For the scope of this thesis, it is not interesting to study what these groups represent. Instead, we will focus on the fact that a cell that develops a tumor will change its internal behavior and consequently the functions that describe the interaction between the group of genes change. These groups have been represented with a low-case letter "g_i" in a non-tumor cell and with an upper-case letter "G_i" in a tumor cell. The group is the same in the two cases but they evolve with different functions and the different notation helps to clarify. For the simulation, we assume that a cell switches from the healthy state to the tumor one instantaneoulsy: $g_i(k) \rightarrow G_i(k)$. This is a simplification: the real pathology is a consequence of different faults that can happen at different times. To justify this we present two motivations: first in a boolean model the time step depends on the experiment, and it is possible to assume it is big enough to cover the transformation. Secondly, the scope of this study is to identify if the model is faulty or not. The introduction of more models can increase the number of shared trajectories, leading to a higher chance of false alarms and as a consequence the reduction of the precision. Dealing with multiple models can increase the number of false reports. As we will see, in this case, it is only requested the knowledge of the normal model, in order to obtain a good fault detector. Another consideration to do is the output function. This function is not reported in the paper, as the scientists created the model based on all the data available to them. They later assumed that this data is always available and there are no hidden variables, which in our model is equivalent to saying that $y(k) = x(k)$

Since this, it has been created as a custom one. As a future study, it is interesting to understand what is the conclusion when the system is not observable. Keeping in mind that reducing the number of observed variables leads to less expensive experiments.

In this chapter, the previous PBNs are tested in order to visualize how powerful is the method described in Chapter 4. In order to show this, we proceed as follows: first, we emulate the system as a classical synchronous PBN, next we reconstruct the system trajectory with the state estimator and the Kalman filter. The test will be performed by assuming for the first 200 steps the normal model, followed by the evolution with the faulty model. This test has been run with a random initial condition that is not shared with the observer. Then we repeated the same test with the asynchronous update scheme.

First, we report the result on the synchronous probabilistic boolean network. These results are compatible with the ones reported in [9], where the authors classified the fault as possibly detectable. In our simulations, we confirmed this since α goes to zero in a couple of steps after the switch of the model in almost every simulation done. It is interesting to look at the difference of α between the two estimation methods. The particle filter seems to outperform the other method. This is because we perfectly know the model. In the real world, additional interference reduces the accuracy of the result.

Figure 5.3 is reported as an interesting case, since the system reached an equilibrium configuration. When the model is switched to the faulty the configuration is not more equilibrium is not preserved and the configuration restart to change in time. Since the estimator will still focus on the normal equation the estimated state remains fixed on the equilibrium. The difference becomes evident even without looking at α.

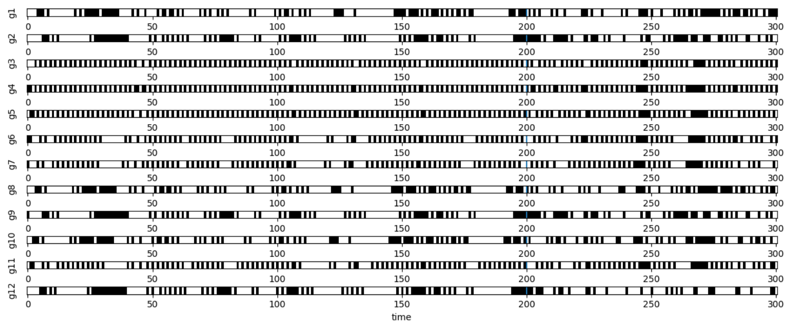
Unfortunately, when we repeat the same type of tests with the asynchronous updated model detected dropped significantly. The correct estimation of the system is still possible with the modified particle filter, but the intersection between $\mathcal{T}_f$ and $\mathcal{T}_{f,faulty}$ is increased and there are more cases of not detectable fault.

