CONTROLLABILITY ANALYSIS AND CONTROL DESIGN OF BIOLOGICAL SYSTEMS MODELED BY BOOLEAN NETWORKS

by

Vianney Mixtur

B.S. in Electrical and Computer Engineering, ENSEA, 2016

Submitted to the Graduate Faculty of

Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2018

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This thesis was presented

by

Vianney Mixtur

It was defended on

April 4, 2018

and approved by

Natasa Miskov-Zivanov, PhD, Assistant Professor
Departement of Electrical and Computer Engineering

Zhi-Hong Mao, PhD, Associate Professor
Departement of Electrical and Computer Engineering

Amro El-Jaroudi, PhD, Associate Professor
Department of Electrical and Computer Engineering

Thesis Advisor: Natasa Miskov-Zivanov, PhD, Assistant Professor
Departement of Electrical and Computer Engineering

Copyright © by Vianney Mixtur

2018

CONTROLLABILITY ANALYSIS AND CONTROL DESIGN OF BIOLOGICAL NETWORKS MODELED BY BOOLEAN NETWORKS

Vianney Mixtur, M.S

University of Pittsburgh, 2018

Cell signaling networks are often modeled using ordinary differential equations (ODEs), which represent network components with continuous variables. However, parameters such as reaction rate constants are needed for ODEs are not always available or known, and discrete approaches such as Boolean networks (BNs) are used in such cases. BNs have been applied in the past, in particular, as means to determine network steady states. The goal of this work is to explore the use of BNs from a control theory point of view, that to help manipulate biological systems more efficiently. In this thesis, we propose two methods to analyze and design control strategies for BNs. The first method, based on the algebraic state-space representation of BNs, consist of defining control strategies to reach predetermined states, namely, given a desired output, find all possible system state transition trajectories to that output, and design an input sequence leading to it. The second method aims at introducing an alternative and an extension of the first method in the sense that it offers broader possibilities for the representation of time and it is scalable to BNs of bigger size. This method is based on binary decision diagrams (BDDs), a data structure very efficient to represent logical functions and allow us to relate outputs of a network to its inputs no matter how many layers it contains and whether or not it has a cyclic structure.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE-ACKNOWLEDGMENT

First, I would like to thank my advisor, Dr. Natasa Miskov-Zivanov for her trust and precious guidance during those two years. Second, I would like to thank Dr. Kara Bocan and the PhD students of the MeLoDy lab, Khaled Sayed, Adam Butchy, Emilee Holtzapple, Yasmine Ahmed and Handa Ding for providing valuable help when I faced technical problems. Also, I want to thank Gavin Zhou, Alexandre Terrier, Nathan Renaudie, Mickael Vimbert and the MeLoDy lab as a whole for the great atmosphere that was present during my time in here.

Third, I would like to thank Thomas Tang, Carine Sabouraud of the International relations team at ENSEA and Dr. Mahmoud El Nokali for giving me the opportunity to study at the University of Pittsburgh. Sandy Weisberg for her constant support to all international students.

Last but not least, I would like to thank my family.

1.0 INTRODUCTION

1.1 PREVIOUS WORK

Modeling enables gaining insights into the modeled system, understanding its behavior, or predicting responses to various stimulations or perturbations. Boolean networks (BNs) have become a popular computational tool to model biological systems [1][2] [5][6][7]. This popularity is motivated by the fact that, despite this modeling approach being conceptually simple, it can provide valuable insights into the modeled biological systems.

As with other systems, in order to understand and control the behavior of biological systems, we can study their controllability. Particularly, one of the main aims of the study of controllability and observability is to develop strategies to move biological systems from an undesirable state, such as a disease, to target state, such as restored health [11][12][13][14][15][16]. However, biological systems are usually very complicated in terms of their component interactions, that is, they often include intertwined feed-forward and feedback loops. These loops usually govern system's transient and steady-state behavior [8]. These complex interactions have been a major obstacle when developing control strategies for biological systems [13].

Recently, several approaches have been suggested to handle this problem. For example, a method offered by Cheng et al. [9] translates the control theoretical approach of continuous systems to Boolean Networks, and, thus, has the advantage of being quite intuitive. This so called *algebraic state-space representation* (ASSR) lead to several other publications. Pursuing this work, Laschov and Margaliot developed an approach based on nonnegative matrices and the Perron-Frobenius theory to investigate the controllability of BNs [20], and a graph-theoretic approach to investigate the observability [22]. Bof et al. extended the known results to asynchronous random networks [23]. A recent study provides algorithms to determine whether a Boolean Network is controllable or observable [15]. This method uses advanced algebraic tools such that rings and ideals of rings. However, it does not provide a way to design control strategies. The methods mentioned so far, rely heavily on the dynamics of the networks as they all explore the state-transition (graph) of the network.

Previously, Thomas investigated the logical structure of systems [8]. Gates and Rocha then showed that structural approaches were not sufficient, as exploring the control of complex networks requires both structure and dynamics [14]. Methods using Binary Decision Diagrams (BDDs) tend to incorporate both in their analysis. One interesting use of BDDs for the analysis of Boolean Networks is the method developed by Garg et al [29][30][31]. In this method, the state of the network is represented by a Boolean vector of size N (number of networks elements) which is then represented by a BDD.

1.2 CONTRIBUTION OF THIS THESIS

1.2.1 Control Design

Most of the results cited in the previous section, just answer the question of whether a given Boolean network is controllable/observable or not, and do not provide solutions to effectively control biological systems. Our goal here is to show how to define control strategies to reach predetermined desirable states for networks by controlling the input of the network under the assumptions of a synchronous update scheme for the time representation of the network [34]. To this end, the contributions of this work include:

- A control strategy to reach desirable states in a cyclic Boolean Network;
- A methodology to deduce input sequences that lead to a desired system state and a desired output pattern;
- An application of the proposed control strategy to biological systems.

In the following, we first provide a brief background on BNs (Chapter 2) and an overview of the Algebraic State-Space Representation (ASSR) described in [9] [20][21] (Chapter 3). Next, we list the steps in the process of defining a control strategy for a network and provide an algorithm to design an input sequence that leads to the desired behavior (Chapter 4). To conclude with this approach, we show an example of application of our proposed methodology (Chapter 6).

1.2.2 Unrolling logic via Binary Decisions Diagrams

Next, we provide a brief background on BDDs (Chapter 8), describe our data structure and our code implementation to model and manipulate BDDs (Chapter 9). We then describe in detail our unrolling methodology (Chapter 10). In this second part, the contributions of this thesis include:

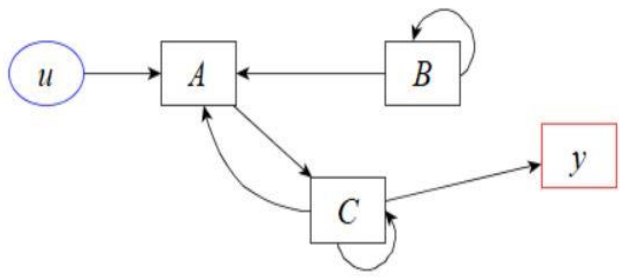
- A data structure to represent, manipulate and analyze Boolean networks
- A methodology to express outputs of a Boolean networks as function of the inputs even in cases where the relationship between the two is not easily identifiable
- A method to study Boolean network controllability and observability that utilizes the BDD data structure and our unrolling Boolean network implementation.

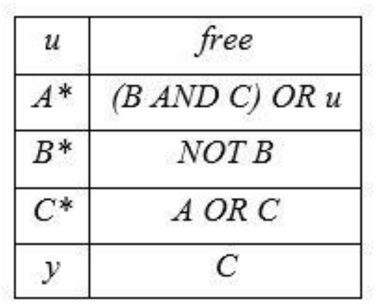
To conclude this thesis, we apply our methodology on a real life, biological example, the T cell signaling pathway [34] (Chapter 11).

2.0 BOOLEAN NETWORK PRELIMINARIES

2.1 STRUCTURE

In Boolean Networks, the variables (corresponding to nodes of the network) can take only two values, 0 and 1. The node value 0 means that the modeled system element (e.g., a protein or a gene), is not expressed, or is below a certain concentration threshold, while the value 1 is used to represent that the element is expressed, or is above a certain threshold. The nodes are connected by edges depicting interactions between system elements and defining regulators for each element. In Figure 1 (left), we show an example BN with one input, *u*, one output, *y*, and three internal nodes, *A*, *B*, and *C*. As it is often the case with BNs, this toy example also has several feedback loops.





u	*free*
*A**	*(B AND C) OR u*
*B**	*NOT B*
*C**	*A OR C*
y	*C*

Figure 1. Toy example of a Boolean network (left). Update rules for elements of the BN (right).

Each node in the BN has a corresponding update rule, which is a function of its regulators and includes logic operators such as disjunction (OR), conjunction (AND), and negation (NOT). In Figure 1 (right), we show example update rules for the BN in Figure 1 (left). The update rules are used to determine values that elements will take at each discrete time step, according to the values of their regulators. The "* " next to *A*, *B* and *C* indicates that we are referring to the next value of the element.

For example, if an update rule for node C is a disjunction between node A and node B, then C will obtain value 1 in the following time step when either A or B has value 1 at a current time step. In a biological system, this interaction can, for instance, describe a gene C which is expressed when either transcription factor A or transcription factor B is above a certain concentration threshold. In Table 1, we show the truth table for disjunction. The truth table of a logical function lists all combination of the inputs in the domain of the function, and for each input combination it gives the corresponding output [24]. This is well defined because the domain of a logical function is finite.

Table 1. Truth table of DISJUNCTION (OR)

A	*B*	*A OR B*
0	0	0
0	1	1
1	0	1
1	1	1

Note 1: We showed in Table 1, the truth table for the disjunction with the input combinations listed according to the natural binary order, however in the ASSR, the structure matrices (cf Chapter 3) are defined as if the truth tables were written with the input combinations following a reverse order.

2.2 DYNAMICS

The main difference between BNs and logic combinational circuits is the representation of time in the former. In logic (combinational) circuits [3][4], the time is continuous, and the inputs of a logical function are assumed to have an instantaneous effect on the outputs of that function. Thus, referring to Figure 1, if *A* is the disjunction of *u* and the conjunction of *B* and *C*, and if *A* is currently 0, a transition of *u* from 0 to 1 will cause an instantaneous transition from 0 to 1 for *A*.

BNs are sequential networks, the time is discretized, and the variables evolve sequentially. They are updated according to their update rules at each time step, defined by the selected update scheme [6][17][34]. An update scheme is, by definition, a description of when and how each variable should be updated.

The simplest update scheme consists of, at each step *k*, updating all the nodes together simultaneously, using the values of the nodes in the update rules (regulators) at step *k*-1 in the. This update scheme is called simultaneous update scheme.

Another update scheme consists of, at each step *k*, updating only one node randomly picked, using the values of the regulators at step *k*-1. This update scheme is called step-based random-order sequential update scheme.

Additionally, there is another random sequential update scheme that is sometimes used in studies of biological networks. In this update scheme, also called round-based random-order sequential , all the nodes are updated within one update round (or, update cycle) but not simultaneously. At each new round, an update order is randomly generated, and the nodes are updated according to that order. Thus, if a node *A* is selected to be updated after a node *B*, and if *A* depends on *B*, while updating the node *A* at step *k*, one should consider the new value of *B* at step *k*, rather than the value of *B* at step *k*-1.

A more detailed description of the different update schemes is provided in the description of the DiSH simulator [34] developed by Sayed et al.

3.0 ALGEBRAIC STATE-SPACE REPRESENTATION

In this chapter, we briefly review the state space representation developed in [9] [20][21].

3.1 SEMI-TENSOR PRODUCT

We introduce here the semi-tensor product [18-19], which will be used throughout Chapters 3 to 7. It is a generalization of the conventional matrix product for matrices whose dimensions do not match.

Definition 1. Let T be an np-dimensional row vector, X a p-dimensional column vector, $X = (x_1, x_2, \dots x_p)$, and $T^1, \dots, T^p$, where each T^i (i=1,..,p) is an $1\times n$ matrix, are equal blocks of T. A left semi-tensor product, denoted by $\ltimes$, can be defined as

$$T \ltimes X = \sum_{i=1}^{p} T^i \cdot x_i \in R^n \tag{1}$$

Definition 2. Let $M \in M_{m\times n}$ and $N \in M_{p\times q}$. If n is a factor of p or p is a factor of n, then $C = M \ltimes N$ is called the left semi-tensor product of M and N, where C consists of $m\times q$ blocks C^{ij}, such that

$$C^{ij} = M^i \ltimes N_j, i = 1 \dots, m, j = 1, \dots, q \tag{2}$$

where M^i is the i-th row of M and N_j is the j-th column of N.

The semi-tensor product extends all the matrix product properties [10]. Additionally, it extends the matrix product to matrices whose dimensions do not match. Considering this fact, we will indifferently use · or ⋉ to indicate the semi-tensor product.

3.2 LOGICAL FUNCTION AND STRUCTURE MATRIX

In the state-space representation [9], logical values 1 and 0 are represented as:

$$\text{Logical value } 1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ Logical value } 0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad (3-4)$$

With this approach, a logical function is represented using a structure matrix, which is derived from the truth table of the given logical function. The structure matrix is obtained by rewriting the truth table in a reverse natural binary order, extracting each entry of the output column, and interpreting it as a vector. For example, for the truth table of disjunction, shown in Table 1, the corresponding structure matrix is:

$$M_d = \begin{bmatrix} m_{d_{11}} & m_{d_{12}} & m_{d_{13}} & m_{d_{14}} \\ m_{d_{21}} & m_{d_{22}} & m_{d_{23}} & m_{d_{24}} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (5)$$

We can then write the disjunction of *A* and *B* in a matrix form as:

$$C = (A\ OR\ B\ ) = M_d \ltimes A \ltimes B \qquad (6)$$

Following the representation of logic values in (3-4), if we write *A* as a column vector $\begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$, and *B* as a column vector $\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$, then we can compute the product in (6) as

$$C = \begin{bmatrix} m_{d_{11}} & m_{d_{12}} & m_{d_{13}} & m_{d_{14}} \\ m_{d_{21}} & m_{d_{22}} & m_{d_{23}} & m_{d_{24}} \end{bmatrix} \cdot \begin{bmatrix} a_1 b_1 \\ a_1 b_2 \\ a_2 b_1 \\ a_2 b_2 \end{bmatrix} \tag{7}$$

$$C = \begin{bmatrix} m_{d_{11}} a_1 b_1 + m_{d_{12}} a_1 b_2 + m_{d_{13}} a_2 b_1 + m_{d_{14}} a_2 b_2 \\ m_{d_{21}} a_1 b_1 + m_{d_{22}} a_1 b_2 + m_{d_{23}} a_2 b_1 + m_{d_{24}} a_2 b_2 \end{bmatrix} \tag{8}$$

Remark 1. A noticeable point here is that all the vectors and all the columns of the matrices that we use in the state-space representation are columns of some identity matrix I_n, where

$$I_n = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \tag{9}$$

For example, if *A* and *B* from the conjunction above (6-8) are

$$A = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{10 - 11}$$

then the result of the conjunction will be

$$C = (A\ OR\ B\ ) = \begin{bmatrix} 1 * 0 + 1 * 1 + 1 * 0 + 0 * 0 \\ 0 * 0 + 0 * 1 + 0 * 0 + 0 * 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{12}$$

3.3 NETWORK TRANSITION MATRIX

Given the description above, a BN can be converted from a set of logical functions to a set of algebraic equations [9] as shown below in (13-16).

