Algorithmes pour la prédiction de stratégies de reprogrammation cellulaire dans les réseaux Booléens.

Hugues Mandon

Algorithms for Cell Reprogramming Strategies in Boolean Networks

Thèse de doctorat de l'Université Paris-Saclay
préparée à École Normale Supérieure Paris-Saclay

École doctorale n°580 Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat: Informatique

Thèse présentée et soutenue à Cachan, le 19 Novembre 2019, par

Hugues Mandon

Contents

Chapter 1

Introduction

During the embryonic development, one single cell divides itself multiple times, allowing the embryo to grow and finally becoming a baby. The first cells of the embryo are known as embryonic stem cells. During the development process, these cells differentiate into neurons, muscle cells, bone marrow cells, etc. This differentiation process is gradual, and can be seen as a landscape, where the differentiation process takes place in descending valleys, and in which the cell naturally follows a path downhill. This landscape is shown in figure 1.1 (left). Notice that even during this path downhill, there might be some hills to climb, as shown in the same figure (right). This is because some cells only partially differentiate, they are called progenitors. They divide to create new cells, that can differentiate to what is needed.

For many years, this differentiation process downhill was thought to be irreversible. However, in 2007, S. Yamanaka et al. found that some factors can induce stem cells from human differentiated cells (fibroblasts) [TTO+07]. Dedifferentiation was born. Using the same principle, some factors can allow a differentiated cell to become an other kind of differentiated cell. This is known as transdifferentiation. In the context of the epigenetic landscape, this means that a cell can be pushed uphill towards the origin or a progenitor, or that they can be pushed in an other valley, and roll downhill until reaching an other kind of differentiated cell.

Cell reprogramming is the term used to speak about both dedifferentiation and transdifferentiation. The factors for cell reprogramming were mostly found by deductive reasoning, knowledge of the network, trial and error, and intuitions. However, as more and more precise mathematical models for the cell are created, the possibility to compute solutions from the models has emerged.

Nowadays, finding solutions to cell reprogramming problems by using algorithms is a tremendous challenge faced by numerous teams, to try to reduce financial costs and time spent on experiments. Multiple approaches are used

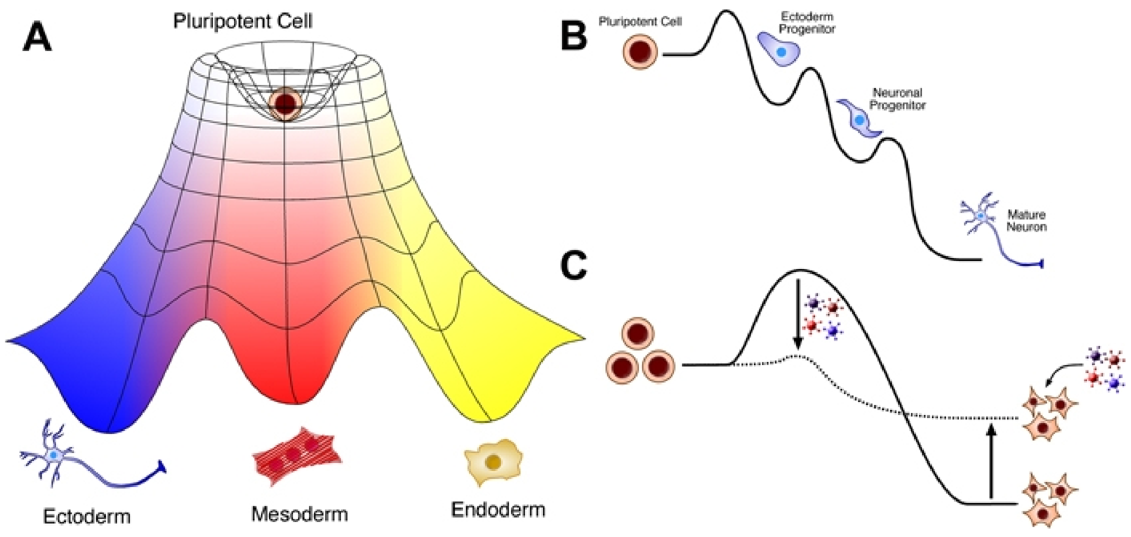

Figure 1.1: The epigenetic landscape, from [Rod08]

to find which factors are the most likely to succeed, using different methods to analyze the available data where computation times are often very high. Among them, creating a (partial) epigenetic landscape can be done, using probabilities and experimental data, or the goal can be to create a model of the cell, to understand better the differentiation process, and perturbing this model allows to find cell reprogrammings. These models can be continuous, they can be probabilistic, or they can be discrete, each of which either in time, values, or both. Different models and methodologies are described in section 2.3.

However, most solutions are ad hoc for a specific model and a given data set, which makes comparison difficult. As a result, a goal of this thesis is to have a unified vision of cell reprogramming, to be able to compare and categorize the different methods. We decided to focus on Boolean networks.

Boolean networks are widely employed for the modelling of the differentiation process and other processes, such as diseases, cell activity or cancer. They require minimal information and thus are particularly adapted to represent biological data, where we might be lacking detailed information. A Boolean network is a set of Boolean functions, each representing the behavior of an interesting part of the cell, most often a gene, a protein or a micro RNA. These functions return Boolean values: 0 or 1, which is a simplification of the state of the part of the cell, with 1 meaning it is active, or that it has a concentration high enough to influence the behavior of other components, and 0 meaning the opposite.

Boolean networks have an updating scheme, which dictate how the functions are applied to the current state of the network, thus dictating the future behavior of the network. We are particularly interested in the long-term behaviors of the network, called attractors. They correspond to the differentiated states of the cell, and among them are often the initial state of the cell reprogramming, as well as its target.

Cell reprogramming can be modelled in the scope of Boolean networks by a modification of the current state of the network, to represent a change in concentration of some components, or a modification of some Boolean functions, to represent a mutation of the DNA. After that, the modified model is studied, to find if it leads to the desired target. However, in experimental protocols, the experimenter sometimes needs to make concentration changes multiple times, waiting multiple hours or even days between each. This possibility has not been studied in the context of Boolean networks.

To introduce these, we had to define what are perturbations in the case of Boolean networks. Moreover, we want these perturbations to verify a property: when the perturbations are applied, the system always end up in the target attractor. This is easy to understand and define if there is only one perturbation in the initial state, but requires a formal definition when sequentiality is used, with perturbations at different times. We define Boolean networks and their dynamics in chapter 2, as well as the new definitions for perturbations, perturbation sequences, reprogramming strategies and reprogrammability. This chapter also states what methods are currently used to infer cell reprogramming, in Boolean networks and in other domains.

We did not think of sequentiality at first, and focused on similar methods that other algorithms use: analyzing the Boolean network to find perturbations from an initial state toward a desired attractor. During this, we understood the importance of dynamics, and that they need to be at least partly computed to find minimal solutions. At this point, we added sequentiality to the perturbations, in order to have a more precise control of the network. The static analysis is discussed in chapter 3. This static analysis already allows for sequentiality in the perturbations. The limits of the static analysis are also discussed, and why computing the dynamics, at least partly, is essential.

As a result, we wanted to be able to study the dynamics of the models, and to be able to have perturbations changing the dynamics or the state of the network. We created a modelling technique to add perturbations to a Boolean network dynamics, and we designed an algorithm to find all perturbation sequences smaller than an upper bound, from an initial state to a target. This algorithm, for complexity reasons, returns a set of state, so we designed an algorithm to extract the solutions from this set. We also designed an algorithm aimed at more practical uses, where every state might not be observable, and where some variables cannot be perturbed. These algorithms are detailed in chapter 4. These algorithms return sets of solutions, to allow multiple possibilities and experiments. The modelling technique to compute the transition graph with perturbations is explained in appendix A.

Lastly, in chapter 5 we tested our algorithms on published biological examples, to compare the results of sequential reprogramming to the ones obtained by other methods. We first studied the algorithms from chapters 3 and 4 on the cardiac network by Hermann et al. [HGZ$^+$12], where the static analysis returns very good results, thanks to the small size of the strongly connected components, and studied the special case of reprogramming from SHF to FHF. We also used the algorithms on a network from Curie Institute, described in [CMR$^+$15] by Cohen et al., where we showed that the perturbation sequences mostly show the artifacts of the model, with very strong input variables. As a result, we studied the possible perturbations when the input variables cannot be changed, with permanent and instantaneous perturbations. Permanent perturbations were shown to be surprisingly strong, where only one perturbation allows to reprogram to any kind of attractor when reprogramming is possible. We also studied the PC12 differentiation network explained and constructed in [OKS$^+$16] by Offerman et al., and the special case of cell cycle arrest to differentiation. In this reprogramming, we gave insight on the perturbations by studying the interaction graph.

Chapter 2

Formal Framework for Boolean Networks Reprogramming

This chapter explains and defines the different kind of objects we will be working with, Boolean networks, interaction graphs, transition graphs and their attractors, and basins of a set of nodes of a graph. It also gives new definitions such as perturbation (of a Boolean network), perturbation sequences, reprogramming strategies and reprogrammability, and gives a thorough state of the art.

2.1 Boolean Networks and their dynamics

Boolean networks are widely used to model biological systems, simplifying the gene interactions to activation and inhibition, and the concentration levels to active or inactive. These networks have proven themselves precise enough to model the dynamics of biological systems[MSRSL10], and allow using only partial information on the interactions between the components of the system.

Boolean Networks: A Boolean network is a function, associating to a variable its next value given the values of other variables, as defined as follows.

Definition 1. *A* Boolean Network *(BN) of dimension n is a function f such that:*

$$\begin{aligned} f : \{0,1\}^n &\to \{0,1\}^n \\ x = (x_1, \dots, x_n) &\mapsto f(x) = (f_1(x), \dots, f_n(x)) \end{aligned}$$

Example 1 gives an example of a Boolean network.

Example 1. *An example of BN of dimension 3 ($n = 3$) is*

$$\begin{aligned} f_1(x) &= x_3 \vee (\neg x_1 \wedge x_2) \\ f_2(x) &= \neg x_1 \vee x_2 \\ f_3(x) &= x_3 \vee (x_1 \wedge \neg x_2) \end{aligned}$$

Interaction graph: The interactions between the genes can be simplified in an interaction graph: a graph in which known positive influences are represented by positive edges. The edges of interaction graphs are often a simplification of multiple biological processes. For example, a gene A could be transcripted to a micro RNA A_m which is then translated to a protein A_p. This protein A_p can bind on an enhancer of gene B, to allow it to be transcripted. In this case, we consider that there is a positive interaction from A to B, and the edge $A \xrightarrow{+} B$ will be in the interaction graph. In the same fashion, negative interactions are represented by negative edges.

Definition 2 (Interaction Graph)**.** *An interaction graph is denoted as $G = (V, E)$, with V being the vertex set, and E being the directed, signed edge set, $E \subseteq (V \times V \times \{-, +\})$*

A cycle between a set of nodes $C \subseteq V$ is said positive (resp. negative) if and only if there is an even (odd) number of negative edges between those nodes.

An interaction graph can also be defined as an abstraction of a Boolean network: the vertex set is the set of variables, and the Boolean functions are abstracted in the edges: if there exists two states, x and y, and a variable v, with only $x_v \neq y_v$, and all other variables are equal, and if $f_v(x) \neq f_v(y)$, then f_v contains either x_u or $\neg x_v$ (depending if the value of f_v increases when v increases or when it decreases).

Definition 3 (Interaction Graph of a Boolean network ($G(f)$))**.** *An interaction graph can be obtained from the Boolean network f: the vertex set is $[1, n]$, and for all $u, v \in [1, n]$ there is a positive (resp. negative) arc from u to v if $f_{vu}(x)$ is positive (resp. negative) for at least one $x \in \{0, 1\}^n$ (For every u, $v \in \{1, ..., n\}$, the function f_{vu} is the discrete derivative of f_v considering u, defined on $\{0, 1\}^n$ by : $f_{vu}(x) := f_v(x_1, .., x_{u-1}, 1, x_{u+1}, .., x_n) - f_v(x_1, .., x_{u-1}, 0, x_{u+1}, .., x_n)$).*

As the interaction graph of a Boolean network abstracts away part of the function specification, two different Boolean networks can have the same interaction graph.

Example 2. *Fig. 2.1 gives an example of an interaction graph, which is also equal to $G(f)$, where f is the Boolean network of Ex.1.*

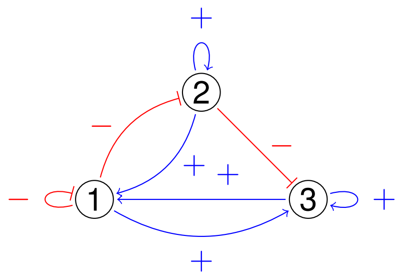

Figure 2.1: Interaction graph of Ex.1 A “normal” blue arrow means an activation, and a “flattened” red arrow means an inhibition.

Transition graph and attractors: The dynamics of a Boolean network f are modelled by transitions between its states $x \in \{0,1\}^n$. Here, we consider the *fully asynchronous semantics* of Boolean networks: a transition updates the value of only one variable $i \in [1,n]$. Notice that examples for the relevance of sequential reprogrammability can easily be exhibited in the synchronous update semantics as well.

Thus, from a state $x \in \{0,1\}^n$, there is one transition for each vertex i such that $f_i(x) \neq x_i$. The transition graph (Def. 4) is a digraph where vertices are all the possible states $\{0,1\}^n$, and edges correspond to asynchronous transitions. The transition graph of a Boolean network f can be denoted as $\mathsf{STG}(f)$.

Definition 4 (Transition graph of a Boolean network ($\mathsf{STG}(f)$)). *The transition graph (also known as state graph) of a Boolean network f is the graph $\mathsf{STG}(f) = (V, E_f)$ having $V = \{0,1\}^n$ as vertex set and the edges set $E_f = \{x \rightarrow y \mid x \in \{0,1\}^n, \exists i \in [1,n], f_i(x) = \neg x_i = y_i$ and $\forall j \in [1,n] \setminus \{i\}, y_j = x_j\}$.*

If at least one path exists from x to y, we can write $x \rightarrow^* y$. The set of all paths from x to y is interesting to better understand the dynamics when used in the transition graph, and is required for future definitions. However, this set is infinite, since there could be infinitely long paths from x to y, because of loops. Therefore, we use the set of paths without loops from x to y, $\mathsf{paths}(x,y)$, (def. 5). This means that the same vertex will never appear twice.