One can ask himself what happens when the system moved from the synchronous and asynchronous update type since the same fault can become undetectable. The answer should be searched in the trajectories set. These sets are not shared between the synchronous system and the asynchronous one. From the simulations we extract that there are a low number of trajectories that the faulty system could go through that are not shared with the non-faulty system. This leads the estimation filter to be able to track the evolution of the faulty system with the normal system update function. Hence, α has some difficulties to reach zero reducing the possibility to detect the fault scenario.

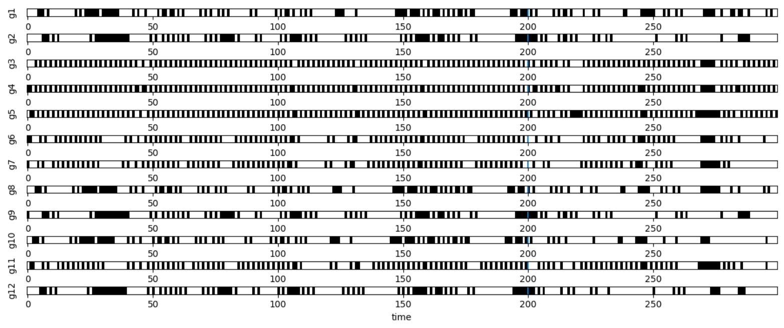
Boolean Rule for Tumor cells	Probability
$G_1(k+1) = (\neg G_5(k) \wedge (\neg G_6(k) \wedge G_{12}(k)) \vee (G_5(k)) \wedge (\neg G_6(k) \vee G_{12}(k)))$	1
$G_2(k+1) = G_{11}(k)$	1
$G_3(k+1) = (\neg G_8(k) \wedge (G_4(k) \wedge G_9(k))) \vee (G_8(k) \wedge (G_4(k) \vee G_9(k)))$	1
$G_4(k+1) = G_8(k)$	0.58
$G_4(k+1) = G_1 0(k)$	0.42
$G_5(k+1) = \neg G_6(k) \wedge (\neg G_{12}(k) \wedge G_1(k)) \vee G_6(k)$	1
$G_6(k+1) = (\neg G_{12}(k) \wedge (\neg G_1 k \wedge G_5(k))) \vee (G_{12}(k) \wedge G_5(k))$	1
$G_7(k+1) = G_1 0(k)$	1
$G_8(k+1) = G_4(k)$	1
$G_9(k+1) = G_3(k)$	1
$G_{10}(k+1) = G_7(k)$	0.34
$G_{10}(k+1) = (\neg G_1(k) \wedge (G_5(k) \vee G_6(k))) \vee (G_1(k) \wedge (G_5(k) \wedge G_6(k)))$	0.33
$G_{10}(k+1) = (\neg G_5(k) \wedge (G_6(k) \wedge \neg G_{12}(k))) \vee (G_5(k) \wedge (G_6(k) \vee \neg G_{12}(k)))$	0.33
$G_{11}(k+1) = G_2(k)$	1
$G_{12}(k+1) = (\neg G_1(k) \wedge (\neg G_5(k) \wedge G_6(k))) \vee (G_1(k) \wedge (\neg G_5(k) \vee G_6(k)))$	0.55
$G_{12}(k+1) = G_2(k)$	0.45