Let us consider the following BN, with inputs $u_1,u_2,..,u_m$, outputs $y_1,y_2,..,y_p$, and internal nodes $A_1,A_2,..,A_n$, all of which can take logical values 0 and 1, and thus, can be represented as in equations (3-4).

$$\begin{cases} A_1(k+1) = f_1\big(A_1(k), A_2(k), .., A_n(k), u_1(k), .., u_m(k)\big) \\ A_2(k+1) = f_2\big(A_1(k), A_2(k), .., A_n(k), u_1(k), .., u_m(k)\big) \\ \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad . \\ \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad . \\ \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad . \\ A_n(k+1) = f_n\big(A_1(k), A_2(k), .., A_n(k), u_1(k), .., u_m(k)\big) \end{cases} \tag{13}$$

$$\begin{cases} y_1(k) = h_1\big(A_1(k), A_2(k), ..A_n(k)\big) \\ \quad\quad\quad\quad . \\ \quad\quad\quad\quad . \\ y_p(k) = h_p\big(A_1(k), A_2(k), .., A_n(k)\big) \end{cases} \tag{14}$$

where f_1, f_2,.., f_n and h_1, h_2,..., h_p are logic functions, k is the current time step, and k+1 is the next time step.

Equations (13) and (14) can be re-written as follows:

$$\begin{cases} A_1(k+1) = M_1 \cdot A_1(k) \cdot\cdot\cdot A_n(k) \cdot u_1(k) \cdot\cdot\cdot u_m(k) \\ A_2(k+1) = M_2 \cdot A_1(k) \cdot\cdot\cdot A_n(k) \cdot u_1(k) \cdot\cdot\cdot u_m(k) \\ \quad\quad\quad\quad\quad\quad\quad . \\ \quad\quad\quad\quad\quad\quad\quad . \\ \quad\quad\quad\quad\quad\quad\quad . \\ A_n(k+1) = M_n \cdot A_1(k) \cdot\cdot\cdot A_n(k) \cdot u_1(k) \cdot\cdot\cdot u_m(k) \end{cases} \tag{15}$$

$$\begin{cases} y_1(k) = H_1 \cdot A_1(k) \cdot\cdot\cdot A_n(k) \\ \quad\quad\quad\quad \vdots \\ y_p(k) = H_p \cdot A_1(k) \cdot\cdot\cdot A_n(k) \end{cases} \tag{16}$$

where M_1, M_2,..,M_n and H_1, H_2,..,H_p are structure matrices, which can be defined according to functions f_1,..,f_n and h_1,..,h_p. We can now write the system state $x(k)$, input $u(k)$, and output $y(k)$ vectors as semi-tensor products:

$$x(k) = \prod_{1}^{n} A_i(k) \tag{17}$$

$$u(k) = \prod_{1}^{m} u_i(k) \ , y(k) = \prod_{1}^{p} y_i(k) \tag{18,19}$$

For the example in Figure 1, vectors *x*, *u*, and *y* can be written as

$$x(k) = A(k) \cdot B(k) \cdot C(k) = \begin{bmatrix} a_{k_1} b_{k_1} c_{k_1} \\ a_{k_1} b_{k_1} c_{k_2} \\ \vdots \\ a_{k_2} b_{k_2} c_{k_2} \end{bmatrix} \tag{20}$$

$$u(k) = \begin{bmatrix} u_{k_1} \\ u_{k_2} \end{bmatrix}, y(k) = \begin{bmatrix} y_{k_1} \\ y_{k_2} \end{bmatrix} \tag{21,22}$$

We can define a matrix *L* as the network transition matrix, and matrix *H* as the state-output transition matrix, such that equations (15) and (16) can be written as:

$$x(k+1) = L \ \cdot x(k) \cdot u(k) \tag{23}$$

$$y(k) = H \cdot x(k) \tag{24}$$

Finally, we introduce a definition of network steady-state.

Definition 3. A network is in a steady-state when the variables associated with network nodes are not changing in time, i.e., the next state is equal to the current state.

Given (23), in the state-space representation, the steady-state is described as follows:

$$x(k+1) = x(k) \tag{25}$$

3.4 CONTROLLABILITY

In order to study the controllability of a system, it is critical to find the set of reachable states. Given an initial state x_0, if the network transition matrix L is known, we can compute the set of reachable states of the BN, $R(x_0)$, as:

$$R(x_0) = Col\left\{\bigcup_{1}^{\infty}(L^{\,i} \cdot x_0\right\} \tag{23}$$

Rx0=Col1∞(Li·x0

Next, we introduce a globally controllable BN, as a network that can reach any state from any given initial state, defined more formally as follows.

Definition 4. A BN is called *globally controllable* if and only if $\forall\, x_0 \in D^n, R(x_0) = D^n$, where D^n is the set of columns of the identity matrix I_n.

However, using the set of reachable states as defined in (26) is not convenient since it requires an infinite number of calculations. In fact, it has been proved [21] that there exists a number N such that

$$R(x_0) = Col\left\{\bigcup_{1}^{N}(L^{\,i} \cdot x_0\right\} \tag{24}$$

Therefore, when investigating the controllability of a network, we will first determine N and then compute R.

4.0 CONTROL STRATEGY

We now describe the steps of the process to define a control strategy given a control objective.

4.1 CONTROL OBJECTIVE

We assume in this section that we are given a Boolean network with *n* internal nodes, *m* inputs and *p* outputs, whose evolution is dictated by (15) and (16), and that our control task is to find a sequence of inputs that will allow us to move from a given state of the network x_0, to a desired state x_d.

The first step is to identify the **initial state of the network**, that is, we need to define initial values for each A_i and each y_i. Initial values for the network can be determined using different approaches. One approach is to use the values corresponding to a particular state of the network that are available in literature or in data. For example, in cell signaling networks, this could mean initializing model elements to represent normal cell or a cell that is affected by a disease. If such a state is not known, we can set the inputs as constants, compute the steady-state and start from there.

The second step is to specify the **desired output** y_d, which defines our control objective, that is, y_d is the goal that we want to reach for the outputs of the network.

4.2 CONTROLLABILITY

Once the control objective is defined, we must check if it is achievable. Investigating the controllability of the network gives insights about the feasibility of the control objective.

If the network is controllable, it is certain that the control objective is feasible. If the network is not controllable, i.e., some states of the network are not reachable, we may still be able to achieve the control objective if one or more states that are reachable can lead to the desired output. In order to check if any of the reachable states leads to the desired output, we use (27) to compute the set of reachable outputs, Y_N, for a given initial state x_0:

$$Y_N(x_0) = Col\{H \cdot R(x_0)\} \tag{28}$$

$$Y_N(x_0) = Col\left\{H \cdot Col\left\{\bigcup_{1}^{N}(L^{\,i} \cdot x_0\right\}\right\} \tag{29}$$

Since multiplying a matrix A by a matrix B is equivalent to multiplying each column of B by A, by multiplying all the elements of $R(x_0)$ by H, we obtain the set of the reachable outputs:

$$Y_N(x_0) = Col\left\{\bigcup_{1}^{N}\left(H \cdot L^{i} \cdot x_0\right)\right\} \tag{30}$$

Ysmaxx0=Col1s(H·Li·x0)

4.3 INPUT DESIGN

We define here a Design Matrix, which we use as part of our control strategy. To build a Design Matrix, *D*, we need to first determine $Y_N(x_0)$ for each x_0, and then, compare this set of reachable outputs to the desired output (or set of outputs). The results of this comparison determine entries in the Design Matrix.

More specifically, each column of *D* corresponds to a different input sequence. Assuming that there are *m* inputs of the system being modeled, $u_1, \ldots, u_m$, and that each input can take either value 0 or value 1, there are exactly 2^m possible input patterns at each time step. Thus, given a step *s*, there are exactly $2^{m \cdot s}$ possible sequences of input patterns up to that step. If we are interested in finding what is reachable within *N* steps, we create a Design Matrix with *N* rows and $2^{m \cdot N}$ columns. The "x's" in the Design Matrix correspond to cases where the matrix is not defined. For example, at step *s*, matrix *D* is not defined for input sequences numbered from ($2^{m \cdot s}$ +1) up to sequence $2^{m \cdot N}$. For a given input sequence, and a given step *s*, if any element in the reachable set of outputs matches desired output y_d, the corresponding element of the *D* matrix takes the value 1, otherwise, it takes value 0. In Table 2, we outline the Design Matrix.

Table 2. Design Matrix

Steps	***D* matrix**									
	1	2	...	2^m	...	$2^{2.m}$	...	$2^{m.s}$	...	$2^{m.N}$
1	d_{11}	d_{12}	...	d_{12^m}	...	x	...	x	...	x
2	d_{21}	d_{22}	⋯	d_{22^m}	⋯	$d_{22^{2.m}}$	⋯	x	⋯	x
⋮	⋮	⋮	⋯	⋮	⋯	⋮	⋯	⋮	⋯	⋮
s	d_{s1}	d_{s2}	⋯	d_{s2^m}	⋯	$d_{s2^{2.m}}$	⋯	$d_{s2^{m.s}}$	⋯	x
⋮	⋮	⋮	⋯	⋮	⋯	⋮	⋯	⋮	⋯	⋮
N	d_{N1}	d_{2N}	⋯	d_{N2^m}	⋯	$d_{N2^{2.m}}$	⋯	$d_{N2^{m.s}}$	⋯	$d_{N2^{m.N}}$

4.3.1 Overall Input

From the column number previously identified we deduced the overall input that leads to the desired output.

Definition 5 The overall input at step *s* is the product of the inputs from 0 to *s-1* as shown in (31). It stores in one vector, all the information contained in the input sequence.

$$U(s) = u(0) \cdot u(1) \cdot \ldots \cdot u(s-1) = \prod_{0}^{s-1} u(k) \tag{31}$$

We recall that the semi-tensor product of *s* vectors of size 2^m being columns of the identity matrix I_{2^m} is a column of the identity matrix $I_{2^{m.s}}$ [10]. In order to achieve our control objective, we can represent *U*(*s*) as:

$$U(s) = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} = \delta_{sl}^{2^{m.s}} \tag{32}$$

where $\delta_{sl}^{2^{m.s}}$ is a 2^{ms} column vector whose rows are all 0, except the one that matches with, *l*, the selected column of *D*, and *s*, the selected row of *D*. *s* is the step where the control objective will be achieved, and *l* is the *l*-th overall input sequence among all the designable input sequences.

4.3.2 Input sequence

Since we assume that we have control of the inputs at every step, we want to derive, from *U*(*s*), the full input sequence, that is, the values of all individual inputs $u_1(k),\ldots,u_m(k)$ for all steps k=0..*s*-1. To do so, we need to derive the expression of *U*(*s*) as a function of individual inputs:

$$U(s) = \prod_{0}^{s-1} u(k) = \prod_{k=0}^{s-1} \prod_{j=1}^{m} u_j\,(k) \tag{33}$$

Given that *U(s)* is a vector with 0 at each row but one, and the terms in the product that define *U(s)* are vectors with two rows, one with value 0 and one with value 1, in order to determine individual input values at each step *k*, we just need to find the location of the 1s in each $u_j(k)$. This can be done by introducing the notation shown in (34)-(35), and expressing each entry of *U*(*s*) in terms of the $u_j(k)$:

$$u_j(k) = \begin{bmatrix} u_j^0(k) \\ u_j^1(k) \end{bmatrix} \tag{34}$$

$$u_j(k) = u_{jk} = \begin{bmatrix} u_{jk}^0 \\ u_{jk}^1 \end{bmatrix} \tag{35}$$

Table 3 shows the expression of *U*(*s*) in terms of the $u_j(k)$.

Table 3. Expression of the entries of *U*(*s*)

***U*(*s*)**	**Row number**
$u_{10}^0 \cdot u_{20}^0 \cdot \dots \cdot u_{m0}^0 \cdot u_{11}^0 \cdot \dots \cdot u_{ms}^0$	1
$u_{10}^0 \cdot u_{20}^0 \cdot \dots \cdot u_{m0}^0 \cdot u_{11}^0 \cdot \dots \cdot u_{ms}^1$	2
$\vdots$	$\vdots$
$u_{10}^1 \cdot u_{20}^1 \cdot \dots \cdot u_{m0}^1 \cdot u_{11}^1 \cdot \dots \cdot u_{ms}^1$	$2^{m.s}$

Remark 2. We can notice two things from Table 3. First, if we omit the superscripts in the terms of the products in the left column of Table 3, all the entries contain the pattern $u_{10} \cdot u_{20} \cdot \dots \cdot u_{m0} \cdot u_{11} \cdot \dots \cdot u_{ms}$. Second, with the notation introduced in (34) and (35), a unique pattern appears in

the superscripts of the products at each row of the table. Indeed, the superscripts follow the natural binary order.