Definition 5 (Set of paths without loops from x to y, $\mathsf{paths}(x,y)$). $\mathsf{paths}(x,y) = \{x = z_1 \rightarrow \cdots \rightarrow z_k = y \mid k \in \mathbb{N}, \{z_1,\ldots,z_k\} \in \{0,1\}^{nk}, \forall a,b \in \{z_1,\ldots,z_k\}^2, a \neq b\}$

This notion of set of all loop-less paths from a state x to a state y can be extended to a set of loop-less paths from a set of states X to a set of states Y.

Definition 6 (Set of paths from X to Y). $\mathsf{paths}(X,Y) = \bigcup_{x \in X, y \in Y} \mathsf{paths}(x,y)$

$$\boxed{\begin{aligned}f_1(x) &= x_1 \vee x_2\\ f_2(x) &= x_1 \vee \neg x_2\end{aligned}}$$

$$\begin{tikzcd}
\boxed{10} \arrow[r] & \boxed{11} \\
\boxed{00} \arrow[r, bend right] & \boxed{01} \arrow[u] \arrow[l, bend right]
\end{tikzcd}$$

Figure 2.2: Boolean network and its transition graph illustrating strong fairness.

Strong fairness: In this work, we assume *strong fairness* of the dynamics, meaning that if a path goes infinitely often in a state, it takes infinitely often all outgoing edges. In biology, a system is supposed to eventually leave transient cycles, which is ensured by strong fairness.

Example 3. *Fig. 2.2 gives a Boolean network f and its transition graph, where strong fairness is important. Indeed, in this example, we can see that 00 and 01 are inside a loop. Without strong fairness, we have to consider that they might never leave the loop, whereas with strong fairness, we know it will eventually get out of the loop, and reach 11.*

Attractors: The terminal strongly connected components of the transition graph can be seen as the long-term dynamics, phenotypes or "fates" of the system; throughout, we will refer to them as *attractors*. An attractor may model a unique state, referred to as a *fixpoint*, $f(x) = x$, or sustained oscillations (*cyclic attractor*) among several states.

Definition 7 (Attractor)**.**

$$\begin{aligned}
A \subseteq \{0,1\}^n \textit{ is an attractor} \Leftrightarrow{}& \\
& A \neq \emptyset \\
& \textit{and } \forall x \in A, \forall y \in \{0,1\}^n \setminus A,\ x \not\to^* y \\
& \textit{and } \forall x, y \in A,\ x \to^* y
\end{aligned}$$

If $|A| = 1$ then A is a fixpoint. Otherwise, A is a cyclic attractor.

We denote $\mathcal{A}(f)$ the list of all attractors of a Boolean network f.

Example 4. *Fig.2.3 gives the transition graph of the asynchronous dynamics of Boolean network of Ex.1, with the attractors in magenta.*

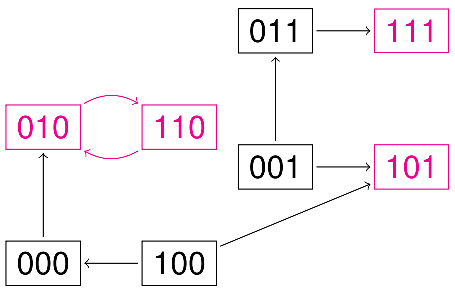

Figure 2.3: Transition graph of the Boolean network defined in Ex.1.

Predecessors, successors and basins: The computation of the predecessors and successors utilize two basic functions, with a set of vertices X of the graph $G = (V, E)$: $Preimage(X, G) = \{s' \in V \mid \exists s \in X \text{ such that } (s' \to s) \in E\}$, which returns the direct predecessors of X in G, i.e. $\mathsf{pre}(G)(X)$; $Image(X, G) = \{s' \in V \mid \exists s \in X \text{ such that } (s \to s') \in E\}$, which returns the direct successors of $X \subseteq V$ in G, i.e. $\mathsf{post}(G)(X)$. To simplify, we define $Preimage^i(X, G) = \underbrace{Preimage(\dots(Preimage(X, G)))}_{i\ times}$ with $Preimage^0(X, G) = X$ and $Image^i(X, G) = \underbrace{Image(\dots(Image(X, G)))}_{i\ times}$ with $Image^0(X, G) = X$. In this way, the set of all predecessors of X via transitions in G is defined as an iterative procedure $\mathsf{pre}^*(G)(X) = \bigcup_{i=0}^{m} Preimage^i(X, G)$ such that $Preimage^m(X, G) = Preimage^{m+1}(X, G)$. Similarly, the set of all successors of X via transitions in G is defined as an iterative procedure $\mathsf{post}^*(G)(X) = \bigcup_{i=0}^{m} Image^i(X, G)$ such that $Image^m(X, G) = Image^{m+1}(X, G)$. Lastly, we define $\mathsf{pre}^+(G)(X) = \bigcup_{i=1}^{m} Preimage^i X, G = \mathsf{pre}^*(G)(\mathsf{pre}(G)(X))$, with m verifying $Preimage^m(X, G) = Preimage^{m+1}(X, G)$, which is the set of all predecessors of X, starting from its predecessors. This set can include X, if it is its own predecessor, because of a loop inside the set.

The (strong) basin of X is the set of states that always eventually reach a state in X. It should be noted that X can be any set of states, not necessarily an attractor. In other works, the term strong basin is used instead of basin, but only for attractors. Here, we chose to use a different term, basin, because it can apply to any set of node. Moreover, the previous definition of $\mathsf{pre}^*(G)(X)$ covers what others call weak basin of X, the set of all vertices having a path to X.

Definition 8 (Basin of X in G, $\mathsf{bas}(G, X)$)**.** *Let X be a set of nodes in G. The basin of X, $\mathsf{bas}(G, X)$ is the biggest set Y such that $X \subseteq Y$ and $\forall y \in Y \setminus$*

$X, \mathsf{post}(G)(y) \subseteq Y \wedge \exists x \in X, y \rightarrow^* x$.

Basins are particularly interesting in the case of attractors: the basin of an attractor is the set of all states that will inevitably end up in the attractor. In most applications to cell reprogramming, the target of a reprogramming is an attractor.

Moreover, if a set of states A is in the basin of an other set of states B, then the basin of A is in the basin of B.

Property 1. *Given two sets of states, A and B of a graph G, if $A \subseteq \mathsf{bas}(G, B)$ then $\mathsf{bas}(G, A) \subseteq \mathsf{bas}(G, B)$.*

Proof. If there exists $a \in \mathsf{bas}(G, A)$ such that $a \notin \mathsf{bas}(G, B)$, then, there exists $b \in \mathsf{post}(G)(a)$ that is not in $\mathsf{bas}(G, B)$, otherwise $\mathsf{bas}(G, b)$ is not minimal. Recursively applying this reasoning to b and its successors not in $\mathsf{bas}(G, B)$, either there exists a loop of states that are in $\mathsf{bas}(G, A)$ and are not in $\mathsf{bas}(G, B)$, in which case $\mathsf{bas}(G, B)$ is not minimal (because this loop could be added to $\mathsf{bas}(G, B)$; or since there exists $a' \in A$ such that $a \rightarrow^* a'$, then $a' \notin \mathsf{bas}(G, B)$. However, $A \subseteq \mathsf{bas}(G, B)$, thus $a \in \mathsf{bas}(G, b)$. □

2.2 Perturbations, reprogramming strategies and reprogrammability

This section introduces definitions for the perturbations, the perturbation sequences, the reprogramming strategies and for existential and inevitable reprogrammability.

2.2.1 Perturbations

In order to modify the future behavior of a Boolean network, we want to perturb either one of its functions or its actual state. We define a *perturbation valuation* (def. 9) as a set of variables of a Boolean network and associated values. The values must be possible values of the system, 0 or 1 since we are working with Boolean networks, and the variables should all be distinct.

Definition 9 (Perturbation valuation)**.** *A perturbation valuation M is a set of values associated to variables $M = \{v_1 = a_1, \dots, v_k = a_k\}$ with $k \leq n$, a_i values verifying $\{a_1, \dots, a_k\} \in \{0, 1\}^k$, and for all $i, j \in [1, k]$,$i \neq j \Rightarrow v_i \neq v_j$.*

This perturbation valuation indicates which variables to perturb. The set of all possible perturbation valuations for a network f is denoted $\mathbb{M}$. We define two operators to perturb the system with a perturbation valuation.

First, the state perturbation operator consists of applying the perturbation valuation to the current state x of a Boolean network f. We can note that since the operator will only modify the value of some variables, the function f is irrelevant, and only the size n of the Boolean network matters.

Definition 10 (State perturbation)**.** *We define $\chi_M(x)$ as the state perturbation of state x (of any Boolean network of size n) with perturbation valuation $M = \{v_1 = a_1, \ldots, v_k = a_k\}$: $\chi_M(x) = x'$ where $\forall v \in [k+1, n], x'_v = x_v$ and $\forall i \in [1, k], x'_{v_i} = a_i$.*

On the other hand, the function perturbation operator consists of changing the Boolean network f function according to the perturbation valuation. In this case, the state of the Boolean network is irrelevant.

Definition 11 (Function perturbation)**.** *We define $\phi_M(f)$ as the function perturbation of the Boolean network f (of size n) with perturbation valuation $M = \{v_1 = a_1, \ldots, v_k = a_k\}$: $\phi_M(f) = f'$ where $\forall v \in [1, n] \setminus \{v_1, \ldots, v_k\}$, $f'_v = f_v$ and $\forall i \in [1, k]$, $f'_{v_i} = a_i$.*

We can see that applying state or function perturbation operator will result in a different behavior for the impacted variables. Applying the state perturbation operator will change the variable's value, changing the current behavior of the system, without changing the function, making it an instantaneous perturbation with no lasting effects. Applying function perturbation operator will modify the variable long-term behavior, the perturbation is always active.

From these operators, we can define instantaneous and permanent perturbations of a Boolean network f, which will be two of the main perturbations discussed in the next chapters. Both of these perturbations take (M, L, x) as parameters: a perturbation valuation M, the perturbation valuation of all previous function perturbations L on the Boolean network f, and one state of f ; and return a couple (f', x') as a result of the perturbation.

Instantaneous perturbations, $\mathsf{pert}_I(M, L, x)$, are short-term perturbations of the network, i.e., only x is perturbed. However, the function could have been perturbed previously, thus we keep the current active function perturbations.

Definition 12 (Instantaneous perturbation)**.** $\mathsf{pert}_I(M, L, x) = (\phi_L(f), \chi_M(x))$.

Permanent perturbations, $\mathsf{pert}_P(M, L, x)$, are long-term perturbations of the network, where both f and x will be perturbed. We chose to perturb both function and state because changing the function to a constant will always result in an update where the variable's value is updated to the constant. Not changing the variable's value at the same time than the function would result in some behaviors which are mostly artifacts of the asynchronous updating scheme. As a result, we change the variable's value and ensure it stays at this value by modifying the function as well.

To have an easier notation for updating the valuation with the function perturbations, we define the operator $+$ for valuation sets as:

$$M' + M = \{(v = a) \mid (v = a) \in M \text{ or } [(v = a) \in M' \text{ and } \forall (v' = a') \in M, v \neq v']\}$$

As a result, a permanent perturbation modifies f by updating the past valuation with the new one.

Definition 13 (Permanent perturbation)**.** $\text{pert}_P(M, L, x) = (\phi_{L+M}(f), \chi_M(x))$.

Since both kinds of perturbations use the same parameters and return the same objects, we will use the notation $\text{pert}(M, L, x) = (\phi_{M'}(f), x')$ when there is no need to precise if the perturbation is instantaneous or permanent. In this case, M' is either L for instantaneous perturbations or $L + M$ for permanent perturbations.

Perturbation sequence: We are interested in multiple perturbations at different times. We need to define perturbation sequences, which are ordered lists of couples (perturbation, parameters). We decided to focus on perturbation sequences where all perturbation in a sequence are of the same type, either instantaneous or permanent.

Definition 14 (Instantaneous perturbations sequence)**.** *An instantaneous perturbation sequence is an ordered list of k instantaneous perturbations for some k ; we denote it as $S = [(\text{pert}_I, (M_1, \emptyset, x_1)), \ldots, (\text{pert}_I, (M_k, \emptyset, x_k))]$.*

Definition 15 (Permanent perturbations sequence)**.** *A permanent perturbation sequence is an ordered list of k permanent perturbations for some k ; we denote it as $S = [(\text{pert}_P, (M_1, L_1, x_1)), \ldots, (\text{pert}_P, (M_k, L_k, x_k))]$, with $L_1 = \emptyset$ and for all $1 < i \leq k$, $L_i = L_{i-1} + M_i$.*

In the case of permanent perturbations sequence, an interesting case to make is to allow the Boolean network f to revert back to its original function when in some states. Therefore, we define a new operator, the function restoration operator, pert_{-1} with parameters a state x and L, the set of valuations for the function perturbations on f, and W a set of variables which function has been perturbed and will return to its original value.

Definition 16 (Function restoration)**.** *Given a Boolean network f, a perturbation valuation for all previous function perturbations L, and $W \subseteq V$ verifying $\forall v \in W, \exists (v = a) \in L$. We define the function restoration operator as* $\mathsf{pert}_{-1}(W, L, x) = (\phi_M(f), x)$ *with* $M = \{(v = a) \mid (v = a) \in L$ *and* $v \notin W\}$.

With this operator, we can define a temporary perturbations sequence, a sequence of permanent perturbations with one or multiple function restorations, over all perturbed variables. In this case, there can be multiple function restorations at different states during the sequence, but each perturbed variable should be restored at one point.

Definition 17 (Temporary perturbations sequence)**.** *A temporary perturbation sequence $S = [s_1, \ldots, s_{k+m}]$ is a sequence containing k permanent perturbations and m function restorations for some k and some m , where the following property is true:*
$\forall i \in [1, k], \forall j \in [1, |M_i|], \exists s_a = (\mathsf{pert}_{-1}, (W, L, x)) \in S$ *where* $v_j \in W$ *and* $s_b = (\mathsf{pert}_P, (M_i, L, x_i))$ *verifies* $b < a$.