Boolean Rule for Non-tumor cells	Probability
$g_1(k+1) = g_8(k)$	1
$g_2(k+1) = g_{12}(k)$	1
$g_3(k+1) = (\neg g_7(k) \wedge (\neg g_4(k) \wedge g_5(k))) \vee (g_7(k) \wedge (\neg g_4(k) \vee g_5(k)))$	1
$g_4(k+1) = (\neg g_5(k) \wedge (\neg g_6(k) \wedge g_7(k))) \vee (g_7(k) \wedge g_5(k))$	1
$g_5(k+1) = (\neg g_6(k) \wedge (\neg g_7(k) \wedge g_4(k))) \vee (g_6(k) \wedge (\neg g_7(k) \vee g_4(k)))$	1
$g_6(k+1) = g_7(k)$	1
$g_7(k+1) = (\neg g_4(k) \wedge g_6) \vee (g_4(k) \wedge (\neg g_5(k) \vee g_6(k)))$	0.37
$g_7(k+1) = (\neg g_3(k) \wedge (\neg g_8(k) \wedge g_{10}(k))) \vee (g_3(k) \wedge (\neg g_8(k) \vee g_{10}(k)))$	0.32
$g_7(k+1) = (\neg g_1(k) \wedge g_3(k)) \vee (g_1(k) \wedge (\neg g_8(k) \vee g_3(k)))$	0.31
$g_8(k+1) = g_{10}(k)$	1
$g_9(k+1) = g_{12}(k)$	1
$g_{10}(k+1) = g_8(k)$	0.36
$g_{10}(k+1) = g_6(k)$	0.33
$g_{10}(k+1) = (\neg g_9(k) \wedge (g_{11}(k) \vee g_{12}(k))) \wedge (g_9(k) \wedge g_{11}(k))$	0.31
$g_{11}(k+1) = g_6(k)$	1
$g_{12}(k+1) = (\neg g_2(k) \wedge (g_9(k) \wedge g_1 1(k))) \vee (g_2(k) \wedge (g_9(k) \vee g_{11}(k)))$	0.35
$g_{12}(k+1) = (\neg g_{10}(k) \wedge (g_1(k) \wedge \neg g_3(k))) \vee (g_{10}(k) \wedge (g_1(k) \vee \neg g_3(k)))$	0.33
$g_{12}(k+1) = g_1(k)$	0.32

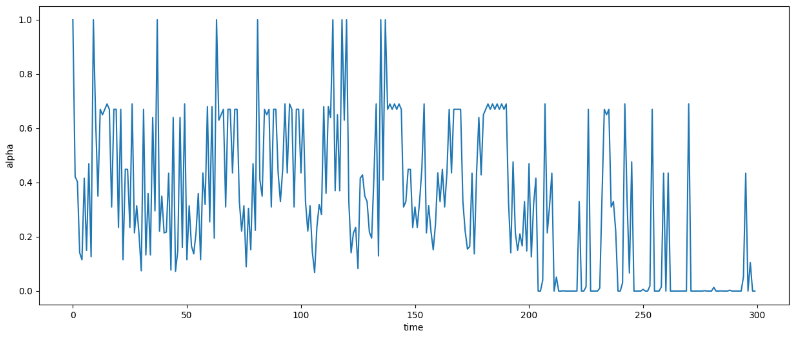
Table 5.1: Tables of the PBN reported in the cited paper.



(a)

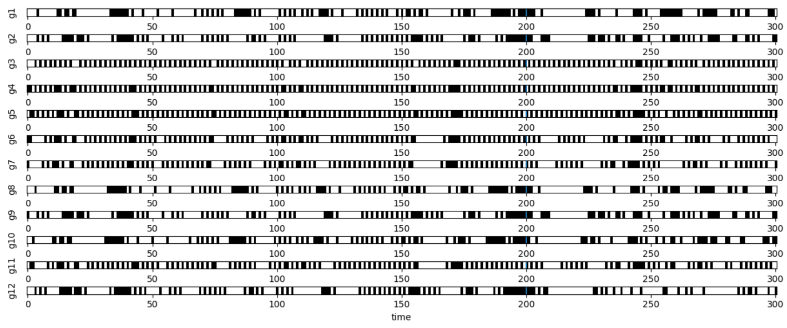


(b)

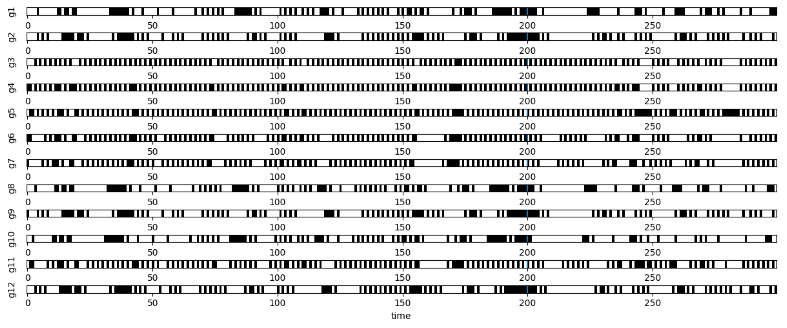


(c)