From these observations, we can write,

$$U(s) = \left(\prod_{i=0}^{s-1} \prod_{j=1}^{m} u_{ji}{}^{bin(j+i.m,\ r-1,\ m.s)} \right)_{r \in [1\ 2^{m.s}]} \qquad (36)$$

where *bin*(*i*,*z*,*n*) is the *i*-th digit of the binary representation of the integer *z* over *n* bits [24][38][39]
.

According to (32), for one and only one row number r_d, we have,

$$\prod_{i=0}^{s-1} \prod_{j=1}^{m} u_{ji}{}^{bin(j+i.m,\ r_d-1,\ m.s)} = 1 \qquad (37)$$

From (36), we can deduce the location of all the “1s” that define the $u_j(k)$*’s,* and consequently, the full input sequence. Indeed, for r_d and all (*i*, *j*) Є ([0 *s*-1] x [1 *m*]),

$$u_{ji}{}^{bin(j+i.m,\ r_d-1,\ m.s)} = 1 \qquad (38)$$

$$u_{ji}{}^{\overline{bin(j+i.m,\ r_d-1,\ m.s)}} = 0 \qquad (39)$$

where “-“on top of the exponent represents logical negation operator (NOT).

5.0 MATLAB IMPLEMENTATION

We implemented our method to compute the *D* matrix and derive the input design in MATLAB by adding new functions to the functions already available in the toolbox at http://lsc.amss.ac.cn/~dcheng/stp/STP.zip developed by Cheng et al [21].

We next give a description of the main functions that we implemented. We stated in Chapter 3.3 that, given a network transition matrix *L* and an initial state x_0, there exists a finite number of steps *N* after which no new states can be computed. Since this number is of great interest, provided that it prevents us from infinite computation, we will describe the algorithm to obtain *N* first (Algorithm 1).

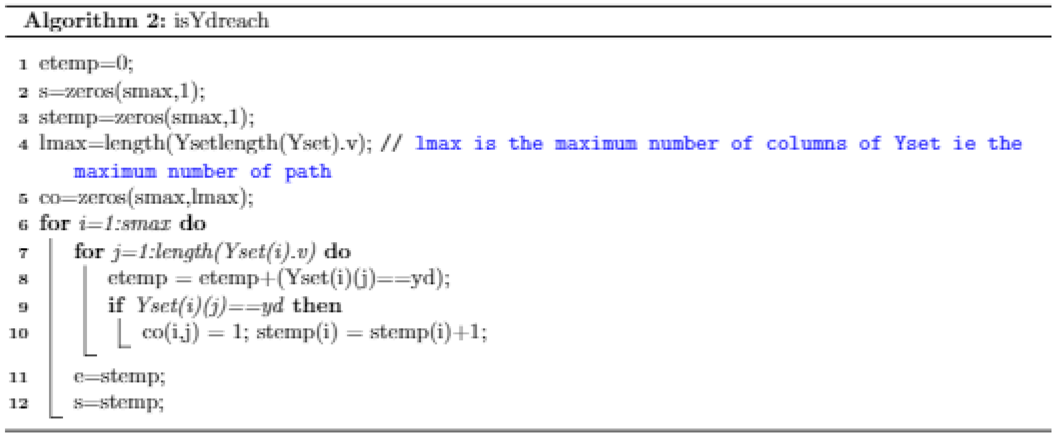
Table 4 Algorithm 1: `maxstep` (algorithm to compute *N*)

```
Algorithm 1: maxstep
new=1;
k=1;
N=$2^n$;
old_columns=zeros(1,N); // old_columns will contain the column of R(x0,s-1)
new_columns=zeros(1,N); // and new_columns those of R(x0,s)
while $new == 1$ do
    Lts=$Lt^k$;
    Rx0s=lsp(Lts,x0);
    vRx0s=Rx0s.v; // Columns of R(x0,s)
    lmax=length(vRx0s); // Should be $2^m i$
    for $i = 1 : N$ do
        for $l = 1 : lmax$ do
            if $(vRx0s(1) == 1)$ then
                new_columns(i)==1; // new_column(1) is 1 if lm([1],N) is a column of
                    R(x0,s) 0 if not.  So we basically storing non-repetively the columns
                    of R(x0,s)
    if (new_columns==old_columns) then
        new=0;
    else
        old_columns=new_columns; k = k+1;
smax = k;
```

This function takes as parameters the network transition matrix *L* renamed *Lt* here, the initial state x_0 and the number of internal nodes *n* and returns the *N* (`smax` in the Algorithm 1). The process consists of computing the set of reachable states at steps *s* (line 8), from which we compute the new states obtained at each step (double `for` loop), by updating `new_columns`. `old_columns` and `new_columns` are vectors of size *SP*, which is the maximum number of states for the *n* internal nodes. `new_columns/old_columns`(*i*) = 1, if the corresponding state is found. We then test the equality between `new_columns` and `old_columns` (line 15) and repeat the process until the equality is true.

With *N* available, we can now compute *D* using the function `isYdreach` in Algorithm 2.

Table 5 Algorithm 2: `isYdreach`



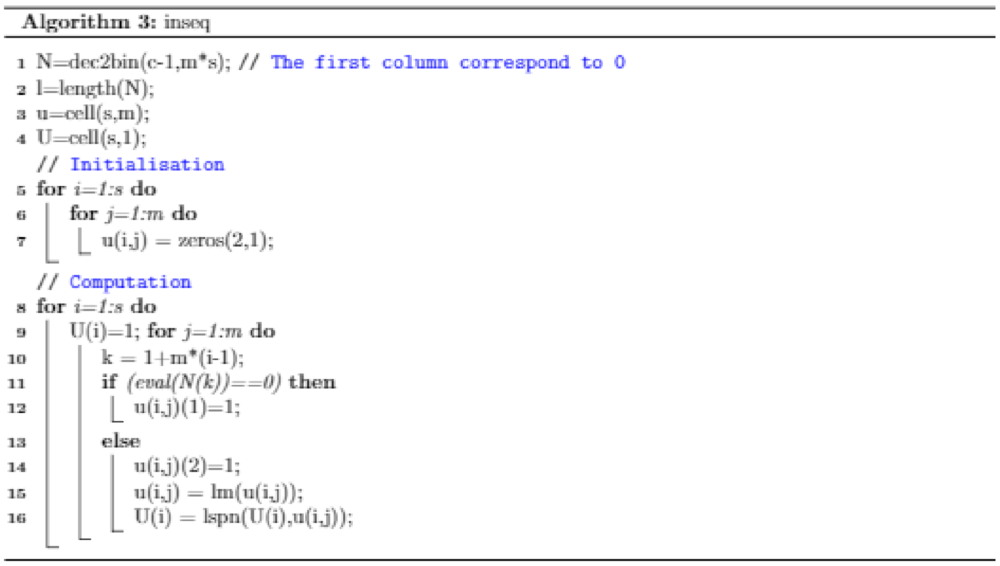
```
Algorithm 2: isYdreach

1  etemp=0;
2  s=zeros(smax,1);
3  stemp=zeros(smax,1);
4  lmax=length(Ysetlength(Yset).v); // lmax is the maximum number of columns of Yset ie the
       maximum number of path
5  co=zeros(smax,lmax);
6  for i=1:smax do
7      for j=1:length(Yset(i).v) do
8          etemp = etemp+(Yset(i)(j)==yd);
9          if Yset(i)(j)==yd then
10             co(i,j) = 1; stemp(i) = stemp(i)+1;
11     e=stemp;
12     s=stemp;
```

The function `isYdreach` takes as parameters the set of reachable output *Yset*, the desired output y_d and the number after which no new states are obtained *N* and returns *D*. *e* is a Boolean variable that indicates if y_d is reachable. If not, D will be an empty matrix. s is here a vector of size N that indicates at which steps y_d is reachable among the N steps considered.

Finally, the function `inseq` elucidates the algorithm described in Chapter 4.3.2 (Algorithm 3).

Table 6 Algorithm 3: `inseq` (algorithm to compute the input sequences)



```
Algorithm 3: inseq
1  N=dec2bin(c-1,m*s); // The first column correspond to 0
2  l=length(N);
3  u=cell(s,m);
4  U=cell(s,1);
   // Initialisation
5  for i=1:s do
6      for j=1:m do
7          u(i,j) = zeros(2,1);
   // Computation
8  for i=1:s do
9      U(i)=1; for j=1:m do
10         k = 1+m*(i-1);
11         if (eval(N(k))==0) then
12             u(i,j)(1)=1;
13         else
14             u(i,j)(2)=1;
15             u(i,j) = lm(u(i,j));
16             U(i) = lspn(U(i),u(i,j));
```

`inseq` takes as parameters satisfying column and row number of D, c and s, and the number of input m. It returns both, the overall input that allows us to achieve the control objective and the corresponding input sequence. The input sequence is composed of s times m vectors u\{i,j\} (line 3) that we initialize as null vectors (line 8) (which temporary breaks the assertion of Remark 1, stating that all the vectors we work with must contain exactly one 1). We then put the 1s at the right location (line 12 and 14) with respect to (38) and (39).

In the next Chapter, we use the above described function to perform a controllability analysis and a control strategy design. The analysis presented in Chapter 6 was run on desktop with a 12-core processor Intel Xeon, at 3.4 GHz, with 32 GB of virtual memory on Windows. The results for the analysis of a network containing 13 nodes including 2 inputs and 3 outputs were obtained in less than 2 minutes.

6.0 CASE STUDY 1 : TOY EXAMPLE

In this section, we apply the theory developed in Chapter 3 on an example illustrated in Figure 2.

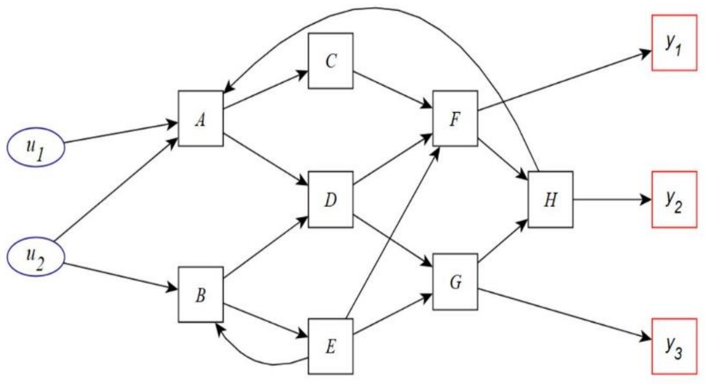


Figure 2. Example of a Boolean control network with two inputs, u_1 and u_2, five internal nodes, *A*, *B*, *C*, *D*, *E*, *F*, *G* and *H* and three outputs, y_1, y_2 and $\boldsymbol{y_3}$.

Table 7 completes Figure 2 by indicating update rules for every component of the network, except u_1 and u_2, which are inputs of the system, and therefore, assumed to be designable. In other words, we consider that at each time step the values of u_1 and u_2 can be set to any value.

Table 7. Update rules of the nodes of the network in Fig.2.

Internal nodes	Rule	Output nodes	Rule
$A*$	$H + u_1 + u_2$	y_1	F
$B*$	$u_2 + \bar{E}$	y_2	H
$C*$	$\bar{A}$	y_3	G
$D*$	$A \cdot B$		
$E*$	B		
$F*$	$C \cdot D + \bar{E}$		
$G*$	$D + \bar{E}$		
$H*$	$F \cdot G$		

Although in this example we use generic names for the network elements, many intra-cellular pathways could be represented using this (or similar) BN. We can identify three main pathways in this model. The first one is the pathway $u_1 \rightarrow A \rightarrow C \rightarrow F \rightarrow y_1$ that leads to the first output. The second one is the pathway $u_2 \rightarrow B \rightarrow E \rightarrow G \rightarrow y_3$ leading to the third output. The last pathway is a mix of the first and the second pathway $(u_1, u_2) \rightarrow (A, B) \rightarrow D \rightarrow (F, G) \rightarrow H \rightarrow y_2$.

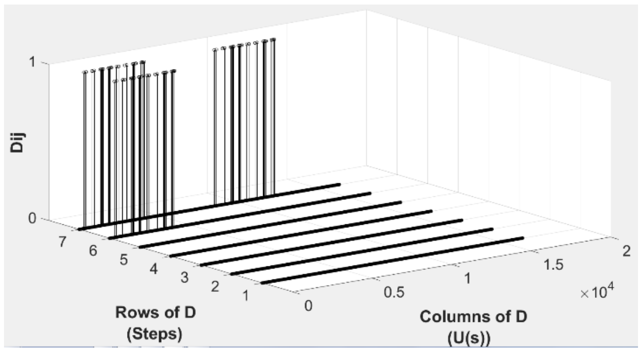
The model also contains two feedbacks loops $A \rightarrow (C, D) \rightarrow F \rightarrow H \rightarrow A$ and $B \rightarrow E \rightarrow B$.

6.1.1 Control objective 1: Disease

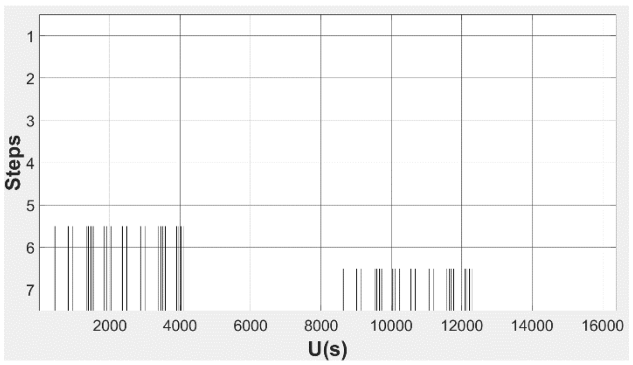
We assume in this case that the three outputs are known as markers of a disease that we would like to inhibit, and the inputs are potential treatments. We want to see if, starting from a state where all outputs are activated, we can drive them to a state where they are all low. In other words, we are interested in finding an input sequence that leads to y_1=y_2=y_3=0. Thus, we follow all the steps listed in Chapter 4, in order to create the Design Matrix *D*. We use a 3D plot to visualize elements of matrix *D*, as shown in Figure 3(a).