Since there always exist a state in which a perturbed function is restored, the Boolean network resulting of the last operator in the sequence is always f. Once again, we made this choice for the sake of simplicity, to ensure that the final network is the same than the original, which corresponds to having long external influences on a cell in biology, without modifying its DNA.

"One-step" perturbation sequence: A "one-step" perturbation sequence is a sequence where only one perturbation is done, and thus contains only one perturbation valuation. In most cases, this perturbation is applied in the initial state of the network.

Size of a perturbation and size of a perturbation sequence: The size of a perturbation $\mathsf{pert}(M, L, x)$ is the size of the perturbation valuation M, denoted $|M|$. The size of a perturbation sequence S, denoted $|S|$ is the sum of the sizes of all perturbation valuations M_i.

$$|S| = \Sigma_{\mathsf{pert}(M,L,x) \in S} |M|$$

We decided that the size of a perturbation sequence should represent the effort necessary to reprogram the cell. Thus, it needs to account for the number of interventions, the number of perturbations in the sequence, and the difficulty of the interventions, the size of each perturbation.

2.2.2 Reprogramming strategies

The goal of reprogramming a system is to apply the right set of perturbations in the right states of the system and eventually reach a target.

To find these perturbations, we need a strategy. A reprogramming strategy $\mathcal{S}$ is a function that associate to each couple (valuation of past function perturbations, state) an action. This action can either be to do a perturbation, instantaneous or permanent, with a perturbation valuation, or to do a function restoration on a set of nodes.

We define the set of possible actions as $\mathbb{A}$, with $2^{[1,n]}$ being the set of all subsets of $[1, n]$:

$$\mathbb{A} = (\mathsf{pert}_I \times \mathbb{M}) \cup (\mathsf{pert}_P \times \mathbb{M}) \cup (\mathsf{pert}_{-1} \times 2^{[1,n]})$$

Moreover, a possible "action" in a reprogramming strategy is the lack of action, to do nothing and let the Boolean network update itself, denoted nothing.

Definition 18 (Reprogramming strategy $\mathcal{S}$)**.** *A reprogramming strategy $\mathcal{S}$ is a function that associate actions to couples (valuation of past function perturbations, state).*

$$\begin{aligned} \mathcal{S} : \mathbb{M} \times \{0,1\}^n &\to \mathbb{A} \cup \{\text{nothing}\} \\ (L, x) &\mapsto \text{action} \end{aligned}$$

We denote the set of reprogramming strategies for a Boolean network as $\mathbb{S}$. We can note that a reprogramming strategy can be adapted to a more general case, where only part of the state is observable. In this case, instead of (L, x), it would be $(L, o(x))$ where $o(x)$ is the observable part of the state.

Note that reprogramming strategies are deterministic, with a given action based on the past perturbations and the current state of the system. Once a strategy has been defined, we need to know what are the semantics of the strategy, what happens when the strategy is applied.

Reprogramming strategy semantics are how the strategy is applied. Applying this strategy to a Boolean network f will result in a new state transition graph, $\mathsf{STG}(\mathcal{S})$.

Definition 19 (Reprogramming strategy semantics $\mathsf{STG}(\mathcal{S})$)**.** $\mathsf{STG}(\mathcal{S})$ *is a transition graph with states $\mathbb{M} \times \{0,1\}^n$ and the transitions $(L, x) \to (L', x')$ with*
$(L', x') = \begin{cases} \mathcal{S}(L, x) \text{ if } \mathcal{S}(L, x) \neq \text{nothing} \\ L' = L \text{ and } x \to x' \in \mathsf{STG}(\phi_L(f)) \text{ otherwise.} \end{cases}$

Note that even if a reprogramming strategy is deterministic, its semantics are not, since they use updates from the Boolean network, which can be non-deterministic.

We want to know if a reprogramming strategy $\mathcal{S}$ semantics from an initial state or set of states, to a set of target states is successful, in the sense that it allows a set of target states to be reached. Moreover, there are different degrees of success in a reprogramming strategy: if the target is reachable with the strategy semantics, but might never be reached, the strategy is called existential. If the target is always reached, then the strategy is called inevitable.

As a result, we have the following definitions for inevitable and existential reprogramming strategies.

Definition 20 (Inevitable reprogramming strategy $\mathcal{S}$ from X to Y)**.** *Given a set of initial states X, a set of targets Y and a reprogramming strategy $\mathcal{S}$, what it means for the reprogramming strategy to be inevitable from X to Y is that $X \subseteq \mathsf{bas}(\mathsf{STG}(\mathcal{S}), Y)$.*

Definition 21 (Existential reprogramming strategy $\mathcal{S}$ from X to Y)**.** *Given a set of initial states X, a set of targets Y and a reprogramming strategy $\mathcal{S}$, what it means for the reprogramming strategy to be existential from X to Y is that $X \subseteq \mathsf{pre}^*(\mathsf{STG}(\mathcal{S}))(Y)$.*

Links between reprogramming strategies and reprogramming sequences: If a strategy $\mathcal{S}$ is existential, we can easily construct a perturbation sequence from it. First an empty list seq is created. Then a path from one state x in X to one state y in Y is chosen, and for each state reached in order from x to y, the action done by $\mathcal{S}$ is added to seq with its parameters, except if the action is $\mathrm{nothing}$.

To create perturbation sequences from an inevitable strategy, we need to consider the set of all loop-less paths from any x in X to any y in Y. Each path results in a sequence, constructed as previously for existential strategies. Moreover, the sequences from a same inevitable strategy can be grouped together to show the inevitability.

A perturbation sequence corresponds to an existential strategy if the following property is verified:

Property 2. *Given a set of initial states X, a set of targets Y and a sequence $S = [\mathsf{pert}_1, (M_1, \emptyset, x_1), \dots, \mathsf{pert}_k, (M_k, L_k, x_k)]$, S corresponds to an existential strategy if:*

- $\exists x \in X, x \in \mathsf{pre}^*(\mathsf{STG}(f))(\{x_1\})$

- $\forall (f', x') = \mathsf{pert}_i(M_i, L_i, x_i), x' \in \mathsf{pre}^*(\mathsf{STG}(f'))(\{x_{i+1}\})$
- $(f_{k+1}, x_{k+1}) = \mathsf{pert}_k(M_k, L_k, x_k), x_{k+1} \in \mathsf{pre}^*(\mathsf{STG}(f_{k+1}))(Y)$

To prove this property, we need to construct the right reprogramming strategy $\mathcal{S}$ from the sequence S. We begin with an "empty" strategy, where the only action done is $\mathrm{nothing}$ in all cases. Then, we apply the following procedure:

- We find the highest i such that $\mathsf{pert}_i(M_i, \emptyset, x_i) \in S$ and $\exists x \in X, x \in \mathsf{pre}^*(\mathsf{STG}(f))(\{x_i\})$.
- Then, we denote $(\phi_L(f), x') = \mathsf{pert}_i(M_i, \emptyset, x_i)$, and we add $(\emptyset, x_i) \mapsto (\mathsf{pert}_i, M_i)$ to $\mathcal{S}$.
- Using the same principle, we find the highest j such that $\mathsf{pert}_j(M_j, L, x_j) \in S$ and $x_i \in \mathsf{pre}^*(\mathsf{STG}(\phi_L(f)))(\{x_j\})$.
- We denote $(\phi_{L'}(f), x") = \mathsf{pert}_j(M_j, L, x_j)$ and we add $(L, x_j) \mapsto (\mathsf{pert}_j, M_j)$
- We repeat the two previous steps replacing L and x' by L' and $x"$ until $j = k$.
- Lastly, we add $(L_k, x_k) \mapsto (\mathsf{pert}_k, M_k)$ to $\mathcal{S}$.

Reviewing step by step the above procedure, we know that each step is possible. Because of the first point of property 2, there exists i such that $\mathsf{pert}_i(M_i, \emptyset, x_i) \in S$ and $\exists x \in X, x \in \mathsf{pre}^*(\mathsf{STG}(f))(\{x_i\})$. And because of the second point, we know that the steps 3 to 5 can be repeated until x_k is reached. We can note that during this procedure, we created a new perturbation sequence $S' = [(\mathsf{pert}'_1, (M'_1, \emptyset, x'_1), \ldots, (\mathsf{pert}'_{k'}, (M'_{k'}, L'_{k'}, x'_{k'}))]$ with perturbations from S and only from S, with $k' \leq k$ and with $(\mathsf{pert}'_{k'}, (M'_{k'}, L'_{k'}, x'_{k'})) = (\mathsf{pert}_k, (M_k, L_k, x_k))$.

We will now prove that this reprogramming strategy $\mathcal{S}$ is existential.

Proof. We want to prove that there exists x in X such that $x \in \mathsf{pre}^*(\mathsf{STG}(\mathcal{S}))(Y)$.

We know that $\exists x \in X, x \in \mathsf{pre}^*(\mathsf{STG}(f))(\{x'_1\})$, by construction of $\mathcal{S}$. To prove that $x \in \mathsf{pre}^*(\mathsf{STG}(\mathcal{S}))(\{x'_1\})$, we only need to prove that there is no perturbation $(\mathsf{pert}'_{i'}, (M'_{i'}, L'_{i'}, x'_{i'}))$ in S' preventing x'_1 to be reached, because the only differences between $\mathsf{STG}(f)$ and $\mathsf{STG}(\mathcal{S})$ is when actions other than $\mathrm{nothing}$ are made.

If there exists such a perturbation, then we know that this was a perturbation of the sequence S, that we denote as pert_i. Moreover, we know that pert'_1 is also a perturbation from the original sequence S, denoted as pert_j. We also chose to take the highest j possible, meaning that $i < j$.

We know that in the original sequence if we denote $(f', x') = \mathsf{pert}_j(M_j, L_j, x_j)$ then x' is in $\mathsf{pre}^*(\mathsf{STG}(f'))(\{x_{j+1}\})$. In S', we chose the highest h such that $x' \in \mathsf{pre}^*(\mathsf{STG}(f'))(\{x_h\})$. As a result, $h \geq j + 1$, and since we can repeat this reasoning for all perturbations of the sequence S', we can deduce that $h \leq i$. From these two inequalities, we deduce that $j < i$. Therefore, such a perturbation does not exists, and $x \in \mathsf{pre}^*(\mathsf{STG}(\mathcal{S}))(\{x'_1\})$. □

We repeat the same proof for each $(f', x') = \mathsf{pert}'_i(M'_i, L'_i, x'_i)$, until $(f'_{k'+1}, x'_{k'+1}) = \mathsf{pert}'_{k'}(M'_{k'}, L'_{k'}, x'_{k'})$ is reached. Moreover, each time we prove that x' is in $\mathsf{pre}^*(\mathsf{STG}(\mathcal{S}))(\{x'_{i+1}\})$, we get that x is in $\mathsf{pre}^*(\mathsf{STG}(\mathcal{S}))(\{x'_{i+1}\})$. We have that x is in $\mathsf{pre}^*(\mathsf{STG}(\mathcal{S}))(\{x'_i\})$ and that x' is in $\mathsf{pre}^*(\mathsf{STG}(\mathcal{S}))(\{x'_{i+1}\})$. Moreover, $x'_i \to x'$ is an edge of $\mathsf{STG}(\mathcal{S})$ by definition. As a result, $x \in \mathsf{pre}^*(\mathsf{STG}(\mathcal{S}))(\{x'\})$ and thus $x \in \mathsf{pre}^*(\mathsf{STG}(\mathcal{S}))(\{x'_{i+1}\})$. □

After the recurrence, we have $x \in \mathsf{pre}^*(\mathsf{STG}(\mathcal{S}))(\{x'_{k'+1}\})$. We know that $(\mathsf{pert}'_{k'}, (M'_{k'}, L'_{k'}, x'_{k'}) = (\mathsf{pert}_k, (M_k, L_k, x_k)$ and that if we denote $(f_{k+1}, x_{k+1}) = \mathsf{pert}_k(M_k, L_k, x_k)$, then $x_{k+1} \in \mathsf{pre}^*(\mathsf{STG}(f_{k+1}))(Y)$. As a result, $x'_{k'+1}$ is in $\mathsf{pre}^*(\mathsf{STG}(f_{k+1}))(Y)$. We know that there is no perturbation that can prevent Y to be reached in $\mathsf{STG}(\mathcal{S})$, because of the previous reasoning. Therefore, $x'_{k'+1}$ is in $\mathsf{pre}^*(\mathsf{STG}(\mathcal{S}))(Y)$ and x is in the same set. □

A set of sequences can correspond to an inevitable strategy if it verifies property 3 (below). We denote the set of sequences P:

$$\begin{aligned}\mathcal{P} = \{ & S_1 = [(\mathsf{pert}^1_1, (M^1_1, L^1_1, x^1_1)), \dots, (\mathsf{pert}, (M^1_{k_1}, L^1_{k_1}, x^1_{k_1}))] \\ & S_2 = [(\mathsf{pert}^2_2, (M^2_1, L^2_1, x^2_1)), \dots, (\mathsf{pert}, (M^2_{k_2}, L^2_{k_2}, x^2_{k_2}))] \\ & \dots, \\ & S_m = [(\mathsf{pert}^m_{k_m}, (M^m_1, L^m_1, x^m_1)), \dots, (\mathsf{pert}, (M^m_{k_m}, L^m_{k_m}, x^m_{k_m}))]\}\end{aligned}$$

With all S_i verifying $L^i_1 = \emptyset$, and pert^i_j among pert_I, pert_P, and pert_{-1} (in which case M is a set of states instead of a perturbation valuation).

Property 3. *Given a set of initial states X, a set of targets Y and a set of sequences P, P corresponds to an inevitable strategy if:*

- $\forall i \in [1, m], \forall(\mathsf{pert}_1, (M, L, x)) \in S_i$ *if* $\exists j \in [1, m]$ *such that* $\exists(\mathsf{pert}_2, (M', L, x)) \in S_j$ *then* $\mathsf{pert}_1 = \mathsf{pert}_2$ *and* $M = M'$.
- $\forall x \in X, x \in \mathsf{bas}(\mathsf{STG}(f), Y \cup \{x \mid \exists i \in [1, m], \exists M^i_1$ *such that* $\mathsf{pert}^i_1(M^i_1, L^i_1, x) \in S_i\})$.
- $\forall j \in [1, m], \forall(\phi_{L'}(f), x') = \mathsf{pert}^j_i(M^j_i, L^j_i, x^j_i), x' \in \mathsf{bas}(\mathsf{STG}(\phi_{L'}(f)), Y \cup \{z \mid \exists h \in [1, m], \exists M^h$ *such that* $\mathsf{pert}(M^h, L', z) \in S_h\})$.