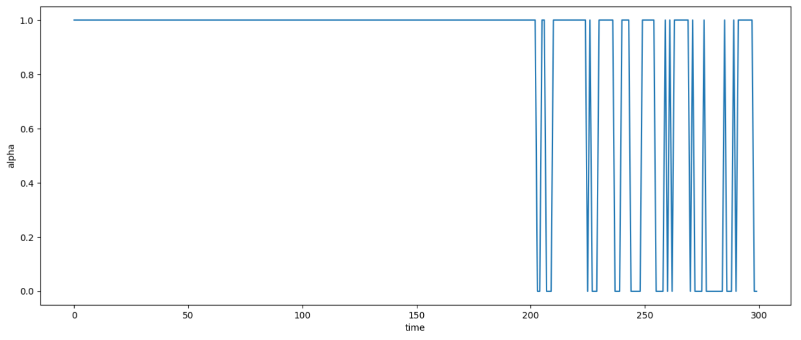
Figure 5.1: In (a) is reported the evolution of the simulated system. Each line corresponds to a specific target. At the time 200 the model changed to the faulty one. In (b) is reported the estimation of the state with the L matrix and the fault detection observer. In (c) is reported the α in time. It is possible to see that α reaches zero after the model has changed to the faulty one.



(a)

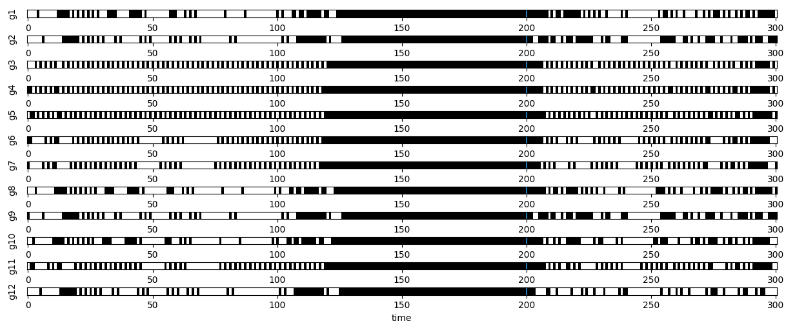


(b)

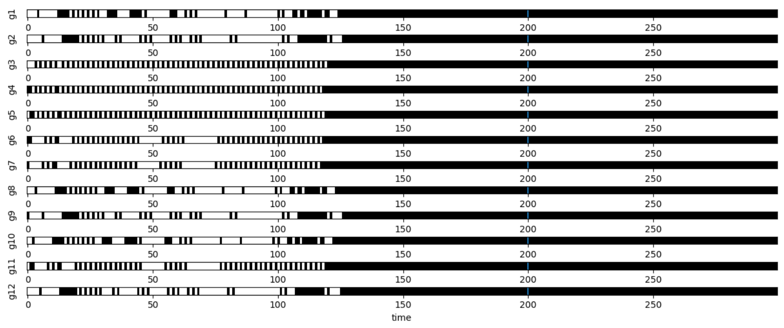


(c)

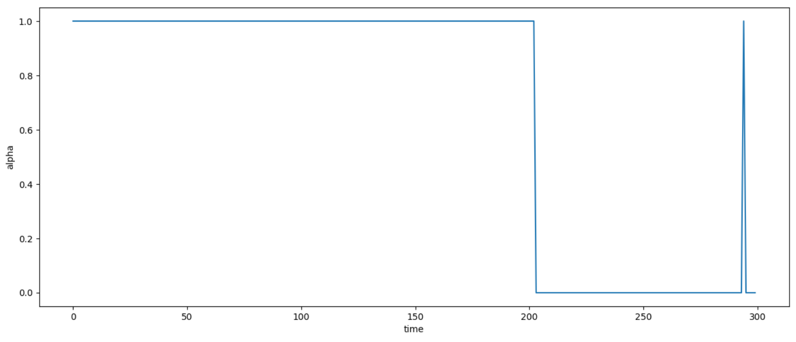
Figure 5.2: In (a) is reported the evolution of the simulated system. Each line corresponds to a specific target. At the time 200 the model changed to the faulty one. In (b) is reported the estimation of the state with the kalman filter matrix and the fault detection observer. In (c) is reported the α in time. It is possible to see that α is almost one until the model has been changed to the faulty one.