The plot in Figure 3(a) is a discrete 3D plot, where the *xy*-plane is defined by the matrix dimensions. We recall that each column of *D* corresponds to a different input sequence. The rows of *D* relate to the time steps. We stopped at step 7 since there are no new states that can be reached after this step. The *z*-axis contains the matrix elements values. Since these values are either 0 or 1, we used a discrete plot. The thick black lines at rows 1 through 5 indicate 0 values. Each 0 value in Figure 3(a) represents either an actual 0, i.e., the output corresponding to the particular input sequence *l*, for a given column of matrix *D*, particular time step *s*, for a given row of *D*, is not the desired output, or it represents an undefined, x value, as described in Chapter 3.

In Figure 3(b), a value 1 at step *s* obtained by an input sequence *l* is represented by a "square" of width 1 whose center is the point (*s*, *l*). Since the number of columns of *D* is usually much greater than the number of rows, the squares are compressed and resemble sticks. The darker the sticks are, the closer the squares are. From the Design Matrix, we can now identify an input sequence that leads to the desired output. One example of such input sequence corresponds to row 6 and column 305 of *D*. Then we apply (38) and (39) to derive the input sequence.



(a)



(b)

Figure 3. Two representations of the Design Matrix for the first control objective, y_1=y_2=y_3=0. (a) 3D representation and (b) 2D "heatmap".

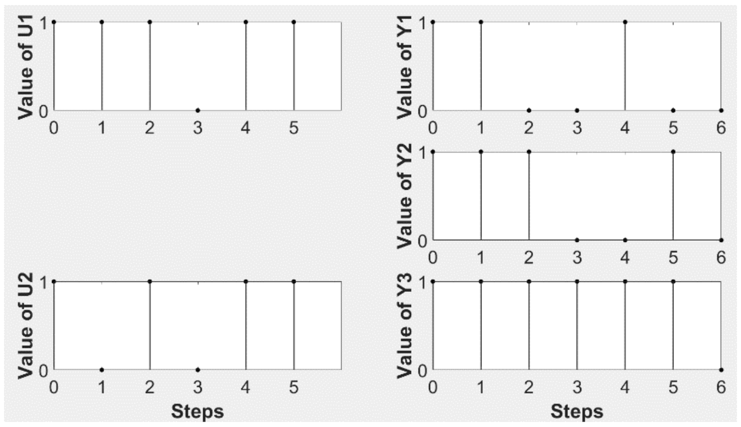
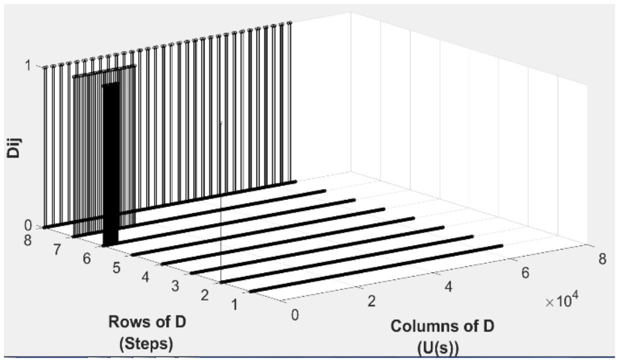


Figure 4. Input and Output sequences for the first control objective

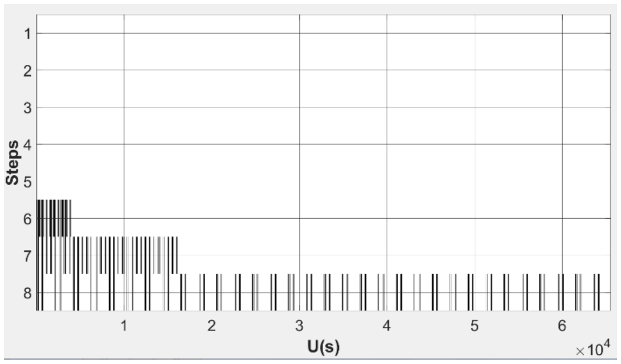
The derived input sequence, and the corresponding output are shown in Figure 4, where the sequences are plotted as stems, and the values of the nodes are indicated by filled circles similarly to Figure 3. However, unlike Figure 3, where circles at 0s and 1s represent presence or absence of elements in a set, in Figure 4 these circles illustrate values of model elements in consecutive time steps.

6.1.2 Control objective 2: Immune response

We assume here that the output are antigens that we want to promote. We want to see if starting from a point where the signaling pathway is off, i.e., all the nodes are low, we can reach a state where all the outputs are high, that is, y_1=y_2=y_3=1. As in Chapter 6.1, we followed the procedure developed in Chapter 3. The results for this case are presented in Figure 5 and Figure 6.



(a)



(b)

Figure 5. Two representations of the Design M**a**trix, for the second control objective, y_1=y_2=y_3=1. (a) 3D representation and (b) 2D "heatmap".

We can notice in Figure 5(b) that for some input sequences, the output will not be maintained at the desired state for the following steps, although the inputs are maintained at the last value they had at the step before reaching the desired state. To stabilize the outputs at the desired state, we will need to continuously update the inputs.

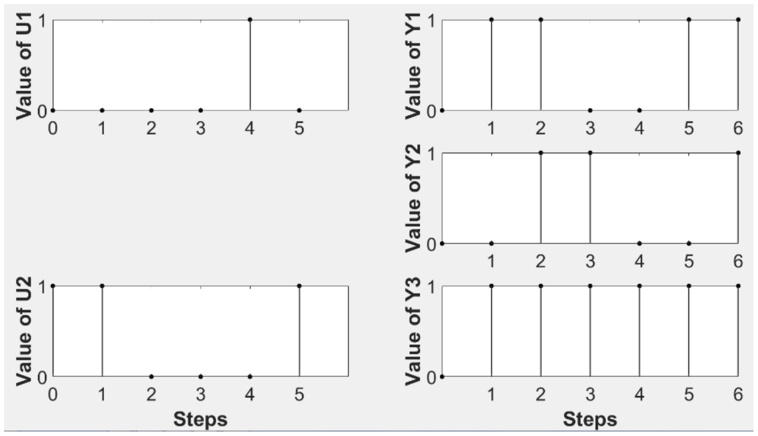


Figure 6. Input and output sequences for the second control objective.

In both cases, we achieved the control objectives, explicitly, in the first case (disease), drive all the output to a low level starting from a state where all the elements were at a high level. In the second case, drive all the outputs to a high level starting from a state where all the elements were at a low level.

7.0 LIMITATIONS OF METHODS BASED ON THE ASSR

The approach that we developed provides a powerful way to drive nodes to a desirable state, and it gives us some control power in the cyclic networks, often encountered in biology. One can infer the influence of inputs on outputs without having to simulate the evolution of the network for a big number of steps and with randomly assigned inputs. Assuming a simultaneous update scheme, that is, all elements updating their values simultaneously, one can find the input values that will give a desired output. However, the simultaneous update scheme does not always reflect the variety of time scales in biological systems [5][6][30][34]. Thereby, an extension of this method to the stochastic, random-order sequential update scheme, is necessary. One way to do so, would be to introduce a random matrix in (15) and (16) that acts as a variable-update-selector. Additionally, this method does not give information about the stability of the desired states. A further development of this method would be necessary to get insights about the design of inputs subsequent to the step where the control objective was achieved.

8.0 BINARY DECISIONS DIAGRAMS

8.1 DEFINITION

A binary decision diagram represents a Boolean function as a rooted, directed acyclic graph [26][27]. It is formed by two types of elements, terminal and non-terminal nodes. The former represent the Boolean variables present in the function, and the latter are Boolean constants whose values are either 0 or 1. Each non-terminal node *A*, possesses two outgoing edges, *low* and *high* [41], labeled with respect to the two possible values for the variable associated with that node. The two edges point to terminal or non-terminal nodes called successors, or equivalently, children of *A*. When reading a BDD node from the root at the top of the graph, to the terminal node at the bottom of the graph, the non-terminal node must be read as if the variable is high, then he node pointed at with the *high* edge is examined next, else, if the variable is low, the node pointed at with the *low* edge is followed. Thus, a BDD can be seen as cascaded if-then-else statements. As an example, Figure 7 illustrates a representation of the function $f(x_1, x_2, x_3)$ defined by the truth table given in Figure 7(left).

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

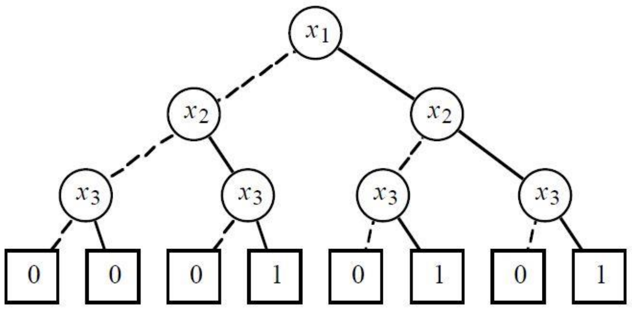


Figure 7. Truth Table and Decision Tree Representations of a Boolean Function. A dashed (solid) tree branch denotes the case where the decision variable is 0 (1).

For a given assignment to the variables, the value of the function is determined by tracing a path from the root to a terminal vertex, following the branches indicated by the values assigned to the variables. The function value is then given by the terminal vertex label. Due to the way the branches are ordered in this figure, the values of the terminal vertices, read from left to right, match those in the truth table, read from top to bottom. However, the BDD presented in Figure 7 is not optimal as it is an exact translation of the truth table. For example, in Figure 7(left), the first row of the truth table is equivalent to the leftmost path of the BDD on the right, as shown in Figure 8.

x_1	x_2	x_3	f
0	0	0	0

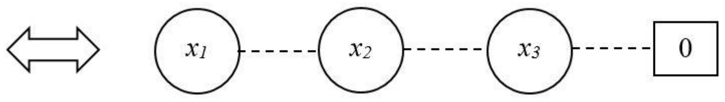


Figure 8. Truth table BDD equivalence

We can make comparisons like the one in Figure 8 for the whole table, as the second row is equivalent to the second path on the BDD (starting from the left, *low-low-high*) and the last row is equivalent to the rightmost path (*high-high-high*).

8.2 ORDERING AND REDUCING

Binary Decisions Diagrams become a very efficient data structure to represent Boolean functions when they are ordered and reduced. A BDD is ordered if on all paths through the graph the variables appear in the same order. The BDD in Figure 7 is actually ordered as the variables always appear in the following order $x_1 \rightarrow\ x_2 \rightarrow x_3$ in all paths. Additionally, if no redundant tests are conducted and if all nodes are unique, the BDD is said to be reduced. A node *A* is unique if no other node associated with the same variable, possesses the same *low* and *high* children than *A*. In Figure 9, we show the process of reducing a BDD.

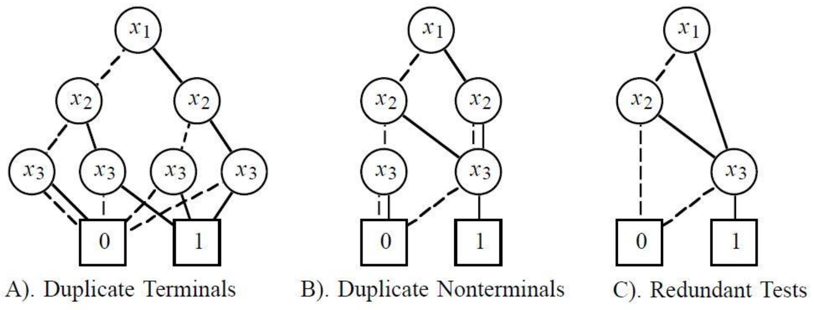


Figure 9. Reducing a BDD

Figure 9 shows the reduction of the BDD shown in Figure 7. We first remove the redundant terminal nodes in A). Three nodes associated with x_3 had the same *low* children (terminal node associated with constant 0) and same *high* children (terminal node associated with constant 1), we then removed the redundant nodes in B). Finally, we removed the unnecessary tests.

The BDDs that are ordered and reduced are referred to as Reduced Ordered Binary Decisions Diagrams (ROBDD). In the literature [25][26][27][28][29][30][31], scientists usually refer to ROBDD using the acronym BDD. Likewise, in the remainder of this thesis, we will be referring to ROBDDs as BDDs.

9.0 BOOLEAN NETWORK VIA BINARY DECISION DIAGRAMS

In this chapter we will introduce the representation of a Boolean network as a set of BDDs. Hereafter, we will describe a Boolean network with *n* nodes as follows,

$$\begin{cases} x_1^* = f_1(x_1, x_2, \dots, x_n) \\ x_2^* = f_2(x_1, x_2, \dots, x_n) \\ \quad\vdots \\ x_n^* = f_n(x_1, x_2, \dots, x_n) \end{cases} \tag{40}$$

where the " * " indicates the value that a node *i* will take the next time it is chosen for update. Among the x_i, some nodes are inputs and some other are outputs of the BN. We call I the set of integers from 1 to *n* and M, Y and P the set of the indices of the input, the internal and the output nodes, respectively.

Remark 3 For all *i* in I, x_i* is a function of the x_j with *j* in a subset J of I. Although all the x_i appear as arguments in all the f_i, each x_i is actually a function of just a subset of the set of the x_i.

With BDDs, the notions of Boolean functions and Boolean variables merge. Indeed, a variable is a BDD containing exactly three nodes, a node labeled with the variable name whose *low* and *high* edges point to the constant terminal nodes 0 and 1.