As for the previous property, to prove this, we need to construct the correct strategy $\mathcal{S}$ from sequence S. We start from an "empty" strategy where the only possible action is $\mathrm{nothing}$. We will use a procedure similar to the one used for the proof of property 2, but which caters greater needs:

- We denote $A = \{x_1^1, x_1^2, \dots, x_1^m\}$
- For each i,
  - we find the highest j such that each x in X is in $\mathsf{bas}(\mathsf{STG}(f), Y \cup ((A \setminus \{x_1^i\}) \cup \{x_j^i\}))$.
  - $A = (A \setminus \{x_1^i\}) \cup \{x_j^i\}$, and we add $(\emptyset, x_j^i) \mapsto (\mathsf{pert}_j^i, M_j^i)$ to $\mathcal{S}$.
- In the same fashion, for each $j \in [1, m]$, for each $(\phi_{L'}(f), x') = \mathsf{pert}_i^j(M_i^j, L_i^j, x_i^j)$
  - We denote $B = \{z \mid \exists h \in [1, m], \exists M^h$ such that $\mathsf{pert}(M^h, L', z) \in S_h\}$
  - For each h such that $\exists x_b^h \in B$
    - $*$ We find the highest c such that x' is in $\mathsf{bas}(\mathsf{STG}(\phi_{L'}(f)), Y \cup ((B \setminus \{x_b^h\}) \cup \{x_c^h\}))$
    - $*$ $B = (B \setminus \{x_b^h\}) \cup \{x_c^h\}$, and we add $(L', x_c^h) \mapsto (\mathsf{pert}_c^h, M_c^h)$
- Lastly, for each i, we add $(L_{k_i}^i, x_{k_i}^i) \mapsto (\mathsf{pert}_{k_i}^i, M_{k_i}^i)$ to $\mathcal{S}$.

We know that each step of this procedure is possible thanks to property 3.

Proof. We want to prove that $\mathcal{S}$ is inevitable. This means that we want to prove that all x in X are in $\mathsf{bas}(\mathsf{STG}(\mathcal{S}), Y)$. We know that each x in X is in $\mathsf{bas}(\mathsf{STG}(f), Y \cup A)$ with A as constructed above. Moreover, we know that we took the highest j possible for each sequence i. In $\mathsf{STG}(\mathcal{S})$, some x might not be in $\mathsf{bas}(\mathsf{STG}(\mathcal{S}), Y \cup A)$, but they will be in the basin of a bigger set $Y \cup A \cup C$, with C being a set of x from perturbations allowed by $\mathcal{S}$, preventing some normal transitions because the action $\mathrm{nothing}$ was not used.

For each state in $A \cup C$, we can apply a perturbation, thus we can denote $(\phi_{L'}(f), x') = \mathsf{pert}_i^j(M_i^j, L_i^j, x_i^j)$. We know that x' is in $\mathsf{bas}(\mathsf{STG}(\phi_{L'}(f)), Y \cup B)$ with B as constructed above. As previously, in $\mathsf{STG}(\mathcal{S})$, we might need a bigger set $Y \cup B \cup D$, with D a set of x from perturbations allowed by $\mathcal{S}$. As previously, this is not problematic. Moreover, by prop. 1, we have that all x in X x is in $\mathsf{bas}(\mathsf{STG}(\mathcal{S}), Y \cup B \cup D)$. Recursively, we repeat this steps, replacing $A \cup C$ by $B \cup D'$ each time, where D' is D from which some x have been removed, as explained below.

We now need to prove that this recursion ends. If $D = \emptyset$ then we know that for all $i \in [1, m]$, the index of the perturbation can only increase, as proved for prop. 2. Thus, we will reach a time where $B = \emptyset$, stopping the recursion. If D contains x^i_j, then if $j > c$ (with c as defined in the procedure), it only speeds up the recursion. If $j < c$ then this will create a loop: we already verified that x is in the basin of a set containing $\chi_{M^i_j}(x^i_j)$, by recursion hypothesis. Thus, we remove x^i_j from the recurrence, allowing it to end. $\square$

Size of a strategy The size of a strategy is the sum of the size of all the possible perturbations.

"One-step" reprogramming strategy: We speak about *"one-step" reprogramming strategy* when $\mathcal{S}$ only allows perturbations when $L = \emptyset$. Often, the perturbations are only allowed in the initial state as an other condition on $\mathcal{S}$. We will speak of sequential reprogramming sequences when comparing with "one-step" reprogramming strategies when a distinction should be made.

2.2.3 Reprogrammability

A Boolean network f is inevitably (resp. existentially) reprogrammable from X to Y if and only if it admits some reprogramming strategy $\mathcal{S}$ among all possible strategies $\mathbb{S}$, which is inevitable (resp. existential) from X to Y.

Definition 22 (existential reprogrammability from X to Y). *A set of states X has existential reprogrammability to Y if:*
$\exists \mathcal{S} \in \mathbb{S}$ such that $\mathcal{S}$ is existential from X to Y.

Definition 23 (inevitable reprogrammability from X to Y). *A set of states X has inevitable reprogrammability to Y if:*
$\exists \mathcal{S} \in \mathbb{S}$ such that $\mathcal{S}$ is inevitable from X to Y.

We also say that there exists one-step reprogrammability (existential or inevitable) if $\mathcal{S}$ is a one step reprogramming strategy.

Remark: Without restrictions on perturbation valuations, existential and inevitable reprogrammability for any X to any state y in f is trivial, because we can use the perturbation valuation $M = \{v = y_v \mid v \in [1, n]\}$ in the sequence $[(\text{pert}, (M, \emptyset, x))]$ for every x in X. Most of the time, we are interested in minimal perturbation strategies, or strategies that result in perturbation sequences that are minimal. Finding reprogramming strategies with constraints on the number of perturbations, on the states in which the perturbations take place, or on the perturbation valuations is also interesting, to have a bigger set of solutions to choose from. The constraints on the perturbation valuations can be that some variables cannot be perturbed, or that they can only be perturbed to a value, or that the perturbation valuations are not allowed to be too big.

2.3 State of the art

While cell reprogramming is quite a new field, there are a lot of techniques that have been found to improve it. Induced pluripotent human stem cells were found in 2007 by Takahashi and Yamanaka [TTO$^+$07], but earlier research already show possible reprogrammings [KFG95, XYFG04]. Since 2007, there have been multiple improvements and discoveries, summed up in [TY16] from 2016.

In literature, we can find that reprogrammability for biological systems has mostly been done for one-step reprogrammability, as shown in subsection 2.3.1. However, sequential reprogrammability research and experiments can be found as well, and are explained in subsection 2.3.2. On the other hand, general sequential reprogrammability can be seen as a two player game, where one player is the system and the other is the experimenter. This is discussed, with comparison to literature, in subsection 2.3.3.

2.3.1 One-step reprogramming strategies

Since cell reprogramming as an experimental protocol often uses a single perturbation of multiple transcription factors, one-step reprogramming strategies are adapted to this context. To find these transcription factors, a wide array of methods are used, with different constraints and complexity.

For experimental verification and predictions from known results, model-checking is often used, for example by Mendoza in 2006 or Hermann et al. in 2012 [Men06, HGZ$^+$12]. These works are often performed by biologists or people very close to the experiments, and aim at creating a model and checking known results, experimental results, and intuitions. After constructing the model, trajectories from an initial state are explored, to show closeness to the reality. Perturbations of the initial state, by changing the function and the value of some variables can also be tested, showing how the trajectories are modified and which new trajectories are found. In [Men06, HGZ$^+$12] the authors have exploited and constructed a Boolean network.

Probabilities are a widely used tool for cell reprogramming, either as possible outcomes of the perturbations or in the model itself. The computation of the results can be probabilistic, computing the probabilities to reach all possible

attractors from a initial state, as used by Sahin et al. in 2009 or by Cohen et al. in 2015 [SFL$^+$09, CMR$^+$15]. With this kind of tool, the method is close to the model-checking approach, where the perturbations are modelled by a change in the model, but returns more detailled results, that can be used for statistical studies.
The model itself can be probabilistic, as the ones created by Chang et al. in 2011 or Hannam et al. in 2016 [CSW11, HAK16], where each variable has a probability of changing value depending on other variables and external factors.

An other kind of model is epigenetic landscape, an example of which is built by Lang et al. in 2014 [LLCM14]. It aims at recreating the dynamics of the differentiation, by creating a landscape, were valleys are attractors and cell fates, and differentiation goes downward towards the valleys. From this, cell reprogramming can be studied by finding how to go over the hills and let the system stabilize in the target attractor.

A less common method is a static analysis of the model, using it to predict which genes to target, as used by Crespo et al. in 2013, or by Zañudo and Reka in 2015 [CPJdS13, ZA15]. By analyzing the network and finding patterns in it, it is possible to find targets for the perturbations. Most of the time, simulations are done with these targets to verify that all behaviors lead to the wanted attractor. In this thesis, we will formally analyze this static approach, to show what are the benefits and the limits.

In [BD17] by Biane and Delaplace, two kinds of perturbations are used, permanent perturbations equivalent to the ones described in def. 13, and a special kind of function perturbation where a literal of the function is deleted, thus changing the dynamics. This method uses integer linear programming to infer the perturbations. Moreover, their initial state is the whole transition graph.

2.3.2 Sequential reprogrammability

However, even if the majority of predictions are done with one-step reprogrammability in mind, sequential reprogrammability has been studied as well. Sometimes, as an improvement of one-step reprogrammability, where timing yields better results, sometimes as a general case which includes one-step perturbation sequences.

As with one-step reprogrammability, model-checking can be used. Perturbations can be allowed only in attractors, or at any state of the system. When perturbations are only in attractors, one can compute all possible one-step perturbation sequences to go from any attractor to any other attractor, and use the attractors as intermediate states, a method used by Abou-Jaoudé et al. in 2015 [AJMN$^+$15]. These computations are made by testing all possible perturbations and testing reachability from the new perturbed state. When perturbations are allowed in every state, it is possible to reduce the set of allowed perturbation valuations to reduce computation times, as shown by Samaga et al. in 2010 in [SVKK10]. In the case of [SVKK10], the updates of the system are synchronous.

As previously, an epigenetic landscape can be created, to find which genes are linked to which phenotypes, in order to perturb the genes the most linked to the target attractor. In the model of Ronquist et al. from 2011 [RPM+17] this is used with time-series and allows for time-dependent perturbations, where the efficiency is better when some factors are introduced later.

2.3.3 Game theory

We can translate the reprogrammability to a game, where player 0 is the Boolean network, and player 1 controls the perturbations. In this context, inevitable reprogrammability becomes a reachability game, where we want to find the solutions for player 1. Creating an game arena from the perturbed transition graph described in chapter 4 is quite easy, and therefore, reachability in this game can be dealt with in the usual way, described in the book from Grädel et al. in 2003 [GTW03]. However, the reprogrammability problem itself is trivial if there is no maximum number of perturbations. Reachability games will return solutions under a bound, and the shortest paths, but they will fail to return shortest perturbation sequences.

2.4 Contributions of this thesis

In this thesis, we explain how static analysis of the Boolean network's interaction graph allows to find reprogramming strategies. We expand upon our results from 2016 in [MHP16] in the light of sequential reprogramming. We show that some parts of the interaction graph are highly influential on the dynamics, the strongly connected components (SCC), and that perturbing some of them in the correct order allows for an inevitable reprogramming strategy. Reprogramming a smaller set of these components with a less strict order allows an existential reprogramming strategy. We also think that to find the complete set of reprogramming strategies, an analysis of the dynamics of the whole Boolean network is required. This is discussed in chapter 3.

To perform an analysis of the dynamics, we use two approaches. The first one, from [MHP17, MSH+19], consists of having minimal constraints on the perturbation valuations and where the perturbation takes place, resulting in a strategy $\mathcal{S}$, creating $\mathtt{STG}(\mathcal{S})$, and analyzing this graph to find smaller reprogramming strategies that are inevitable or existential, whichever is needed. We created a model to compute $\mathtt{STG}(\mathcal{S})$, and two algorithms to extract a simplified reprogramming strategy from it.

The second approach consists of only allowing perturbations in attractors, resulting in much smaller computation times. In this approach, from [MSP+19], we use a modified Hamming distance to compute quickly the perturbations required to reach the target attractor from the initial attractor, and we then repeat this procedure to use other attractors as intermediate steps.

These two approaches are further discussed in chapter 4.

Chapter 3

Network Static Analysis

A first approach to Boolean Network reprogrammability, is to find the perturbations from the Boolean network f and its interaction graph $G(f)$ (which can easily be computed from f), without computing the transition graph. To be able to reprogram the system, we also need a starting point, the initial state $\mathsf{initial}$, and a target target. In this chapter, we will discuss how to find strategies or perturbation sequences for inevitable and existential reprogrammability with permanent, temporary or instantaneous perturbation sequences, with limited computation of the system dynamics. We will use the terms short-term temporary perturbation to describe a temporary perturbation which only last a few updates, and long-term temporary perturbations to describe temporary perturbations lasting until an attractor is reached.

This chapter contains published results from [MHP16], which have been extended upon in the light of sequential reprogrammability. In this chapter, we give an algorithm that results in clusters of variables to perturb, namely some strongly connected components of the interaction graph, and when to perturb them. We also prove that to find more specific variables, the system cannot be divided in subparts studied individually, and should be studied as a whole, and that dynamics should be taken into account. The detailed study of dynamics is explained in chapter 4.

3.1 Computations on the Interaction Graph and properties on $\mathsf{initial}$ and target

Comparing $\mathsf{initial}$, the initial state, and target, the target state, already gives a reprogramming of the system, even if it is the least efficient one, by looking at all variables which have different values between the two states. For this perturbation sequence, the perturbation valuation is the set $M = \{(v = \mathsf{target}_v)$ with v verifying $\mathsf{initial}_v \neq \mathsf{target}_v\}$. This perturbation sequence corresponds to an inevitable reprogramming strategy, as applying $\mathsf{pert}(M, \emptyset, \mathsf{initial}) = (\phi_{M'}(f), \mathsf{target})$. Instantaneous, temporary and permanent perturbations will all ensure such property.