(a)

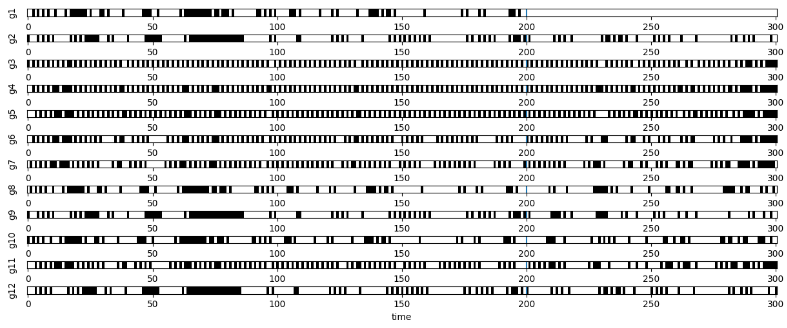


(b)

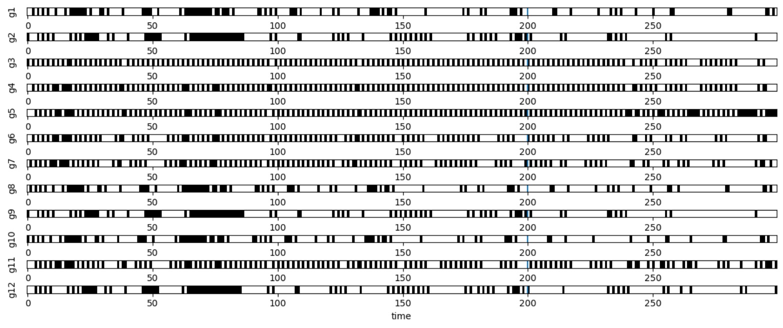


(c)

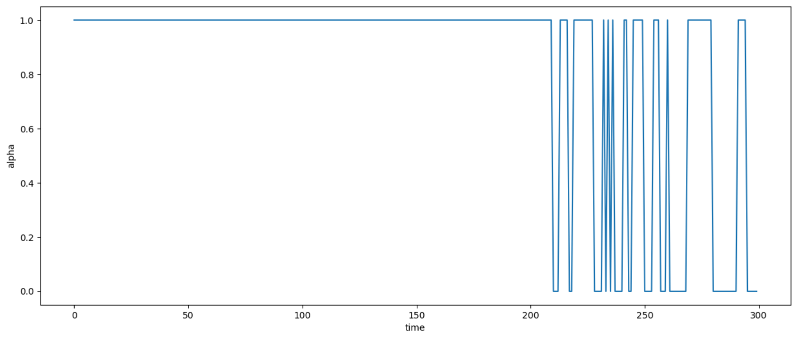
Figure 5.3: Here is reported an interesting case, where the system status is in equilibrium until the update function change and the estimated filter cannot "follow" the new trajectory. This led to a drop in the α value.



(a)



(b)



(c)

Figure 5.4: In (a) is reported the evolution of the simulated system with the cascade update scheme. Each line corresponds to a specific target. At the time 200 the model changed to the faulty one. In (b) is reported the estimation of the state with the kalman filter and the fault detection observer. In (c) is reported the α in time. It is possible to see that α reaches zero after the model has changed to the faulty one.