We can present Boolean networks as a collection of *2·n* BDDs as shown in Figure 10. Among those *2·n* BDD, *n* BDDs are used to store the Boolean variables $(x_1, x_2, \dots, x_n)$ and *n* others are used to store the Boolean functions $(x_1^*, x_2^*, \dots, x_n^*)$.

$$\begin{cases} x_1^* = f_1(x_1, x_2, \dots, x_n) \\ x_2^* = f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ x_n^* = f_n(x_1, x_2, \dots, x_n) \end{cases}$$

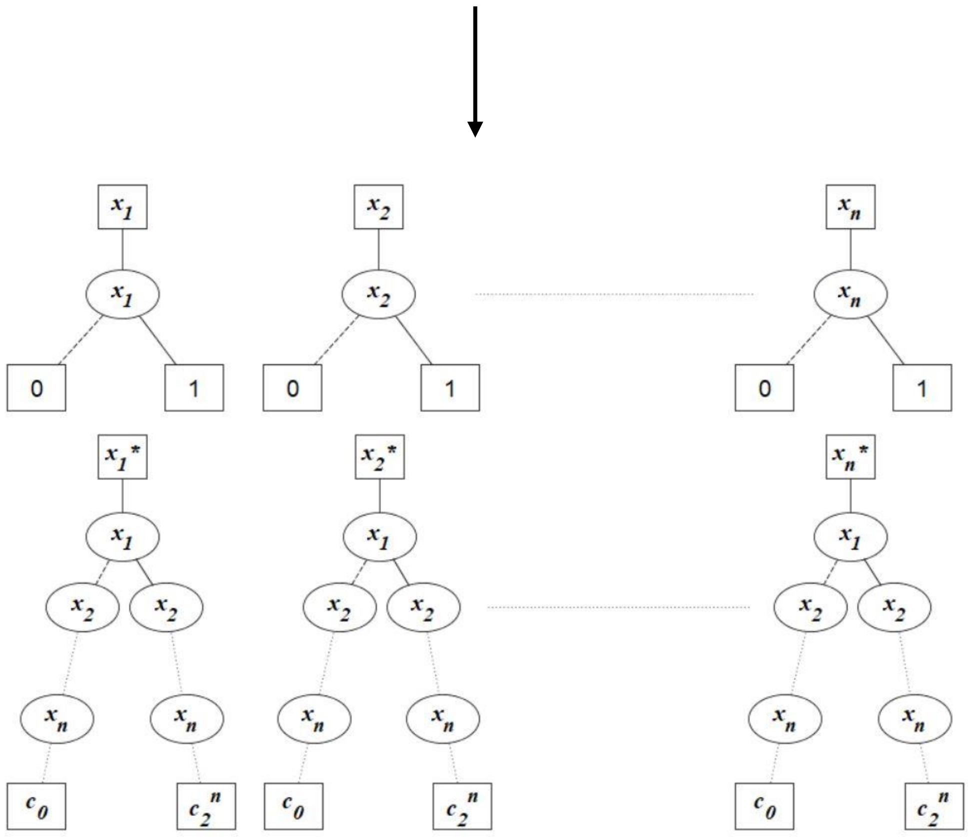


Figure 10. Conversion of a BN to a set of BDD

Figure 10 illustrates the conversion of a Boolean network as a set of BDDs. The first row of a BDD contains the variables $x_1, x_2, \dots, x_n$, and the second row of a BDD contains the Boolean functions $x_1^*, x_2^*, \dots, x_n^*$. In Figure 10, for the second row of a BDD, to keep the Boolean expression general, we showed general non-reduced BDD and used Boolean constants c_i without specifying their values. We also willingly omit to specify the nature of the edges of the nodes labeled , x_2 and

x_n which is why we use dotted lines for the edges (different from the dashed lines used for the *low* edges).

10.0 UNROLLING LOGIC

We now explain how to relate inputs of a BN to the outputs of that BN.

10.1 METHOD

10.1.1 Simultaenous update scheme

In the simultaneous update scheme, the dynamics of a BN is deterministic. We can predict the evolution of the BN deterministically, precisely compute the steady states, and see the exact effect of the inputs on the outputs. If there is a path between an input x_i and an output x_j, we can find a Boolean expression relating a future value of x_j, $x_j^{*^s}$ and x_i . $x_j^{*^s}$ indicates the expression of node j at step s .This can be done by recursively calling the different Boolean functions according to (41-42).

$$x_j^{**} = f_j(x_1^*, x_2^*, \cdots, x_n^*) \tag{41}$$

$$x_j^{**} = f_j\big(f_1(x_1, x_2, \cdots, x_n), f_2(x_1, x_2, \cdots, x_n), \cdots, f_n(x_1, x_2, \cdots, x_n)\big) \tag{42}$$

This process is further illustrated in Table 5 and Figure 11, where we show how to express the output *y* with respect to the input *u* in Figure 1, with the difference that we now consider that the output is delayed of one step.

Table 8. Unrolling procedure applied to the example from **Figure 1**.

Steps	Expression of *y*
1	$y^* = C$ (43)
2	$y^{**} = C^* = A + C$ (44)
3	$y^{*^3} = A^* + C^* = B \cdot C + u + \bar{B}$ (45)

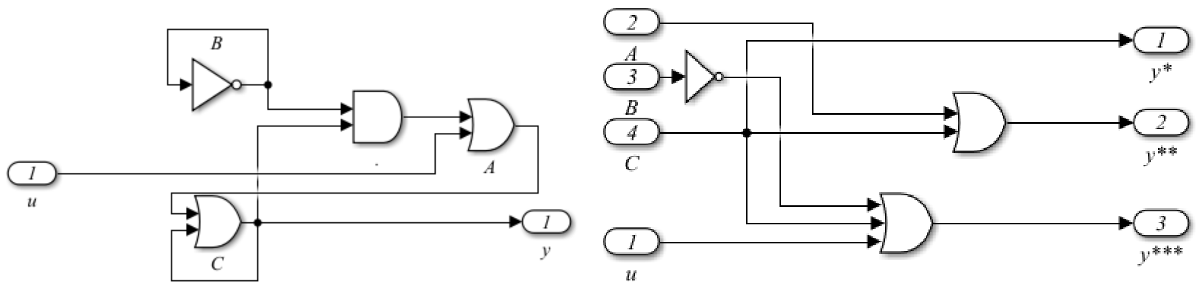


Figure 11. Unrolling process applied to the example from **Figure 1**.

The procedure shown in Figure 11 allows us to transform a nonlinear network Figure 11(left) into a linear network Figure 11(right), and to link the outputs to the inputs. The expression for y^{*^3} in (45) can be further simplified into

$$y^{*^3} = \bar{B} + C + u \tag{46}$$

We observe that the effect of the input *u* is seen in *y* only three steps later. Indeed, activating *u* at a given step *s* will result in *y* being high at step *s*+3. If we keep recursively, unrolling *y* we obtain the following result

$$y^{*^4} = A + B + C + u^* \tag{47}$$

And, by mathematical induction we can prove that,

$$y^{*^s} = f_y(A, B, C, s)_{\bar{u}} + u^{s-3} \tag{48}$$

Where $f_y(A, B, C, s)$ is the part of the expression of y^{*^s} that does not depend on u. The superscript $\bar{u}$ next to f_y is used to emphasize that f_y does not depend on u. From (48), we can conclude that if u is maintained at a high value, then y will, after a transient phase of three steps, stabilize to a high value.

We showed through this example, how to get the expressions of an output x_j of a network as function of an input x_i, under the assumption that there exists a path between x_j and x_i. The process also works to relate multiple outputs to multiple inputs. However, the example in Figure 11, is sufficiently small to be done manually. For bigger networks, this process is tidious, thus we implemented a code (description provided in Chapter 9.2) to automatically perform this operation.

10.1.2 Random-order sequential update scheme

In the random-order update scheme, the dynamics of a BN is stochastic, we cannot exactly, determine the effect of the inputs on the outputs. If there is a path between an input x_i and an output x_j, we can, similar to the simultaneous update scheme, find a Boolean expression relating a future value of x_j, $x_j^{*^s}$ and x_i.

However, in this case, the expression is associated to a weight that corresponds to a given sequence of update (permutation). In the random-order update scheme, each node x_i has a probability p_i to be updated at a given step. Thus, $x_j^{*^s}$ is associated with a probability

$$p_j^{*^s} = \prod_I p_i \tag{49}$$

where I is a given sequence of update.

To illustrate this update scheme, let us consider a network with three nodes where each node has a specific probability to be selected for update: $p_1 = \frac{1}{2}, p_2 = \frac{1}{3}, p_3 = \frac{1}{6}$.

$$\begin{cases} x_1^* = f_1(x_1, x_2, x_3),\ p_1 = \dfrac{1}{2} \\ x_2^* = f_2(x_1, x_2, x_3),\ p_2 = \dfrac{1}{3} \\ x_3^* = f_3(x_1, x_2, x_3),\ p_3 = \dfrac{1}{6} \end{cases} \tag{50}$$

Then the expression for $x_1^{*^2}$ will depend on the sequence of update as shown in (51)

$$x_1^{*^2} = \begin{cases} f_1(f_1(x_1, x_2, x_3), x_2, x_3),\ p_1^{*^2} = \dfrac{1}{2} \cdot \dfrac{1}{2} = \dfrac{1}{4} \\ f_1(x_1, f_2(x_1, x_2, x_3), x_3),\ p_1^{*^2} = \dfrac{1}{2} \cdot \dfrac{1}{3} = \dfrac{1}{6} \\ f_1\big(x_1, x_2, f_3(x_1, x_2, x_3)\big),\ p_1^{*^2} = \dfrac{1}{2} \cdot \dfrac{1}{6} = \dfrac{1}{12} \end{cases} \tag{51}$$

Similarly, we can get expressions for $x_2^{*^2}$, $x_3^{*^2}$ and their counterparts at further steps.

10.2 DATA STRUCTURES AND IMPLEMENTATION

In order to automate the method described previously in Chapter 9, we developed a library to manipulate BNs. The foundations of our library rests on the CUDD package [28], which is publicly available at https://github.com/ivmai/cudd .

The CUDD library provides functions to manipulate BDDs and other Decisions Diagrams such as Algebraic Decisions Diagrams, which are not used in the context of this thesis, but can be used to model biological systems as well. The functions and the data structures implemented in the CUDD library are implemented in C but the authors also provided a C++ wrapping.

10.2.1 Data structures

Our library, developed in C++, is structured mainly around two main classes. The first class is called *BnetNode*. This class is basically an extension of the BDD object of the CUDD package.

A BnetNode (BNN) object contains, a CUDD (class) object the attribute *m_mgr*, a BDD object *m_node*, a string *m_nodename*, and two integers that indicate the type of the BNN *m_nodeTimeType* and *m_nodeType*.

m_nodeTimeType is an enumeration (*present, future*) that specifies if the BDD of the BNN object is a variable (*present*) like $x_1, x_2, \ldots, x_n$ in the canonical example of Fig.10 or an update rule (*future*) $x_1^*, x_2^*, \ldots, x_n^*$*.

m_nodeType is an enumeration (*undefined, input, intermediate, output*) used only if the BNN is of time type *present* and indicates the role that the variable x_i associated to the BNN plays in the Boolean network.

m_nodeName is a string that obviously contains the name of the BNN. The names of the BNN must respect this convention:

- Every BNN in a BN must have a unique name
- The name of a BNN of time type *future* must be the same name than the corresponding BNN of time type *present* followed by the character '*'.

Under those condition, the name of a BNN can be used as an ID number to identify each instance of a BNN object.

m_node is a BDD that represents a Boolean variable if the BNN has a time type *present* and a Boolean function if the BNN has a time type *future*.

m_mgr is a CUDD object called manager that is necessary to manipulate BDD. It is important to verify that every BNN in a BN have the same manager.

The *BnetNode* class is then used to define the class Bnet in a *has-a* relationship. A Bnet (BN) object contains, an array of BNNs and six arrays of integers, *m_BNNlist m_BNNlist_n*, *m_BNNlist_r*, *m_BNNlist_m*, *m_BNNlist_y*, *m_BNNlist_p* which gather all the element that compose the Boolean Network (we may subsequently refer to those arrays as lists) and six integers which are actually the size of the corresponding list.

m_BNNlist, is a dynamical array that contains all the BNNs of the BN that is $x_1, x_2, \dots, x_n, x_1^*, x_2^*, \dots, x_n^*$.

m_BNNlist_n is the list of the indices of the elements of *m_BNNlist* whose field *m_nodeTimeType* are set to *present*.

m_BNNlist_r is the list of the indices of the elements of *m_BNNlist* whose field *m_nodeTimeType* are set to *future*.

m_BNNlist_m is the list of the indices of the elements of *m_BNNlist* whose field *m_nodeType* are set to *input*.

m_BNNlist_y is the list of the indices of the elements of *m_BNNlist* whose field *m_nodeTimeType* are set to *intermediate*.

m_BNNlist_p is the list of the indices of the elements of *m_BNNlist* whose field *m_nodeTimeType* are set to *output*.

10.2.2 Unrolling Implementation

The unrolling function in the simultaneous update scheme case is described by a flowchart in Figure 12.

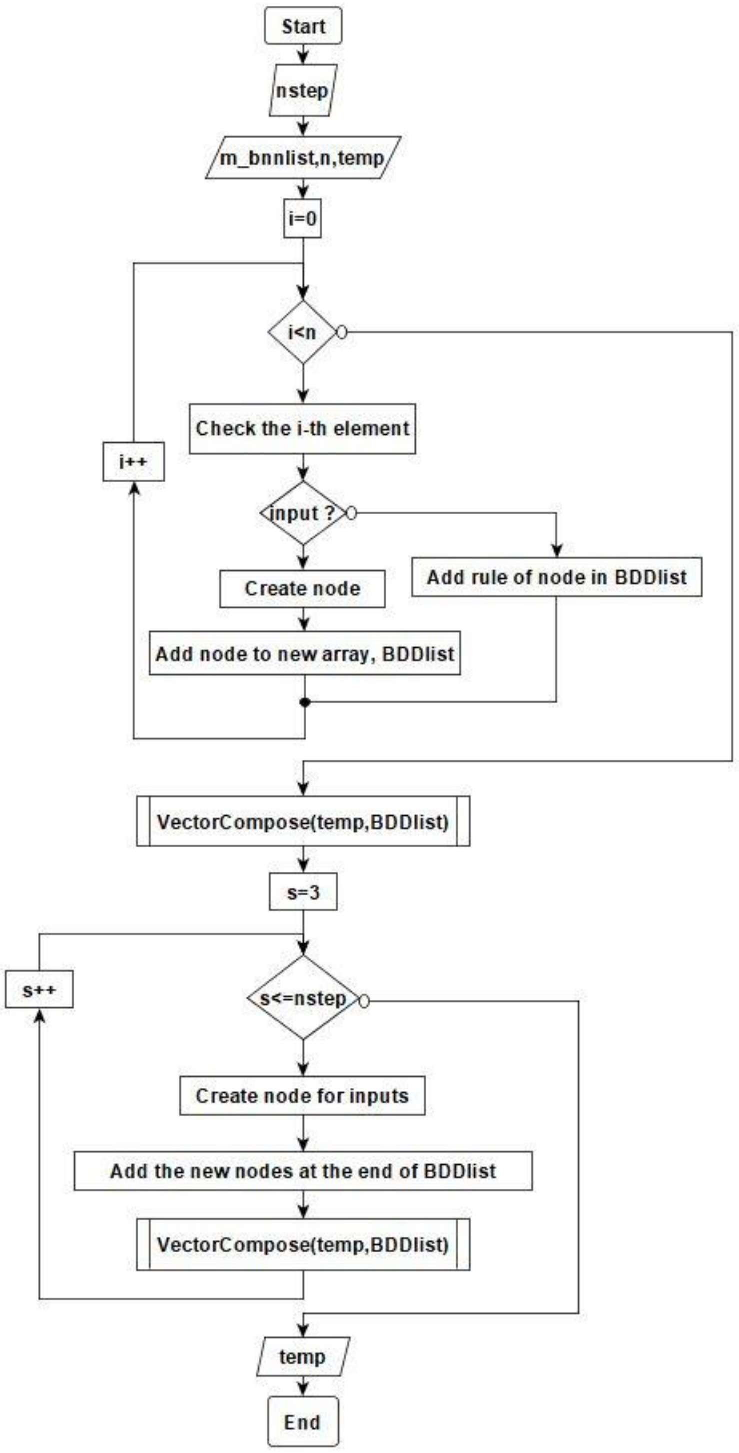

Start
nstep
m_bnnlist,n,temp
i=0
i<n
i++
Check the i-th element
input ?
Create node
Add rule of node in BDDlist
Add node to new array, BDDlist
VectorCompose(temp,BDDlist)
s=3
s++
s<=nstep
Create node for inputs
Add the new nodes at the end of BDDlist
VectorCompose(temp,BDDlist)
temp
End

Figure 12 Flowchart of Unroll function in the case of the simultaneous update scheme.