We will suppose that target is in an attractor, as properties on the interaction graph rely on attractors.

3.1.1 Using the cycles of the Interaction Graph

The first rule of Thomas (theorem 1) gives useful information on the attractors of f.

Theorem 1 (Thomas' first rule)**.** *If $G(f) = (V, E)$ has no positive cycles, then f has at most one attractor.*
Moreover, if f has two distinct fixed points x and y, then $G(f)$ has a positive cycle $C \subseteq V$ such that $x_v \neq y_v$ for every variable v in C.

This conjecture, made by René Thomas[Tho73] in 1973, has been demonstrated for Boolean and discrete networks[Ara08, RRT08] in 2008.

If $G(f)$ has no positive cycle, and since target is in an attractor, then initial always reach target, without any perturbation needed, because of the first point of the theorem. Therefore initial is in $\mathsf{bas}(\mathsf{STG}(f), \mathsf{target})$, resulting in inevitable reprogrammability being trivial (def. 23) in this case.

However, if $G(f)$ has positive cycles, then it can have multiple attractors. In the completely asynchronous dynamics, to have a complex attractor which is not a fixed point, $G(f)$ has to contain a negative cycle:

Theorem 2 (Thomas' second rule)**.** *If f has a cyclic attractor, then $G(f)$ contains a negative cycle.*

This theorem is derived from the second rule of Thomas, and is proved for Boolean networks[RRT08].

With this theorem, we can deduce than perturbing all cycles of the network would probably allow target to be reached. However, in order to prove it, and to find which kind of perturbation to make and which kind of reprogrammability to expect, we need to order the graph.

3.1.2 Ordering the Interaction Graph

Theorems 1 and 2 give an intuition on the parts of the graph which have a strong influence on the dynamics: the cycles. Indeed, we know that the value of a variable v of the Boolean Network f depends on f_v. In terms of Interaction Graph, it means that the value of v is determined by its direct predecessors, by the definition of Interaction Graph (def. 3). By recursion, we can deduce that the value of v depends on the value of all predecessors in $G(f)$, the set $\text{pre}^+(G(f))(v)$. From this, we can also deduce that if all predecessors of a variable have the same value as in a fixed point, then the variable will always have the value it has in the fixed point. The same kind of reasoning can be applied if the value are the ones in a complex attractor, the set $\text{pre}^*(G(f))(u)$ will keep values in the same attractor. The set of all predecessors including the variable are locally stable.

Theorem 3 (Local stability)**.** *Let f be a Boolean network of size n, and y be a state.*
$\forall z \in \{0,1\}^n, \forall v \in [1,n],$ *if* $\forall u \in \text{pre}^+(G(f))(v),\ z_u = y_u$
then $\forall u \in \text{pre}^*(G(f))(v), f_u(z) = f_u(y)$.
If y is a fixed point, the second part of the implication can be refined to $\forall u \in \text{pre}^*(G(f))(v), f_u(z) = y_u$.
If y is part of a more complex attractor C, the second part of the implication can be refined to $\forall u \in \text{pre}^*(G(f))(v), \exists c \in C, f_u(z) = c_u$.

Proof. Let z be a state in $\{0,1\}^n$, and v a variable of f. We suppose that for all u predecessor of v, excluding v if it is not a predecessor of itself, in $G(f)$, $z_u = y_u$. Then, by construction of the Interaction Graph, $f_v(z) = f_v(y)$, since the values of $\text{pre}(G(f))(v)$ are the same in y and z. This can be generalized to all variables in $\text{pre}^*(G(f))(v)$, for the same reason, all values of the predecessors are the same in y and z. Hence, for all u in $\text{pre}^*(G(f))(v)$, we have $f_u(z) = f_u(y)$. □

Moreover, if y is a fixed point, then for all u variable, $f_u(y) = y_u$. With this fact and $f_u(z) = f_u(y)$, we have that for all u in $\text{pre}^*(G(f))(v)$, $f_u(z) = y_u$. □

Finally, if y is part of a more complex attractor C, then, by definition of attractor, for every state c in the attractor C and every variable i, there exists c' in C such that $f_i(c) = c'_i$. Since y is in C, then for all variables, including u, there exists $c \in C$ such that $f_u(y) = c_u$. Since we know that $f_u(z) = f_u(y)$, we have $f_u(z) = c_u$. □

This confirms our intuition: a variable inside a cycle has a behavior hard to predict, since it will influence itself, and its influence on its successors will also influence its predecessors and itself. On the other hand, it shows that a variable outside a cycle is easier to reprogram, since it does not have this kind of feedback, perturbing the cycles in the predecessors is enough in most cases.

In order to hierarchize the candidate variables to perturb, we want to order the Interaction Graph. It allows us to find which variables impact strongly the network's dynamics, and which ones do not have much influence. However, cycles are a problem, and even more when they are interwoven. To solve this issue, we order the strongly connected components (SCCs) of the Interaction

Graph. Moreover, we can distinguish between trivial SCCs, which do not contain cycles, and complex SCCs, which contain one or more cycles. We can remark that all trivial SCCs are of cardinality 1 by definition of SCC: if there are more than 1 variables in a SCC, then by definition there is a cycle, and hence the SCC is not trivial.

We order the SCCs following any topological order. The local stability theorem (theorem 3) gives an intuition on which SCCs to perturb, especially when paired with the ordering. Indeed, if we suppose that a SCC C and all its predecessors have the same value as in target, then there are two possibilities.

First, if target is a fixed point, then the system will reach a state where all predecessors of C that are trivial SCCs have the same value as in target, by recursively applying the local stability. Second possibility is if target is in a more complex attractor. Then, if for all successors of target, C and all predecessors keep the same values, then we fall back to the first case. Lastly, if some predecessors can change value because they can change value in the attractor, then the local stability applies, and the predecessors keep values that are possible in the attractor. Since the system is asynchronous, there is no need to time the perturbation of the SCCs predecessors of C to have them be synchronous to their successors.
This is discussed in details and proved further in the next section.

Notations: We denote the set of all SCC as $\mathcal{C}$ and the set of all non-trivial SCCs as $\mathcal{C}^{NT}$.

3.1.3 Perturbing the Strongly Connected Components

One might wonder if ordering the SCCs always reduce the number of complex SCCs to perturb, and if perturbing the first SCCs that have different values than in target in each topological order is enough to have a perturbation sequence to target.

However, that is not the case. We can always end up in a different attractor than target if only the first SCCs with different values that target are perturbed, as shows example 5.

Example 5. *This example shows that perturbing a node in a SCC first in topological order is not enough to perturb the nodes below. In this example,* $\mathsf{target} = 10101$ *and* $\mathsf{initial} = 01100$*, both of which are fixed points. Perturbing the first SCC, the set of variables* $\{1, 2, 3\}$ *in order to have the same value as in* target *results in* $\{4\}$ *never being activated, and thus* $\{5\}$ *never being activated, and* target *is never reached.*

$$\boxed{\begin{array}{l} f_1(x) = \neg x_2 \\ f_2(x) = \neg x_1 \\ f_3(x) = x_1 \lor x_2 \\ f_4(x) = x_2 \land \neg x_3 \\ f_5(x) = x_4 \lor x_5 \end{array}}$$

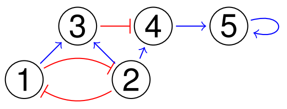

Figure 3.1: Boolean network preventing changes in the lower SCC

Inspired by this counter-example, we deduced that almost all SCCs need to be perturbed. Which SCCs are perturbed will depend on the kind of reprogrammability desired. However, the sources, variables which have a constant function and thus, no incoming edges in the interaction graph, can be a huge issue. If the perturbations are instantaneous, and there exists a source s which verifies $\mathsf{initial}_s \neq \mathsf{target}_s$ then only existential reprogramming strategies can be found, since s will always come back to its value. Moreover, even if the system reaches target, it will not stay in it. If the perturbations are temporary, reprogramming strategies exist, but the system will not stay in target after the perturbations stop. If the perturbations are permanent, reprogramming strategies exist as well. In all cases, all sources $\{s\}$ are added to $\mathcal{C}^{NT}$, as they behave in the same fashion as a self-loop for reprogrammability with permanent and temporary perturbations, and are required in all cases.

Existential reprogrammability: It is usually easier to find reprogramming strategies for existential reprogrammability (def. 22) than for inevitable reprogrammability (def. 23). In order to be existential, the strategy only needs to perturb the SCCs that contain a variable v with $\mathsf{initial}_v \neq \mathsf{target}_v$. In this case, the kind of perturbation (instantaneous (def. 12), temporary, or permanent (def. 13)) does not matter much, because a path can be built with the variable never being updated.

Property 4. *Let* $M = \{(v = \mathsf{target}_v) \mid v$ *verifies* $\mathsf{initial}_v \neq \mathsf{target}_v$ *and* $\exists C \in \mathcal{C}^{NT}$ *such that* $v \in C\}$. $\mathsf{pert}(M, f, \mathsf{initial}) = (f', x)$ *where* $x \in \mathsf{pre}^*(\mathsf{STG}(f))(\mathsf{target})$.

Proof. Applying M to $\mathsf{initial}$ results in a state x and a Boolean network f'. Here, we are only interested in the trivial SCCs $\{v\}$ which verify $\mathsf{initial}_v \neq \mathsf{target}_v$. We know that $\mathsf{pre}^*(G(f))(v) \neq$ because sources are transformed in self loops, and we know that $v \notin \mathsf{pre}^+(G(f))(v)$ because $\{v\}$ is a trivial SCC. Therefore, we can apply theorem 3 to all these trivial SCCs in a topological order. Since all variables inside non trivial SCCs have the same values in x as in target, applying local stability in this specific order results in target being reached, thus $x \in \mathsf{pre}^*(\mathsf{STG}(f))(\mathsf{target})$. $\square$

Inevitable reprogrammability: In order for $\mathsf{initial}$ to be inevitably reprogrammable (def. 23) to target, the strategy requires more SCCs to be perturbed than for the existential strategy. We need to add the successors of the variables perturbed by the existential reprogramming strategy.

Property 5. *Let $V_0 = \{v \mid \mathsf{initial}_v \neq \mathsf{target}_v$ and $\exists C \in \mathcal{C}^{NT}$ such that $v \in C\}$. Let $V_1 = \{v \mid v \in \mathsf{post}^*(G(f))(V_0)$ and $\exists C \in \mathcal{C}^{NT}$ such that $v \in C\}$.*
Let $M = \{(v = \mathsf{target}_v) \mid v \in V_1\}$ $\mathsf{pert}_P(M, f, \mathsf{initial}) = (f', x)$ where $x \in \mathsf{bas}(\mathsf{STG}(f'), \mathsf{target})$.

Proof. V_0 is the set of non-trivial SCCs that have at least one variable with a different value in $\mathsf{initial}$ and target. V_1 is the set of all non-trivial SCCs successors of V_0, including itself. M is the perturbation valuation coupling each variable v from V_1 with target_v. Let x be the state resulting from the perturbation M in $\mathsf{initial}$, and f' is the new Boolean network. For every variable in the new network f', either the variable has no predecessor and has a constant function (if the variable was perturbed) or theorem 3 can be recursively applied in a topological order. As a result, all non-trivial SCCs will be updated to reach the value in target. □

Long-term temporary perturbation can be used in the same way: the perturbation has to hold until the system reaches an attractor, and the proof is the same.

For $\mathsf{initial}$ to be inevitably reprogrammable to target with instantaneous perturbations, the perturbations need to be done in a sequence, following a topological order. Moreover, if there are sources in the interaction graph, then inevitable reprogrammability with instantaneous perturbations is never achieved. If it is not the case, more SCCs are perturbed than the permanent perturbations case because we do not know how stable the system is in $\mathsf{initial}$.

Property 6. *Let $\mathcal{C}^{NT} = \{C_1, \ldots, C_k\}$ in topological order. For i from 1 to k, let $M_i = \{(v = \mathsf{target}_v) \mid v \in C_i\}$. $\mathsf{initial}$ is inevitably reprogrammable to target in f. Using the perturbation valuations in the list $[M_1, \ldots, M_k]$ with instantaneous perturbations in attractors is an inevitable reprogramming strategy.*

Proof. The first perturbation has to be done in $\mathsf{initial}$. All paths will end up in attractors. Let $\{A_1, \ldots, A_m\}$ be the set of reachable attractors from $\chi_{M_1}(\mathsf{initial})$. For all $i \in [1, m]$, there exists a state $a \in A_i$ such that for all $c \in C_1$, $a_c = \mathsf{target}_c$, by theorem 3. Moreover, all trivial SCCs $\{v\}$ after C_1 and before the next non trivial SCC C_i verify $a_v = \mathsf{target}_v$ by theorem 3. Perturbing the system in a state a from an attractor A with M_2 ($\mathsf{pert}_I(M, f, a) = (f', a')$) results in a', which also verifies $a'_v = \mathsf{target}_v$ for C_1 and all non trivial SCCs after it and before the next non-trivial SCC.

Recursion:
Let $i \in [2, k]$ be an integer, we suppose that all perturbations with perturbation valuation M_j have been done for all $1 \leq j < i$. Let a be a state of an attractor A reached after the previous perturbation, we have $\mathsf{pert}_I(M_i, f, a) = (f', a')$. By recursion, we know that for all $1 \leq j < i$, C_j verifies $\forall c \in C_j, c_j = \mathsf{target}_j$. By recursion, we also know that for all trivial SCCs after any C_j with $1 \leq j < i$ but before any C_j with $i \leq j < k$, the same property holds. Let b be a state in a reachable attractor B from a'. Since M_i perturbs C_i, by theorem 3, for all $c \in C_i$, we have $b_c = \mathsf{target}_c$. Therefore, since b is in an attractor, all trivial SCCs $\{v\}$ after C_i but before the next non-trivial SCC verify $b_v = \mathsf{target}_v$. Applying the perturbation M_{i+1} to b results in a state b' where these properties are still true. □

This property is even true with perturbations done before an attractor is reached, but the right part of the system (the non-trivial SCC and the successors trivial SCCs before the next non-trivial SCC) needs to be stable. Short-term temporary perturbations, where the perturbations stop before an attractor is reached can be used instead of instantaneous perturbations.