Chapter 6

Conclusion

In this thesis, we started with by presenting the boolean algebra and the boolean systems. We showed how this set up can be used to model some biological functions and how is it possible to introduce some improvements in order to encapsulate more complex behaviors. We concluded the theoretical part with the introduction of the study needed in order to perform some sort of fault detection. All these results have been used to simulate a real system and showed that the introduction of more complex behaviors as the asynchronous update has a huge impact on the detectability of a fault. This must be a warning for future studies: every problem has more study stages, and it is very unlikely that the initial proposed solution works when the approach starts to go deeper into the problem. A second remark that we want to do regards a purely theoretical approach to pathology. From the model and knowledge of a biological issue, one can think to design some sort of feedback that leads the system to behave as a normal one. This is correct in theory but is impossible with the current state of technology. To do so, one should have access to all the pathology-afflicted cells, estimate the state of the cell and generate the correction input. This must be done for each cell and each pathology, and is based on the assumption that the immune system accepts this external component. Even if we can design a solution to the problem, it is necessary to take into account all the complications of a biological environment. This should not discourage a mathematical approach to the biological world, but raise awareness of what needs to be done in this field of study.

Bibliography

[1] J. Roberts J. Watson N. Hopkins. *Molecular Biology of the Gene*. 7. 2013, p. 85. DOI: 10.1016/0302-4598(89)85035-4.

[2] Francis Crick. “On protein synthesis”. In: *The Symposia of the Society for Experimental Biology* 12 (1958), pp. 138–163.

[3] S. A. Kauffman. “The Origins of Order: Self-organization and Selection in Evolution”. In: *International Journal of Biochemistry* 26.6 (1994), p. 855. DOI: 10.1016/0020-711x(94)90119-8.

[4] Madalena Chaves. “Methods for qualitative analysis of genetic networks”. In: *2009 European Control Conference (ECC)*. 2009, pp. 671–676. DOI: 10.23919/ECC.2009.7074480.

[5] Sui Huang Kwang-Hyun Cho Jae Il Joo Joseph X. Zhou. “Determining Relative Dynamic Stability of Cell States Using Boolean Network Model”. In: *Nature / Scientific Reports* 8 (2018), pp. 2045–2322. DOI: 10.1038/s41598-018-30544-0.

[6] Masado Ishii, Jacob Gores, and Christof Teuscher. “On the sparse percolation of damage in finite non-synchronous random Boolean networks”. In: *Physica D: Nonlinear Phenomena* 398 (2019), pp. 84–91. DOI: 10.1016/j.physd.2019.05.011.

[7] Trinh Van Giang, Tatsuya Akutsu, and Kunihiko Hiraishi. “An FVS-Based Approach to Attractor Detection in Asynchronous Random Boolean Networks”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 19.2 (2022), pp. 806–818. DOI: 10.1109/TCBB.2020.3028862.

[8] Ritwik Layek et al. *Cancer therapy design based on pathway logic*. Dec. 2010. DOI: 10.1093/bioinformatics/btq703. URL: https://doi.org/10.1093/bioinformatics/btq703.

[9] Thomas Leifeld, Zhihua Zhang, and Ping Zhang. “Fault detection for probabilistic boolean networks”. In: *2016 European Control Conference (ECC)*. 2016, pp. 740–745. DOI: 10.1109/ECC.2016.7810377.

[10] Ulisses Braga-Neto. “Optimal state estimation for Boolean dynamical systems”. In: *2011 Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*. 2011, pp. 1050–1054. DOI: 10.1109/ACSSC.2011.6190172.

[11] Arghavan Bahadorinejad, Mahdi Imani, and Ulisses M. Braga-Neto. “Adaptive Particle Filtering for Fault Detection in Partially-Observed Boolean Dynamical Systems”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 17.4 (2020), pp. 1105–1114. DOI: 10.1109/TCBB.2018.2880234.

[12] Mahdi Imani and Ulisses M. Braga-Neto. “Particle filters for partially-observed Boolean dynamical systems”. In: *Automatica* 87 (2018), pp. 238–250. DOI: 10.1016/j.automatica.2017.10.009.

[13] Danh Cong Nguyen and Farhad Azadivar. “Early detection of cancer by regression analysis and computer simulation of gene regulatory rules”. In: *2012 International Conference on Biomedical Engineering (ICoBE)*. 2012, pp. 144–148. DOI: 10.1109/ICoBE.2012.6178972.