The Unroll function is applied on a BN object and it takes as parameters the position of the node we want to unroll in the array of BNN object (*m_BNNlist*) that is composed of the other BN objects, it is represented in the flowchart by the variable *temp*. More precisely, *temp* is the BDD of the node that we want to expand. Moreover, *nstep,* is the step number at which we want the expression of the node. *VectorCompose* is a precious function provided in the CUDD library. The function takes as input arguments a BDD (first argument) and an array of BDDs (second argument). Then, it replaces every variable in the support of the first argument by the corresponding BDD in the array of the second argument. An important point about the operation principle of *VectorCompose* needs to be mentioned. Each BDD has an index that matches the order of creation of the BDDs. *VectorCompose* matches a variable of index *i* in the support of the first argument by the *i*-th BDD in the array of the second argument. The first loop is an initialization loop. We fill BDDlist with the corresponding rule of each node that is not an input since inputs do not have a rule. Instead, for them, we create a new node that represents the input at the following step. Then we call the function *VectorCompose*.

In the second loop we repeat the process of the first loop, and while we do not need to add the rules in BDDlist anymore, we still need to create new variables for the inputs.

The unrolling function in the random-order sequential case is now described in Figure 13.

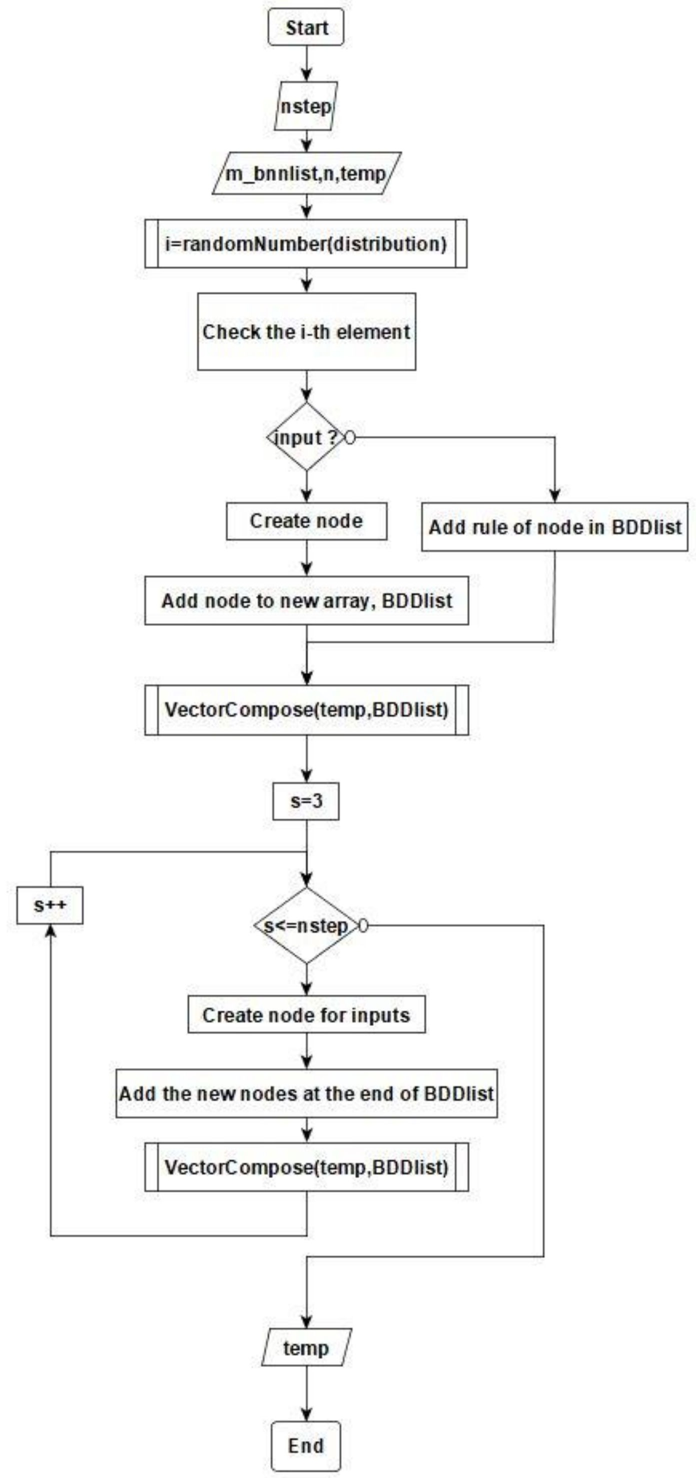

Start
nstep
m_bnnlist,n,temp
i=randomNumber(distribution)
Check the i-th element
input ?
Create node
Add rule of node in BDDlist
Add node to new array, BDDlist
VectorCompose(temp,BDDlist)
s=3
s++
s<=nstep
Create node for inputs
Add the new nodes at the end of BDDlist
VectorCompose(temp,BDDlist)
temp
End

Figure 13 Flowchart of Unroll function in the case of random order sequential update scheme

The procedure in the random-order case is similar to the one of the simultaenous case. However, instead of replacing every node by its corresponding rule (or by a new node for an input), we first generate a random number i, and then we replace only the i-th variable.

10.3 PROSPECTIVE ANALYSIS

With the unrolling approach, nonlinear (cyclic) networks are converted into linear networks. Yet, for linear networks, we have tools to study their controllability and observability [42][43].

Among those tools are the “don’t cares”. In digital logic, a don't-care term for a function is either an illegal combination of inputs or a combination of inputs for which the function output does not matter. Generally speaking, the “don’t cares” by pointing out the combination of inputs that do not affect the outputs can provide information about the combination of inputs that do affect the outputs. Before we define the different type of “don’t cares” we need to define a few operators that are at the origin of the definition of the different “don’t cares”.

The first operator is the Shannon expansion of a Boolean function. Given the Boolean variables $x_1, x_2, \dots, x_n$ and any Boolean function $F(x_1, x_2, \dots, x_n)$, F can be expressed as,

$$F = x_i \cdot F_{x_i} + \overline{x_i} \cdot F_{\overline{x_i}} \text{ for any } x_i$$

where $F_{x_i}$and $F_{\overline{x_i}}$ are defined as,

$$F_{x_i} = F(x_1, \dots x_i = 1, \dots x_n) \tag{52}$$

$$F_{\overline{x_i}} = F(x_1, \dots x_i = 0, \dots x_n) \tag{53}$$

are called cofactors of F with respect to x_i and $\overline{x_i}$.

The second operator is the Boolean difference. The Boolean difference of a function F with respect to x_i is defined by,

$$\frac{\partial F}{\partial x_i} = F_{x_i} \oplus F_{\overline{x_i}} \tag{54}$$

where $\oplus$ represents the logical operator XOR.

Derivatives of function of continuous real variables indicate how the function changes with small change of the variables and the Boolean difference has the same role for Boolean function. It indicates how the function changes when a variable switches from 0 to 1 or from 1 to 0.

The next operator, the existential quantification is defined by replacing the operator XOR in (54) by the operator OR.

$$(\exists x_i\ F)(x_1, \dots x_{i-1}, x_{i+1}, \dots x_n) = F_{x_i} + F_{\overline{x_i}} \tag{55}$$

The existential quantification gives the closest function to *F* independent of the variable *xi* by adding some terms.

Finally, we define the last operator, the universal quantification by replacing the XOR in (54) by an AND.

$$(\forall x_i\ \ F)(x_1, \dots x_{i-1}, x_{i+1}, \dots x_n) = F_{x_i} \cdot F_{\overline{x_i}} \tag{56}$$

The universal quantification gives the closest function to *F* independent of the variable *xi* by removing the terms which depend on x_i.

Using the operators described above, we can now define the satisfiability don't cares (SDC), controllability don't cares (CDC) and observability don't cares (ODC).

Satisfiability don't cares are patterns of inputs and outputs of an internal node which are impossible because they contradict the value of the node output. Formally, for a node $X = F(x_1, x_2, \dots, x_n),)$, we have

$$SDC_X = X \oplus F \tag{57}$$

Controllability don't cares are impossible patterns of just inputs of an internal node. They are obtained by removing the output of the SDCs.

$$CDC_X = (\forall x_j)\left(\sum_i SDC_{x_i}\right) \tag{58}$$

x_j∈[variables which are not input to X], x_i∈[variables which are input to X]

Observability don’t cares are patterns of input of an internal node that make the node output unobservable at the network output. In other words, we can compute the network value without knowing the node output value and that a change in this node output value does not affect the network output.

$$ODC_X = \overline{\left(\frac{\partial Q}{\partial X}\right)} = \overline{(Q_X \oplus Q_{\bar{X}})} \tag{59}$$

In Section 10.1, we showed how to get an expression of the outputs of a cyclic Boolean network as function of the inputs. It is straightforward to apply the operators defined in (57-59) to the expanded expressions obtained through the methodology of Section 10.1. This analysis will be developed as an extension of the thesis in the future.

11.0 CASE STUDY 2: BIOLOGICAL EXAMPLE

11.1 T-CELL MODEL

T cells are a specific cell type in humans and that play a key role in the immune system [33]. The T refers to the thymus, organ in which the T cells mature and are released into the body to recognize and fight against infections and other threats to the immune system, also known as pathogens that, are responsible for activating and differentiating the T cell. After differentiation, Tcells they can be grouped into two main subtypes: those that mediate the immune response [T helper (T_H) cells] and those that suppress the immune response, regulatory T (T_{reg}) cells [32] [34]. It has been shown previously that both the concentration of antigen and the duration of stimulation of T cells with antigen influence the differentiation of T cell. Thus, increased antigen concentrations favor T_H cell generation, whereas decreased concentrations favor T_{reg} cell generation [35][36][37].

In [32]], Miskov-Zivanov et al provided a logical model of the T cell signaling pathway and here we include the illustration of this model in Appendix C. The model contains about 40 elements among which are the transcription factor of the T_{reg} cells (FOXP3), the T cell receptor (TCR), the costimulatory molecule (CD28), the interleukin 2 (IL-2) and interleukin 2 receptor (IL-2R), the transforming growth factor β (TGF- β) and transforming growth factor receptor β (TGF-βR). We highlight the presence of those particular elements among the 40 of the networks as in a control engineering point of view, they would be the inputs and outputs of the systems. Thus, our goal in this case study is to apply the methodology described in Chapter 9 and 10 to this biological example. Namely, we want to express the outputs of the T-Cell model as functions of the inputs of the model and to analyze the expressions obtained. The results of this approach are presented in the next section.

11.2 RESULTS

In this Chapter, we present the results of our unrolling method applied to the T-Cell signaling pathway described in Section 11.1.

We converted the model as a collection of 107 BDDs. Among those 107 BDDs, 61 were variables and 46 were Boolean rules. Given the size of the network, we are not able to show it entirely but Figure 4 and Figure15 give a partial representation of the network as a collection of BDDs.