3.2 Adding attractors information

In discrete models for biology, often all the attractors are computed when possible, to show how close the model is to reality. This allows us to study the same problem as in the previous section, but with extra information, $\mathcal{A}(f)$, the list of attractors of the Boolean network f. Indeed, knowing all attractors allows us to compare them, and compare the values of the complex SCCs, and to know in which attractor target is. When target is not a fixed point, we define target' as the state inside the attractor containing target which has the least variables in non-trivial SCCs whose values differ from $\mathsf{initial}$, and we use target' instead of target for all computations. If multiple such states exist, a random one is chosen.

Knowing in which attractor is target is the only improvement we found in the case of existential reprogrammability. Knowing in which attractor the perturbed system may end up is not necessary, and the set described in property 4 is the smaller we found.

Therefore, the following algorithms are made for inevitable reprogrammability with permanent (or long-term temporary) perturbations. A quick explanation on how to use them with instantaneous (or short-term temporary) perturbations is given after each algorithm.

Boolean Network

$$\boxed{\begin{array}{l} f_1(x) = x_1 \\ f_2(x) = x_2 \\ f_3(x) = x_1 \wedge \neg x_2 \\ f_4(x) = x_3 \vee x_4 \end{array}}$$

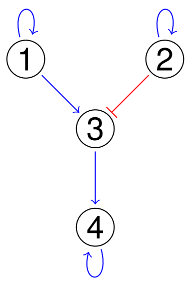

Figure 3.2: Boolean network f and its interaction graph

3.2.1 A "one-step" algorithm

Example 6 (below) shows how the list of all attractors can be used to reduce the number of SCCs to perturb.

Example 6. *In the context of the Boolean network f, we want* initial $= 0000$ *to be inevitably reprogrammable to* target $= 1011$*, which is in an attractor (as indicated at the very beginning of the chapter, static analysis of the graph and network relies on this fact). f and its interaction graph are shown in fig. 3.2. By only using the methods of the previous section, we can deduce from the interaction graph an order on the SCCs, and which SCCs are trivial. In this example, all SCCs are set of size 1, and are $\{1\}, \{2\}, \{3\}, \{4\}$, and only $\{3\}$ is trivial. Moreover, the variables are already in a topological order. Since there are positive cycles, the self-loops on $\{1\}, \{2\}$ and $\{4\}$, we need to perturb the system. By the properties of subsection 3.1.3, we know that perturbing $\{2\}$ is not needed. Thus, we can deduce that $\{1\}$ and $\{4\}$ need to be perturbed in our strategy to make it inevitable from* initial *to* target.

However, if new information is given, the list of all attractors, then we can reduce the number of variables to perturb. The set of all attractors, which are all fixpoints, of the system is:

$$\{\{0000\}, \{0001\}, \{0100\}, \{0101\}, \{1011\}, \{1100\}, \{1101\}\}$$

Since $\{1\}$ has no ancestor other than itself and has a different value in initial *and* target*, we know we need to perturb it. For $\{4\}$, a close look at the list of attractor reveals that if $\{1\}$ has the value 1 and $\{2\}$ has a value 0, then only one attractor contains such values, $\{1011\}$, from which we can deduce that $\{4\}$ does not need to be perturbed.*

Inspired from this example, we design algorithm 1 to find which SCCs to perturb.

Algorithm explanation: Algorithm 1 searches for inevitable reprogramming strategies to target with permanent perturbations, and returns a perturbation sequence (of one perturbation), based on property 5, which corresponds to an inevitable strategy (see prop 3). The inputs are f, the Boolean network, initial the initial state, target the target, and $\mathcal{A}(f)$ the list of attractors of f. The first step is to compute the interaction graph $G(f)$, as most computations are done on it. If there is no positive cycle in the interaction graph, there is no need to perturb the system. If not, the list of non-trivial SCCs is computed, and ordered in a topological order. A list of the SCCs to perturb, SCC_list is initialized to the empty list. Then, for all non-trivial SCC C taken in topological order:

- If no predecessor of C is in SCC_list, meaning there is no SCC perturbed in the predecessors of C, then if there is a variable v inside C that verifies $\text{initial}_v \neq \text{target}_v$, C is added to SCC_list.
- If a predecessor of C is in SCC_list, then we need to compare the attractors.

To compare the attractors, we look at each state of each attractor that verifies that the variables in $\text{pre}^+(G(f))(C)$ have the same values as target. If this attractor has the same values in every variable of C than target has, then there is no need to perturb C, as it will always take the right value after some updates. Otherwise, C needs to be perturbed, and is added to SCC_list as a consequence. Lastly, the SCC_list is translated to a perturbation valuation M, and the perturbation sequence containing the perturbation from initial with the valuation M is returned.

Instantaneous perturbations: If the perturbations are instantaneous, the perturbation sequence returned by the algorithm can not happen, allowing the system to never reach target. To ensure such property, we need to verify that all predecessors (instead of only non-trivial SCCs) have the correct value and none are perturbed. As a result, the condition "$\forall v \in \text{pre}^*(G(f))(C), v \notin$ SCC_list" becomes "$\forall v \in \text{pre}^*(G(f))(C), v \notin$ SCC_list and $\text{initial}_v = \text{target}_v$". Moreover, the algorithm needs to return a perturbation sequence with multiple steps as described in prop. 6. To do so, the last lines are modified as shown in sub-algorithm 2, with a sequence created before the For loop, and a modification to the loop, where the sequence is created inside the loop. In this case, the state in which the perturbation needs to be done is not explicit, it is any state of a reached but unknown attractor.

Complexity: For a Boolean network f of n variables and m attractors, algorithm 1 runs in $O(n^3 \times m)$ in the worst case. The first For loop runs in the number

Algorithm 1 Reprogrammability by static analysis and knowledge of attractors

```
1:  function DIRECT_REPROGRAMMABILITY($f$, initial, target, $\mathcal{A}(f)$)
2:      $G(f) = \text{interaction\_graph}(f)$
3:      if has_no_positive_cycle($G(f)$) then
4:          Return $\emptyset$
5:      SCC = non_trivial_strongly_connected_components_ordered($G(f)$)    ▷ This function is not described here, but is quite easy. Multiple algorithms exist that return an ordered list of all SCCs, from which removing trivial SCC is easy
6:      SCC_list = []
7:      for $C \in$ SCC do
8:          if $\forall v \in \text{pre}^*(G(f))(C), v \notin$ SCC_list then    ▷ To replace with "$\forall v \in \text{pre}^*(G(f))(C), v \notin$ SCC_list and $\text{initial}_v = \text{target}_v$" for instantaneous perturbations
9:              if $\exists c \in C, \text{initial}_c \neq \text{target}_c$ then
10:                 SCC_list.$add(C)$
11:         else
12:             for $A \in \mathcal{A}(f)$ do
13:                 for $a \in A$ do
14:                     if $\forall v \in \text{pre}^+(G(f))(C), a_v = \text{target}_v$ then
15:                         if $\exists c \in C, a_c \neq \text{target}_c$ then
16:                             SCC_list.$add(C)$
            ▷ In the case of instantaneous perturbations, replace the following lines with sub-algorithm 2
17:     $M = \{\}$
18:     for $C \in$ SCC_list do
19:         for $v \in C$ do
20:             $M.add((v = \text{target}_v))$
21:     Return [$\text{pert}_P(M, f, \text{initial})$]
```

Algorithm 2 Replacement in case of instantaneous perturbations

```
$s = []$
for $C \in$ SCC_list do
    $M = \{\}$
    for $v \in C$ do
        $M.add((v = \text{target}_v))$
    if $s = []$ then
        $s.add(\text{pert}_I(M, f, \text{initial})$
    else
        $s.add(\text{pert}_I(M, f, \cdot)$
Return $s$
```

of non trivial SCCs, which can be as many as n if they are all self loops. The If checks again all predecessors, which in this case is between 0 and n, and can fail, entering the Else. In this Else, all attractors are checked, which is m attractors, and all predecessors are tested for equality, which is n again. This means that the algorithm depend on the number of attractors in the network, which can range from 1 to e^n. However, optimizations can be done on the list of attractors, by removing some of them that are not interesting because their first SCCs in different topological orders have variables with different values than target.

3.2.2 A more complex, sequential algorithm

Using the same toy model, we can see that algorithm 1 still does not always returns the smallest SCC set to perturb. Example 7 explains a case where a smaller solution set can be found.

Example 7. *With the same Boolean network f, shown with its interaction graph in fig. 3.2, we now want the same initial state,* initial $= 0000$ *to be inevitably reprogrammable to a new target,* target $= 1101$*. Since both* 1100 *and* 1101 *are in attractors, the previous algorithm would return the following permanent perturbation sequence* $[(\text{pert}_P, (\{(x_1 = 1), (x_2 = 1), (x_4 = 1)\}, \emptyset, \text{initial}))]$*. However, a smaller perturbation sequence exists,* $[(\text{pert}_P, ((x_1 = 1), \emptyset, \text{initial}); (\text{pert}_P, ((x_2 = 1), (x_1 = 1), 1011))]$*.*

$[(\text{pert}_P, ((x_1 = 1), \emptyset, \text{initial})]$ *is an inevitable reprogramming strategy from* initial $= 0000$ *to* 1011*, as shown in ex. 6. Next, from* 1011*,* $[(\text{pert}_P, ((x_2 = 1), (x_1 = 1), 1011))]$ *is an inevitable reprogramming strategy to* target $= 1101$*.*

To find this kind of solution set, we need our algorithm to include sequentiality. The idea is to use other attractors as intermediate steps to reach target.

We use algorithm 1 to compute which SCCs to perturb in each step. However, since attractors can be complex, we only keep fixpoints as intermediate steps. This point is discussed in the remarks at the end of the subsection.

Algorithm 3 explanation: Algorithm 3 recursively computes the perturbations to do to reach target from all single state attractors of the system, with a number of perturbations growing at each repetition.

It takes as inputs f, the Boolean network, initial the initial state, target the target, and $\mathcal{A}(f)$ the list of attractors of f. The first thing it does is to compute the direct reprogrammability from initial to target, using algorithm 1, and it computes the size of the perturbation sequence. This number, max_dist, will be used as a maximal value for all future computations. This algorithm only return sequences which are smaller. Note that a variable perturbed twice will be counted twice, this is on purpose, to represent the total size of the perturbation sequence. This first solution is added to solution, the list of all perturbation sequences from initial to target.

Next, the list of attractors we will work with is reduced to the list of single state attractors $\mathcal{A}_f$. The list of attractors that can be used as a step towards target is created at the same time, next_step. For each fixpoint a in $\mathcal{A}_f$, an empty list is created, solution_a, which is the set of all perturbation sequences from a to target. The direct reprogrammability from a to target is computed, and added to solution_a if its size is lower than max_dist. a is added to the list of possible next steps.

Then the algorithm adds new reprogrammability possibilities to all attractors. To do so, it does the following steps as long as long as next_step is not empty:

1. A new empty list is created, next_step_2 which has the same purpose as next_step
2. For all $a_1 \in \mathcal{A}_f$ and $a_2 \in$ next_step
    (a) the perturbation sequence s from a_1 to a_2 is computed, as well as its length d.
    (b) if there exists a perturbation sequence combining p and a perturbation sequence from solution_{a_2} that has a smaller length than max_dist:
    (c) a_1 is added to next_step_2
    (d) all sequences verifying this condition are added to solution_{a_1}
3. next_step replaces next_step_2, and the algorithm goes back to 1.

Algorithm 3 Sequential reprogrammability by static analysis and attractors

```
 1: function SEQUENTIAL_REPROGRAMMABILITY($f$, initial, target, $\mathcal{A}(f)$)
 2:     direct_sequence $=$ Direct_Reprogrammibility($f$, initial, target, $\mathcal{A}(f)$)
 3:     max_dist $=$ seq_size(direct_sequence)
 4:     solution $=$ [] ; solution.$add$(direct_path)
 5:     $\mathcal{A}_f$ $=$ fixpoints($\mathcal{A}(f)$)
 6:     next_step $=$ []
 7:     for $a \in \mathcal{A}_f$ do
 8:         solution$_a$ $=$ []
 9:         $l$ $=$ Direct_Reprogrammibility($f, a$, target, $\mathcal{A}(f)$)
10:         if seq_size($l$) $<$ max_dist then
11:             next_step.$add(a)$ ; solution$_a$.$add(l)$
12:     while next_step $\neq$ [] do
13:         next_step$_2$ $=$ []
14:         for $a_1 \in \mathcal{A}_f$ do
15:             for $a_2 \in$ next_step do
16:                 $s$ $=$ Direct_Reprogrammibility($f, a1, a2, \mathcal{A}(f)$)
17:                 $d$ $=$ seq_size($s$)
18:                 if $\exists$seq $\in$ solution$_{a_2}$, $d$ $+$ seq_size(seq) $<$ max_dist then
19:                     next_step$_2$.$add(a_1)$
20:                     for seq $\in$ solution$_{a_2}$ do
21:                         dist $=$ seq_size($s$) $+$ seq_size(seq)
22:                         if dist $<$ max_dist and $s \notin$ seq then
23:                             solution$_{a_2}$.$add(s.add$(seq))
24:         next_step $=$ next_step$_2$
25:     if $\exists a \in \mathcal{A}_f, a = \{$target$\}$ then
26:         solution $=$ solution$_a$ ; solution.$add$([direct_sequence])
27:     else
28:         for $a \in \mathcal{A}_f$ do
29:             if solution$_a$ $\neq$ [] then
30:                 $s_a$ $=$ Direct_Reprogrammibility($f$, initial, $a, \mathcal{A}(f)$)
31:                 $d$ $=$ seq_size($s_a$)
32:                 if $\exists$seq $\in$ solution$_a$, $d$ $+$ seq_size(seq) $<$ max_dist then
33:                     for se $\in$ solution$_a$ do
34:                         dist $=$ seq_size($s_a$) $+$ seq_size(se)
35:                         if dist $<$ max_dist and $s_a \notin$ seq then
36:                             solution.$add(s_a.add$(seq))
37:     Return solution
```

Algorithm 4 Length functions

```
1: function SEQ_SIZE($s$)
2:                ▷ Returns the size of a perturbation sequence with multiple steps
3:     Return $\Sigma_{\mathsf{pert}(M,f,\cdot)\in p}|\mathsf{M}|$
```

Once this is done, the possible perturbation sequences from all fixpoints to target are computed. We need to find how to reach this fixpoints from initial. If initial is in one of the fixpoints, the sequences from this fixpoint are returned, with the direct reprogrammability added. If not, the above procedure (3.a to 3.d) is repeated, with $a_1 = \mathsf{initial}$ and a_2 any attractor in $\mathcal{A}_f$, and the solutions are directly added to solution. Lastly, solution is returned.