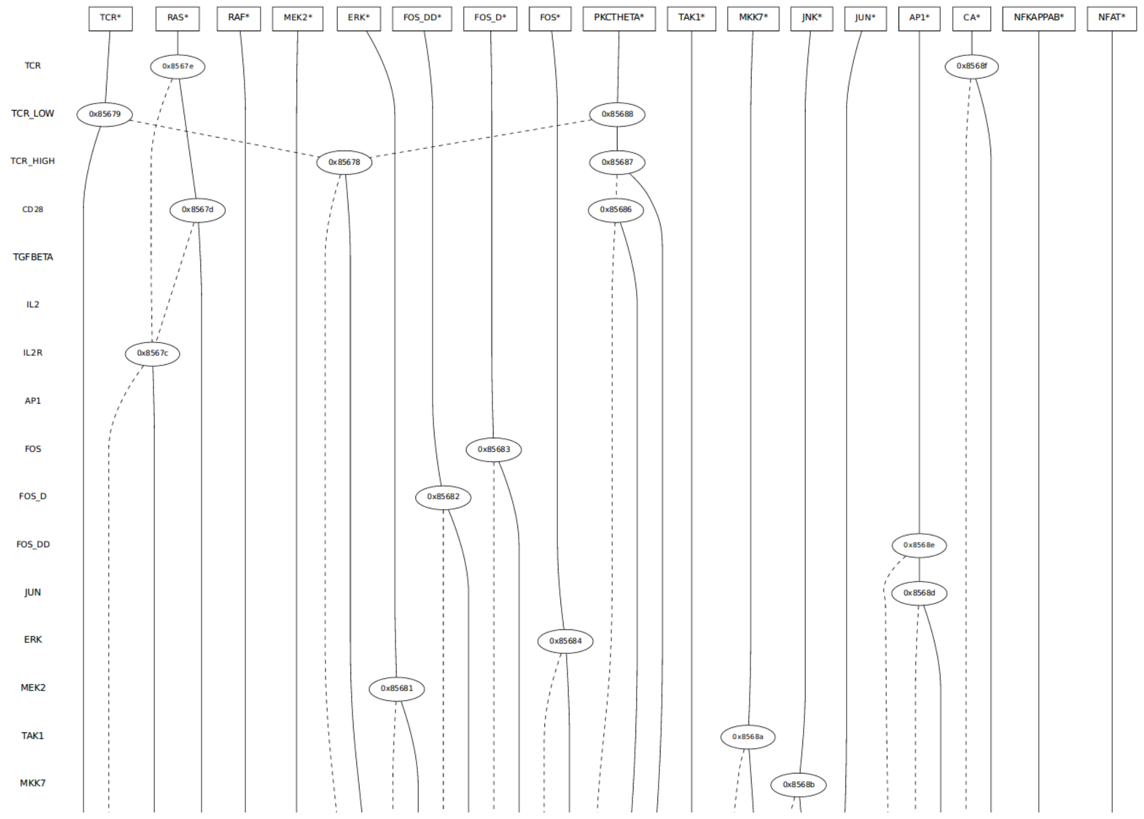


Figure 14 T-Cell model with BDD (Top partial view)

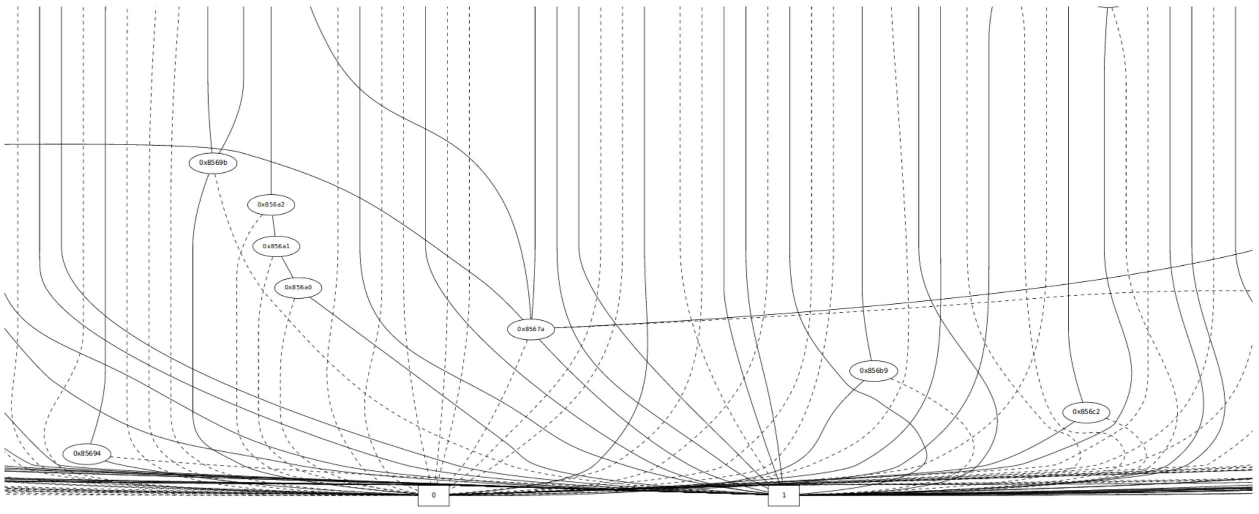


Figure 15 T-Cell model with BDD (Bottom partial view)

In Figure 14, the variables names are listed on the left, the update rules names are listed on top; the rules are named after the variable that they are associated with. The BDDs for the 46 rules are merged into one supra-BDD but we can alternatively, display each rule individually as shown in Figure 16. The bottom part of a graphical representation of a BDD provides essential information as it contains the terminal nodes which show the value associated with each combination of inputs. Under this consideration and because the BDD in Figure 14 is not complete, Fig.15 was inserted to show the bottom part of the supra-BDD representing the T-Cell model with the purpose of completing Figure 12.

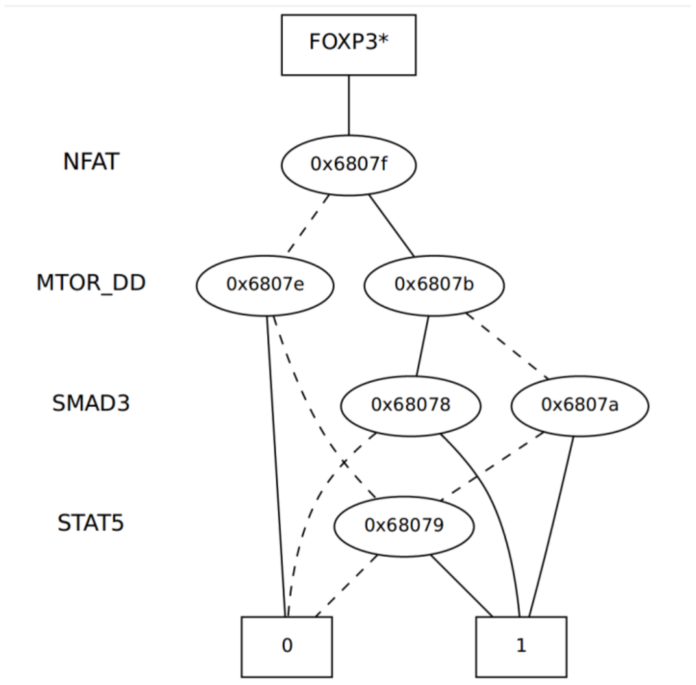


Figure 16 BDD of FOXP3*, Boolean function update rule for FOXP3

As stated in the previous paragraph we can as well display the BDD of each rule of the model individually. Figure s16 is an example of the BDD of a rule. It shows the rule of FoxP3 which is an element of particular interest for the remainder of this thesis.

We highlighted a couple of elements, CD28, TCR, TGF-β, IL-2 and FOXP3 in Section 11.1 because those elements are the inputs and outputs of the model. More precisely, CD28, TCR and TGF- β are the inputs and IL-2 and FOXP3 are the outputs. TCR is actually represented by two variables TCR-HIGH and TCR-LOW. Our goal is then to find an expression of FOXP3 as a function of CD28, TCR-HIGH, TCR-LOW and TGF-β.

We now present our results for this purpose.

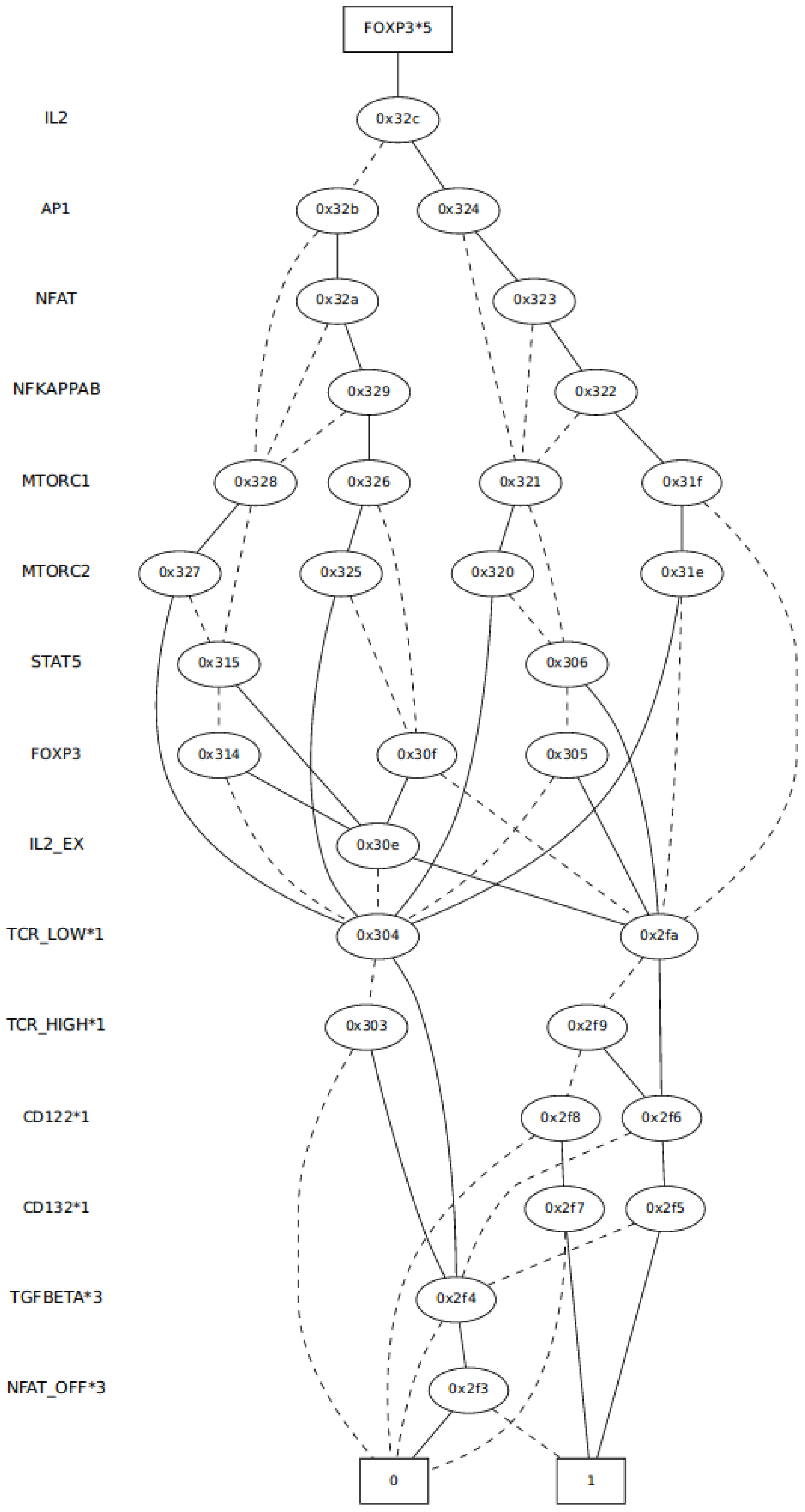


Figure 17 Expression of FOXP3^{*5}

We show in Figure 17 an expression of FOXP3 connected to 3 out of the 4 inputs of the system. In Figure 17, X*s, where X is a given variable name and *s* a given integer, means value of variable X at step *s*. To follow the notation introduced in Chapter 10, we should write $X^{*^{s}}$, however, the function that we used to generate the graphical representation of the BDD, generates it in a format that does not allow editing.

Using the unrolling approach, we reach all the input for FOXP3 after 13 steps but the BDD obtained then is too large to be shown here. TGF-β is reached at the second step while TCR-HIGH and TCR-LOW are reached at the third step. From those results, we can draw the following observations, the effect of CD28 on FOXP3 is delayed compared to the effect of TCR-LOW, TCR-HIGH and TGF- β. FOXP3 can have an oscillatory behavior with a period of 5 time steps since FOXP3$^{*^{5}}$ is a function of FOXP3 as shown in Figure17.

12.0 CONCLUSION

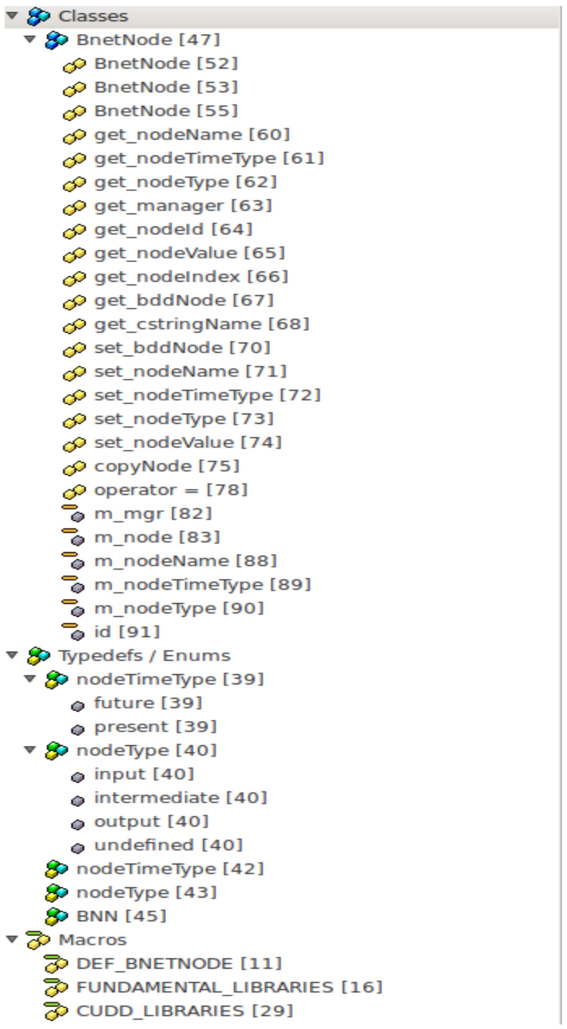
The main goal of the study of biological systems in a control engineering viewpoint is to develop strategies to move biological systems from an undesirable state, such as a disease, to a target state, such as restored health.

For this purpose, we proposed a method to drive biological systems modeled by Boolean networks from an arbitrary state to a desirable state by controlling the inputs of the network. This operation is possible when the modeled system is controllable. Therefore, we first had to study the controllability of the modeled systems. To study the controllability of the systems and to develop our method, we based our work on the algebraic state-space representation which provides a framework similar to the state-space representation in modern control strategy for Boolean networks.

However, because of the limitations of that representation, such that the assumption regarding the synchronicity of the Boolean network and the scalability to real life networks. We explored a new way to control biological systems. Considering the importance of transfer functions in control theory for continuous systems, we wanted to suggest an equivalent object for Boolean networks. On that basis, we proposed a method to find for each output of a Boolean network, a function that relates that output to the inputs of that network. We concluded our thesis by applying this methodology to a real life, biological example.

APPENDIX A

BNETNODE CLASS

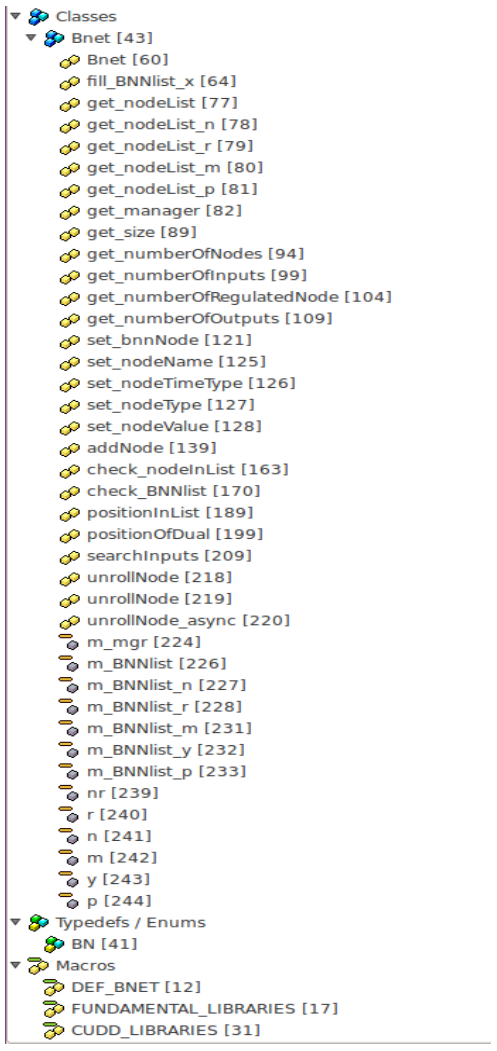


```
▼ Classes
  ▼ BnetNode [47]
      BnetNode [52]
      BnetNode [53]
      BnetNode [55]
      get_nodeName [60]
      get_nodeTimeType [61]
      get_nodeType [62]
      get_manager [63]
      get_nodeId [64]
      get_nodeValue [65]
      get_nodeIndex [66]
      get_bddNode [67]
      get_cstringName [68]
      set_bddNode [70]
      set_nodeName [71]
      set_nodeTimeType [72]
      set_nodeType [73]
      set_nodeValue [74]
      copyNode [75]
      operator = [78]
      m_mgr [82]
      m_node [83]
      m_nodeName [88]
      m_nodeTimeType [89]
      m_nodeType [90]
      id [91]
▼ Typedefs / Enums
  ▼ nodeTimeType [39]
      future [39]
      present [39]
  ▼ nodeType [40]
      input [40]
      intermediate [40]
      output [40]
      undefined [40]
    nodeTimeType [42]
    nodeType [43]
    BNN [45]
▼ Macros
    DEF_BNETNODE [11]
    FUNDAMENTAL_LIBRARIES [16]
    CUDD_LIBRARIES [29]
```

Figure 18 Composition of the BnetNode class

APPENDIX B

BNET CLASS



```
▼ Classes
  ▼ Bnet [43]
      Bnet [60]
      fill_BNNlist_x [64]
      get_nodeList [77]
      get_nodeList_n [78]
      get_nodeList_r [79]
      get_nodeList_m [80]
      get_nodeList_p [81]
      get_manager [82]
      get_size [89]
      get_numberOfNodes [94]
      get_numberOfInputs [99]
      get_numberOfRegulatedNode [104]
      get_numberOfOutputs [109]
      set_bnnNode [121]
      set_nodeName [125]
      set_nodeTimeType [126]
      set_nodeType [127]
      set_nodeValue [128]
      addNode [139]
      check_nodeInList [163]
      check_BNNlist [170]
      positionInList [189]
      positionOfDual [199]
      searchInputs [209]
      unrollNode [218]
      unrollNode [219]
      unrollNode_async [220]
      m_mgr [224]
      m_BNNlist [226]
      m_BNNlist_n [227]
      m_BNNlist_r [228]
      m_BNNlist_m [231]
      m_BNNlist_y [232]
      m_BNNlist_p [233]
      nr [239]
      r [240]
      n [241]
      m [242]
      y [243]
      p [244]
▼ Typedefs / Enums
    BN [41]
▼ Macros
    DEF_BNET [12]
    FUNDAMENTAL_LIBRARIES [17]
    CUDD_LIBRARIES [31]
```

Figure 19 Composition of the Bnet class

APPENDIX C

T-CELL MODEL

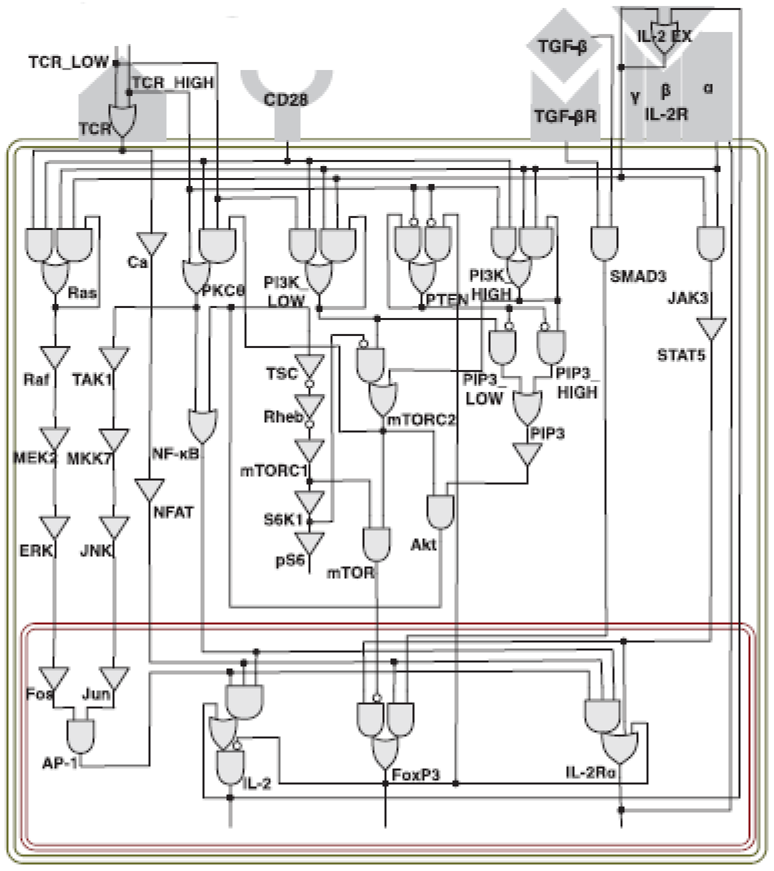


Figure 20 T-Cell model

BIBLIOGRAPHY

[1] Stuart Kauffman, Carsten Peterson, Björn Samuelsson and Carl Troein “Random Boolean Network Models and the Yeast Transcriptional Network” PNAS 2003 100 (25) 14796-14799

[2] Wynn, Michelle L. et al. “Logic-Based Models in Systems Biology: A Predictive and Parameter-Free Network Analysis Method.” Integrative biology : quantitative biosciences from nano to macro 4.11 (2012): 10.1039/c2ib20193c. PMC. Web. 18 Apr. 2017.

[3] Riedel, M. D. (2004). Cyclic combinational circuits (Doctoral dissertation, California Institute of Technology).

[4] Riedel, M. D., & Bruck, J. (2012). Cyclic boolean circuits. Discrete Applied Mathematics, 160(13-14), 1877-1900.

[5] Assieh Saadatpour, Réka Albert, “Boolean modeling of biological regulatory networks: A methodology tutorial” Elsevier Volume 62, Issue 1. 15 July 2013, pp 3-12

[6] Assieh Saadaptour, Réka Albert, “Attractor analysis of asynchronous models of signal transduction networks” Elsevier Volume 266, Issue 4, 21 October 2010 pp 641-65

[7] Ilya Shmulevich, Edward R. Dougherty, Seungchan Kim, Wei Zhang; “Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks”. Bioinformatics 2002; 18 (2): 261-274. doi: 10.1093/bioinformatics/18.2.261

[8] R. Thomas, On the relation between the logical structure of systems and their ability to generate multiple steady states and sustained oscillations, in: J.Della Dora, J. Demongeot, B. Lacolle (Eds.), Numerical Methods in the Study of Critical Phenomena, Springer Verlag, Berlin, 1981, pp. 180–193

[9] Cheng, Daizhan, and Hongsheng Qi. "A linear representation of dynamics of Boolean networks." IEEE Transactions on Automatic Control 55.10 (2010): 2251-2258.

[10] Introduction to Linear Algebra / Edition 5 by Lee W. Johnson, R. Dean Riess, Jimmy T.

[11] Zdzislaw Bubnicki, Moden Control Theory, Springer Publishing Company, Incorporated, 2010

[12] Chi-Tsong Chen, Linear System Theory and Design, 4th Edition, Oxford University Press, Oxford, UK, 2013

[13] Yang-Yu Liu, Jean-Jacques Slotine and Albert-Lásló Barabási “Controllability of complex networks” Nature, Vol. 473, No. 7346. (12 May 2011), pp. 167-173.

[14] A.J Gates, and L.M Rocha, Control of complex networks requires both structure and dynamics. Sci.Rep. 6, 24456; doi:10.1038/srep24456 (2016).

[15] Rui Li, Meng Yang and Tianguang Chu, “Controllability and observability of Boolean networks arising from biology ” Chaos: An Interdisciplinary Journal of Nonlinear Science 2015 25:2

[16] D.Laschov, M.Margaliot “Controllability of Boolean control networks via the Perron-Frobenius theory”, Automatica 2012 1218-1223

[17] Ming Liu, “Analysis and Synthesis of Boolean Network”, thesis in Electronic and Computer systems, KTH School of Information and Communication Technology, Stockholm Sweden

[18] D.Cheng,and H.Qi “Semi-tensor product and its application,” A survey.In: Proc. ICCM 2007. Vol. 3 (641668).

[19] D.Cheng and H.Qi ,“An Introduction to Semi-Tensor Product of Matrices and Its Applications”, World Scientific

[20] D.Cheng and H.Qi “Input-state approach to Boolean networks. IEEE Transactions on Neural Network, 20(3), 512-521.

[21] D.Cheng and H.Qi ,“ Controllability and Observabillity of Boolean control networks”, Automatica 2009 1645-1667

[22] Laschov, D., Margaliot, M., & Even, G. (2013). Observability of Boolean networks: A graph-theoretic approach. Automatica, 49(8), 2351-2362.

[23] N. Bof, E. Fornasini, M.E Valcher “Output feedback stabilization of Boolean control networks”, Automatica 2015 21-28

[24] Quine, W.V. (1982), Methods of Logic, 4th edition, Cambridge, MA: Harvard University Press

[25] DeMicheli, G., Synthesis and Optimization of Digital Circuits.McGraw-Hill Higher Education, 1994.

[26] Bryant, R. E. (1992). Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Computing Surveys (CSUR), 24(3), 293-318.

[27] Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. Computers, IEEE Transactions on, 100(8), 677-691.

[28] Somenzi, F. CUDD: Colorado university decision diagram package.(1996).

[29] Abhishek Garg, Ioannis Xenarios, Luis Mendoza, Giovanni DeMicheli, « An Efficient Method for Dynamic Analysis of Gene Regulatory Networks and inÂ silico Gene Perturbation Experiments, Research in Computational Molecular Biology: 11th Annual International Conference, RECOMB 2007, Oakland, CA, USA, April 21-25, 2007. Proceedings 62-76 Springer

[30] Garg, A., Di Cara, A., Xenarios, I., Mendoza, L., & De Micheli, G. (2008). Synchronous versus asynchronous modeling of gene regulatory networks. Bioinformatics, 24(17), 1917-1925.

[31] Garg, A., Mohanram, K., Di Cara, A., De Micheli, G., & Xenarios, I. (2009). Modeling stochasticity and robustness in gene regulatory networks. Bioinformatics, 25(12), i101-i109.

[32] Miskov-Zivanov, N., Turner, M. S., Kane, L. P., Morel, P. A., & Faeder, J. R. (2013). Duration of T cell stimulation as a critical determinant of cell fate and plasticity. Science signaling, 6(300), ra97

[33] https://www.medicinenet.com/script/main/art.asp?articlekey=11300

[34] Sayed Khaled & A. Telmer, Cheryl & Butchy, Adam & Miskov-Zivanov, Natasa. (2018). Recipes for Translating Big Data Machine Reading to Executable Cellular Signaling Models. 1-15. 10.1007/978-3-319-72926-8_1

[35] K. Kretschmer, I. Apostolou, D. Hawiger, K. Khazaie, M. C. Nussenzweig, H. von Boehmer, Inducing and expanding regulatory T cell populations by foreign antigen. Nat. Immunol. 6,1219–1227 (2005).

[36] M. S. Turner, L. P. Kane, P. A. Morel, Dominant role of antigen dose in CD4+Foxp3+regulatory T cell induction and expansion. J. Immunol. 183, 4895–4903 (2009).

[37] S. Yamazaki, D. Dudziak,G. F. Heidkamp, C. Fiorese, A. J. Bonito, K. Inaba, M. C. Nussenzweig, R. M. Steinman, CD8$^+$ CD205$^+$ splenic dendritic cells are specialized to induce Foxp3$^+$ regulatory T cells. J. Immunol. 181, 6923–6933 (2008).

[38] Leibniz, “Mathematische Schriften” (Explanation of binary arithmetic), vol. 8 Berlin A.Ascher, 1863, pp 223-227

[39] http://www.leibniz-translations.com/binary.html

[40] [Bahar, R. I., Frohm, E. A., Gaona, C. M., Hachtel, G. D., Macii, E., Pardo, A., & Somenzi, F. (1997). Algebric decision diagrams and their applications. Formal methods in system design, 10(2-3), 171-206.

[41] Henrik Reif Andersen, “An Introduction to Binary Decision Diagrams” lecture note for Efficient Algorithms and Programs, Fall 1999, University of Copenhagen, Copenhagen Danemark

[42] Savoj, H. (1992). Don't cares in mult-level network optimization (Doctoral dissertation, University of California, Berkeley).

[43] Bartlett, K. A., Brayton, R. K., Hachtel, G. D., Jacoby, R. M., Morrison, C. R., Rudell, R. L., ... & Wang, A. (1988). Multi-level logic minimization using implicit don't cares. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 7(6), 723-740.