Remarks: First, we can note that we use the number of perturbed variables as a limit for the iterations. Using the number of SCCs would have returned different SCCs, but ultimately, we are interested in having a minimal number of variables to perturb. An other option would have been to set it to infinite, to have the complete set of solutions.

Secondly, the algorithm is not optimal. Improvements could be made on this algorithm, by allowing cyclic attractors to act as intermediate steps. However, doing so requires Direct_Reprogrammibility to work with a set as initial states. It implies knowing more of the observability of the system, and includes a lot of different possibilities, making the algorithm much more complicated. A more general version algorithm is given in chapter 4.

3.3 Limitations of the SCC approach

Algorithm 3 returns solutions, and uses sequentiality to return more solutions than algorithm 1. It works great on networks with small SCCs, but when the SCCs are bigger, it raises the issue of finding which variables to perturb. Moreover, the results returned are not always minimal. As a result of these, we conclude that the dynamics have to be at least partially computed. We try to find necessary conditions on the variables to perturb or the order in which they are perturbed.

3.3.1 Main issues

The main issue is large networks, with big SCCs. These SCCs will contain a lot of variables, and we want a minimal number of variables to perturb.

$$\boxed{\begin{aligned} f_1(x) &= \neg x_3 \wedge \neg x_2 \\ f_2(x) &= \neg x_1 \\ f_3(x) &= \neg x_1 \\ f_4(x) &= x_2 \wedge \neg x_1 \wedge \neg x_3 \\ f_5(x) &= x_4 \vee x_5 \end{aligned}}$$

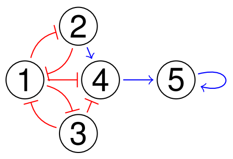

Figure 3.3: Boolean network where the feedback vertex set strategy is only existential

Trying to find a minimal set of variables to perturb in each SCC results in an important issue: inevitable reprogramming strategies can become existential reprogramming strategies when some variables of the SCC are removed from the perturbation valuations. However, there always exists a set of variables to perturb inside the SCC that results in an inevitable reprogramming strategy, but this set can be the whole SCC.

Finding this set is no easy task. An idea was to use the feedback vertex set, but example 8 shows that this does not always ensure inevitability of the strategy.

Example 8. *Let f be the Boolean network in fig. 3.3, pictured with its interaction graph. Let* initial $= 10000$ *be the initial state and* target $= 01100$ *be the target. Both algorithms would return that the first SCC, $\{1, 2, 3\}$ is the only one to perturb. We choose to make permanent perturbations so that this strategy is inevitable. Using the feedback vertex set, we choose to perturb only $\{1\}$, to the value 0. However, in the new Boolean network f' where $f'_1(x) = 0$ and $f'_i(x) = f_i(x)$ for all $i \neq 1$, two attractors are reachable,* target *and 01101, thus the strategy is only existential.*

Moreover, the previous algorithms 1 and 3 can return non-minimal results, as shows example 9.

Example 9. *Let f be the Boolean network described in fig. 3.4, with its interaction graph in the same figure. The SCCs are $\{1\}$ and $\{2, 3, 4\}$, squared in black in fig 3.4. We choose* initial $= 0011$ *and* target $= 1000$*, and the set of all attractors, once again all fixpoints, is:*

$$\{\{0000\}, \{0011\}, \{0100\}, \{1000\}, \{1100\}\}$$

Algorithms 3 and 1 would both return to perturb the two SCCs of the system. However, perturbing only $\{1\}$ is an inevitable reprogramming strategy from initial *to* target*, as 1100 is not reachable from 1011. This means that both algorithm do not return complete or minimal solutions.*

$$\boxed{\begin{aligned} f_1(x) &= x_1 \\ f_2(x) &= x_2 \land \neg x_3 \land \neg x_4 \\ f_3(x) &= x_4 \land \neg x_2 \land \neg x_1 \\ f_4(x) &= x_3 \land \neg x_2 \land \neg x_1 \end{aligned}}$$

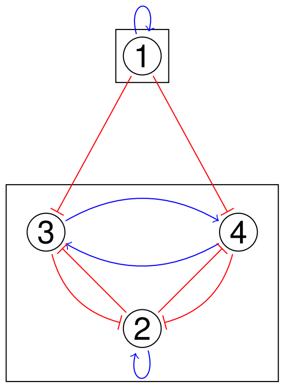

Figure 3.4: Boolean network that returns non minimal results.

These two examples imply that some analysis of the dynamics of the network is needed.

3.3.2 Necessary conditions?

However, even if analysis of the dynamics is needed, we want to find if there are some necessary conditions on which variables to perturb or which order to follow, to simplify our task.

Conditions on the variables

We did not find any necessary conditions on which variables to perturb when the perturbations are temporary. There is so much possibilities on how to time the perturbations, from nearly instantaneous to almost permanent, that the variables perturbed can be anywhere, inside the non-trivial SCCs or outside of them, variables that can have a different value in initial and target, or ones that keep the same value.

If the perturbations were not sequential, then for permanent perturbations, we would not perturb variables that do not have a different value in initial and target. However, due to the sequentiality of the perturbations, perturbing twice such a variable can be a very interesting option, as shows example 10. In the special case of one-step reprogramming strategies, permanent perturbations are always on variables v_i perturbed to have the value $a_i = \text{target}_{v_i}$. Inside or outside of SCCs does not matter.

Theorem 4. *Let $\mathcal{P} = (P_1)$ be a one-step perturbation sequence with permanent perturbations.* $\forall v_i \in P_1, a_i = \text{target}_{v_i}$.

$$\boxed{\begin{aligned} f_1(x) &= x_1 \\ f_2(x) &= x_1 \wedge x_2 \\ f_3(x) &= x_1 \wedge x_3 \end{aligned}}$$

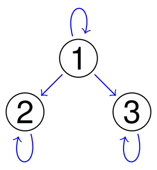

Figure 3.5: Boolean network where the same variable can be perturbed twice

Proof. If there exists a v_i in P_1 such that $a_i \neq \text{target}_{v_i}$, then for any state x, when perturbed permanently results in $\text{pert}_P(M, L, x) = (\phi_{L+M}(f), \chi_M(x)) = (x', f')$. x' always verify $x'_{v_i} = a_i$, and $f'(x') = a_i$. $\square$

Example 10. *Let f be the Boolean network in fig. 3.5, and its transition graph. From* initial $= 000$ *to* target $= 011$*, there are two perturbation sequences: either perturbing variables 2 and 3 to the value 1, or perturbing variable 1 to the value 1 and wait for 111 to be reached, then perturbing 1 again to the value 0.*

With the idea that a variable can be perturbed multiple times, all variables in the network can be permanently perturbed, with a small exception. If target is part of a complex attractor, and we want the whole attractor to be reachable, then a variable that have different values inside the attractor should not be permanently perturbed.

For instantaneous perturbations no necessary conditions on the variables were found.

Conditions on the perturbation order

We wondered if the perturbations order is linked to the topological order of the interaction graph, or if the system can be cut in subparts to reduce computation times.

However, no interesting way to cut the graph has been found, other than SCCs. Example 11 shows a Boolean network where the last SCC in topological order should be perturbed first.

Example 11. *Let f be the Boolean network described in fig. 3.6, with its interaction graph. To go from* initial $= 100$ *to* target $= 011$*, the shortest perturbation sequences are:*
first, either x_2 or x_3 to 1, then x_1 to 0.

Algorithm 3 finds the shortest perturbation sequences for this kind of example.

$$\boxed{\begin{array}{l} f_1(x) = x_1 \\ f_2(x) = x_2 \vee (x_1 \wedge x_3) \\ f_3(x) = x_3 \vee (x_1 \wedge x_2) \end{array}}$$

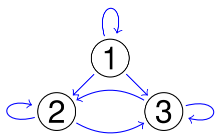

Figure 3.6: Sequentiality can be from lower SCCs first

$$\boxed{\begin{array}{l} f_1(x) = x_1 \wedge x_3 \\ f_2(x) = x_2 \vee (x_3 \wedge \neg x_1) \\ f_3(x) = x_3 \vee (x_2 \wedge \neg x_1) \end{array}}$$

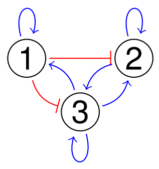

Figure 3.7: Better results in a single SCC with sequential perturbation sequence

Inside the SCC itself, we wondered if "one-step" perturbation sequences were always as good as sequential, and it is not the case, as shows example 12

Example 12. *Let f be the Boolean network in fig. 3.7, with its interaction graph. We want f to be inevitably reprogrammable from* $\mathsf{initial} = 000$ *to* $\mathsf{target} = 111$*, both of which are attractors, using instantaneous perturbations only (permanent perturbations would not change the example). If sequential, x_2 or x_3 can be modified first, then x_1. If simultaneous, the 3 of them have to be changed.*

Since we know that the order of the perturbations does not depend on the topology of the interaction graph, we wondered if it is still possible to cut the network in subparts, in order to lower computation times. However, perturbing the minimal number of variables in a SCC might result in a higher number of variables to perturb in an other one, as shows example 13.

Example 13. *Let f be the Boolean network presented in fig. 3.8, next to its interaction graph. Let* $\mathsf{initial} = 0100000$ *as the initial state and* $\mathsf{target} = 1010000$ *as the target attractor, we use permanent perturbations and want* $\mathsf{initial}$ *to be inevitably reprogrammable to* target*. Taking the minimum number of nodes to change in the first block means that x_2 has to be perturbed. Then both x_6 and x_7 have to be perturbed. On the other hand, if both x_1 and x_3 are perturbed in the first block, then there is no other change needed.*

As a conclusion, we need to have a global approach on the whole graph, using the complete dynamics of the system.

$$\begin{aligned}
f_1(x) &= \neg x_2 \\
f_2(x) &= \neg x_1 \wedge \neg x_3 \\
f_3(x) &= \neg x_2 \\
f_4(x) &= \neg x_2 \wedge (\neg x_1 \vee \neg x_3) \\
f_5(x) &= x_4 v \vee x_5 \vee x_6 \\
f_6(x) &= x_4 v \vee x_5 \vee x_6
\end{aligned}$$

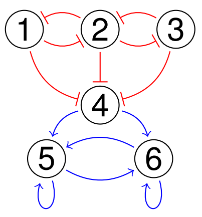

Figure 3.8: BN that results in higher number of changes in below block

Chapter 4

Sequential Reprogrammability

We saw in chapter 3 that even if static analysis of the Boolean network can return perturbation sequences, in most cases they are not minimal, and we concluded that to have such minimal results, an analysis of the dynamics is required. As explained in section 2.3, other works find some perturbation sequences, but none of them give the complete set of perturbation sequences.

This chapter uses published results from [MHP17, MSP$^+$19, MSH$^+$19], and explains a model to compute the complete set of perturbation sequences, as well as the algorithm to find this set. We also expand on the idea of algorithm 3 to use other attractors as intermediate steps, to reduce computation times.

4.1 Model

To find reprogrammability of the system and the perturbation sequences, we decided to create a model that includes the normal transitions and the perturbations that are possible at any given state of the network. To do so, we used Petri nets that copy the dynamics of the system, and add new places and transitions to model the perturbations. The Petri net model is explained in [MHP17], which is given in Appendix A (in an updated version in line with the definitions and terms used in this thesis). We are interested in the resulting "perturbed transition graph", which will include perturbation transitions and multiple layers.

From the Petri net model, we can create a *Perturbed Transition Graph* (Def. 24) which encompasses the transitions allowed by the Boolean network (often multiple times, to be able to track the perturbation sequence) and transitions due to perturbations. A new variable v_k tracks the number of applied perturbations. Therefore, the set of states of the perturbed transition graph is $V \times [0, k] \cup V'$, where V' represents the variables locking the functions when permanent perturbations are done, or other necessary variables. V' can be empty. The set of transitions contains normal transitions, which do not change the value of v_k, and perturbations, which increase its value by one.

Definition 24 (Perturbed Transition Graph). *Given a Boolean network f of dimension n, its transition graph* $\mathsf{STG}(f) = (V, E_f)$*, and a maximum number of allowed perturbations k, the* Perturbed Transition Graph *$\mathcal{G}$ is a pair (V_p, E_p) where*

- $V_p = (V \times [0, k]) \cup V'$ *is the set of states, with V' accounting for new variables required to model the perturbations.*
- $E_p \subseteq \{(s, i), v \to (s', i), v \mid i \in [0, k], (s \to s') \in E_f, v \in V'\} \cup \{(s, i), v \to (s', i + 1), v' \mid i \in [0, k - 1], s, s' \in V, v, v' \in V'\}$*, is the set of transitions, where variables inside V' change value only if perturbations are done.*

It should be noted that how the perturbations are made, which variables can be perturbed, in which states the variables can be perturbed, and other details are all part of the model. The model is an input from the user, and should conform to the following definition in order for the properties and algorithm to work on it. Other methods to compute the model can work as well, as long as the graph complies with def. 24.

Graph layers and layer subsets: The transition graph can be layered by regrouping all states having the same perturbation count in the same layer. Let $\mathcal{N}_j = V \times \{j\}$ be the set of all states in layer j. A subset of the original transition graph $\mathsf{STG}(f)$, $A \subseteq V$, times a layer j is $A_j := A \times \{j\}$. The subscript j will also be used when defining sets that are only on layer j.

Property 7. *Given a Perturbed Transition Graph $\mathcal{G}$ of a Boolean network f with* temporary *perturbations, for all $i \leq k$, the subgraph $\mathcal{N}_i$ of $\mathcal{G}$ is isomorphic to* $\mathsf{STG}(f)$*, the original network's transition graph.*

Property 8. *If $i > j$, then $(s, j) \in V_p$ cannot be reached from $(s', i) \in V_p$, regardless of s and s'. That is, the perturbation counter can only increase.*

From the definition of basin (Def. 8), the basin of a set of nodes X restricted to a layer considers only the transitions in the layer. Since we will always work with the same transition graph, the perturbed transition graph $\mathcal{G}$, we drop the $\mathcal{G}$ in the notation of basin restricted to a layer for the sake of readability.

Definition 25 (Basin restricted to a layer, $\mathsf{bas}_j(X)$). *The basin of X restricted to a layer j, denoted by $\mathsf{bas}_j(X)$, is the biggest set Y such that $X \subseteq Y \subseteq \mathcal{N}_j$ and $\forall y \in Y \setminus X, (\mathsf{post}(\mathcal{G})(y) \cap \mathcal{N}_j) \subseteq Y \wedge \exists x \in X, y \to^* x$.*

$$\begin{tikzcd}
 & 0100 & & 0101 & & 1100 & & 1101 \\
0110 \arrow[ur] \arrow[rr] & & 0111 \arrow[ur] & & 1110 \arrow[ur] \arrow[rr] & & 1111 \arrow[ur] &
\end{tikzcd}$$

$$\begin{tikzcd}
 & 0000 & & 0001 & & 1000 \arrow[dl] & & 1001 \arrow[dl] \\
0010 \arrow[ur] \arrow[rr] & & 0011 \arrow[ur] & & 1010 \arrow[rr] & & 1011 &
\end{tikzcd}$$

Figure 4.1: Transition graph of f

Examples

Examples 14 and 15 show how to build perturbed transition graphs when instantaneous or permanent perturbations are used.

Example 14. *From a Boolean network f, we will build the perturbed transition graph when the perturbations are instantaneous. f is the same Boolean network than in example 6:*
$f_1(x) = x_1$
$f_2(x) = x_2$
$f_3(x) = x_1 \wedge \neg x_2$
$f_4(x) = x_3 \vee x_4$

Fig. 4.1 is the transition graph of f. We want to add two perturbations to this graph, that is, $k = 2$ and $V_p = V \times \{0, 1, 2\}$, which means there will be three layers, layer 0, layer 1, and layer 2. All three layers are exact replicas of layer 0, because instantaneous perturbations are used. In order to obtain a graph easier to read, we only allow the first perturbation to happen in 0000, and only one variable can be perturbed ; also, we only allow the perturbation valuation $M = \{x_2 = 1\}$ as the valuation for the second perturbation. The perturbed transition graph with these constraints is shown in fig. 4.2, with perturbation transitions in red.

Example 15. *From a smaller Boolean network f', we build the perturbed transition graph with permanent perturbations. We use the Boolean network from ex. 1, defined as:*
$f'_1(x) = x_3 \vee (\neg x_1 \wedge x_2)$
$f'_2(x) = \neg x_1 \vee x_2$
$f'_3(x) = x_3 \vee (x_1 \wedge \neg x_2)$
Fig. 4.3 gives the transition graph of the Boolean network f'.

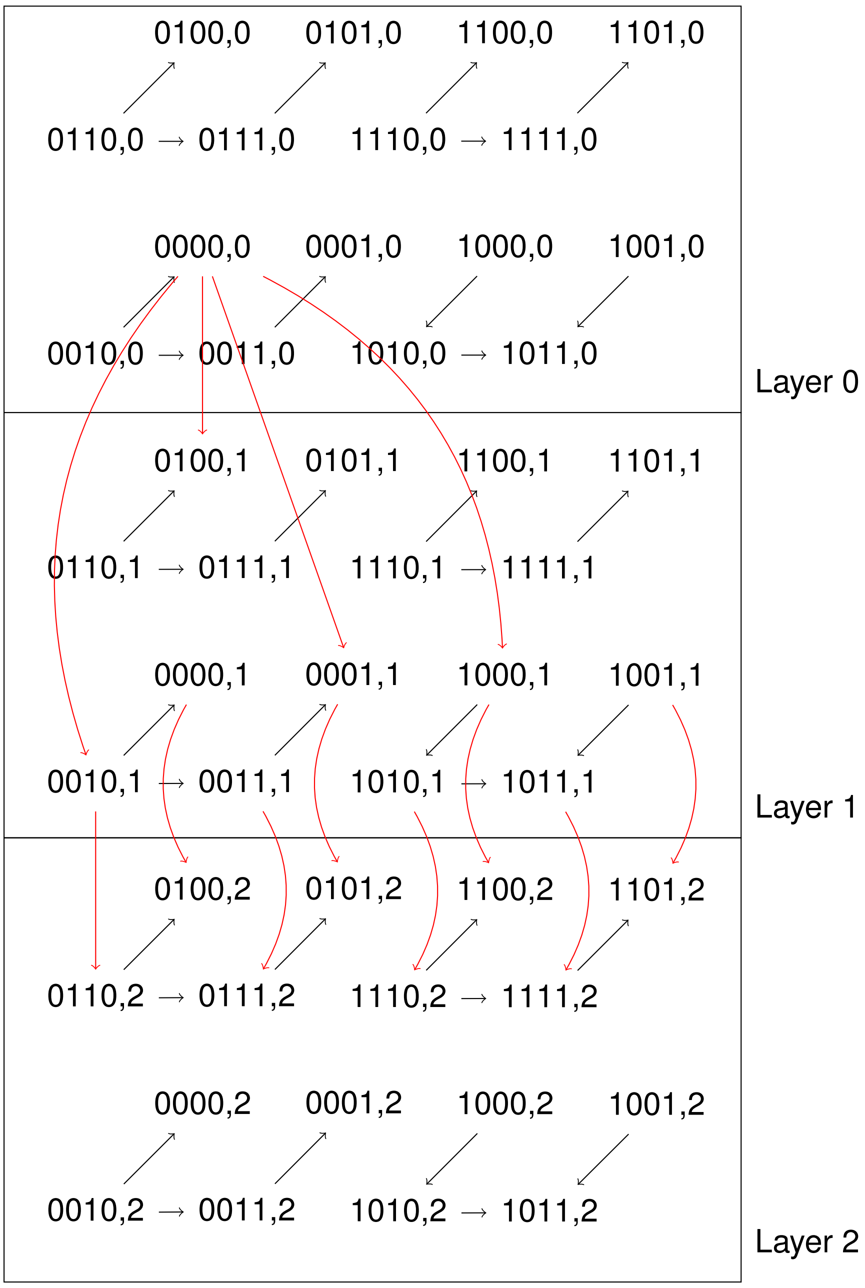

Figure 4.2: Perturbed transition graph with $k = 2$ and constraints on the perturbations.

$$\begin{tikzcd}
 & & 010 \arrow[r, bend left] & 110 \arrow[l, bend left] \\
 & & 000 \arrow[u] & 100 \arrow[l] \arrow[ddll] \\
011 \arrow[r] & 111 & & \\
001 \arrow[u] \arrow[r] & 101 & &
\end{tikzcd}$$

Figure 4.3: Transition graph of the Boolean network f'.

We construct a (partial) perturbed transition graph of this network in fig. 4.4, allowing all variables to be perturbed, but only allowing the perturbations to occur in the states 000 *and* 111. *Three new variables are created,* $lock(x_1)$, $lock(x_2)$, *and* $lock(x_3)$. *These variables start at value* 0, *and change value when a perturbation is made, locking the variable, preventing it to change value.*

Please note that this perturbed transition graph is only partial. For the sake of simplicity, we have omitted the parts where multiple locks are active at the same time, either two locks or all three locks (which would add 38 vertices to the graph). In this example, the layer is the number of active locks.

4.2 Set characterization of inevitable and existential reprogrammability

Given a Perturbed Transition Graph (V_p, E_p) of a Boolean network f with a maximum number of perturbations k (temporary or permanent), let $\mathsf{target} \subseteq V$ be a set of states of the original $\mathsf{STG}(f)$. We first focus on inevitable reprogrammability, and we explain how the method can be changed to return existential reprogramming strategies. States in target are the targets of the inevitable reprogrammability. $\mathsf{solution}$ is the solution set, the set of all states that have inevitable reprogrammability (often using different strategies) from them to one of the target states. The initial state of the system is $\mathsf{initial}$.

As a reminder, target_j is the set of states in target in $\mathsf{STG}(f)$ which are in layer j in the perturbed transition graph.

In order to compute $\mathsf{solution}$, two other sets ψ_j and $\mathsf{solution}_j$ for each layer are computed. ψ_j is the set of states in layer j that need we need to reprogram

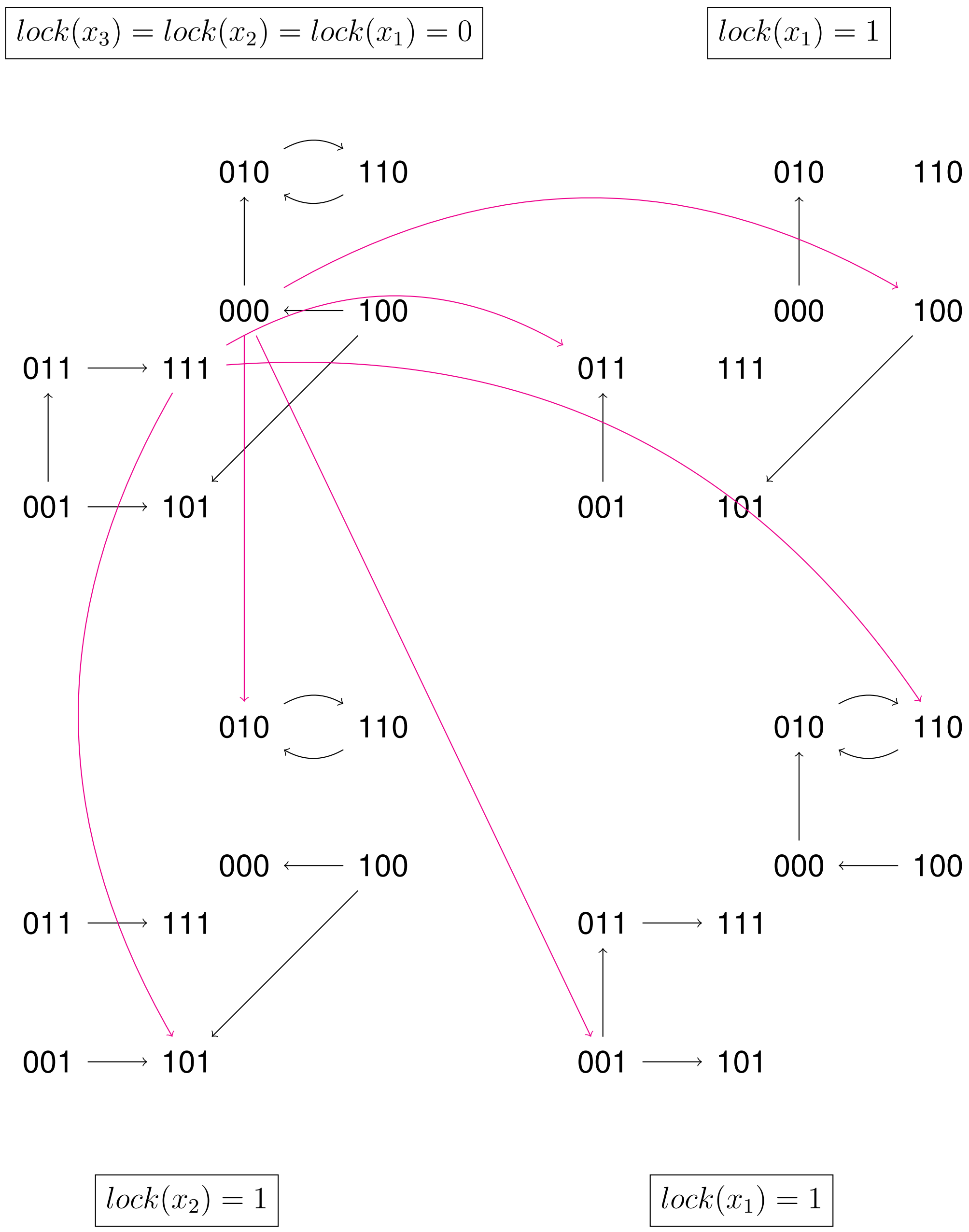

Figure 4.4: Perturbed Transition Graph of the Boolean network f'.

to in order to reach one of the target states in target. Thus, ψ_j contains states of target_j and states allowing one or multiple perturbations to a lower layer, where target can be reached. $\mathsf{solution}_j$ is the basin, restricted to the layer j, of ψ_j. Hence, $\mathsf{solution}_j$ is the set of states, in layer j, that can be reprogrammed to target, whether there are states in target in layer j or not.

The sets ψ_j and $\mathsf{solution}_j$ are defined as follows:
$\psi_k = \{x \mid x \in \mathcal{N}_k \wedge x \in \mathsf{target}_k\}$
$\mathsf{solution}_k = \mathsf{bas}_k(\psi_k)$

For j between $k-1$ and 0:
$\psi_j = \{x \mid x \in \mathcal{N}_j \wedge (x \in \mathsf{target}_j \ \vee x \in \mathsf{pre}(\mathcal{G})(\mathsf{solution}_{j+1})\}$
$\mathsf{solution}_j = \mathsf{bas}_j(\psi_j)$
Since ψ_k exists and is defined, all ψ_j and $\mathsf{solution}_j$ are defined.

The set describing the solution is $\mathsf{solution} = \cup_{j\in[1,k]}\mathsf{solution}_j$.
If the initial state $\mathsf{initial}$ is in $\mathsf{solution}$, then there exist inevitable reprogramming strategies, from which we can construct perturbation sequences (with a maximum size equal to k) and all these perturbation sequences are in $\mathsf{solution}$.

Property 9. *By construction, the set of all attractors reachable in the graph restricted to* $\mathsf{solution}$ *is a subset of* $\mathsf{target} \times [0, k]$.

4.2.1 Algorithm computing the set of states verifying the characterization

Algorithm 5, given in pseudo-code, returns all states verifying set characterization explained previously. The notations are the same. A loop invariant is that $\mathsf{solution}$ corresponds to the set of states from layer j to layer k that have inevitable reprogrammability to a state in target.

The inputs of the algorithm are a Perturbed Transition Graph (V_p, E_p), a set of states target, an initial state $\mathsf{initial}$, and the maximal number of perturbations, k.

The first step is to set $\mathsf{solution}_{k+1}$ to the empty set, as there are no states in layer $k+1$ (line 2), and $\mathsf{solution}$ is the set of all possible solutions, hence $\mathsf{solution} = \emptyset$ at the beginning.

Then, for each layer j, ψ_j has to be computed. ψ_j is the set of states either in target_j or being direct predecessors of the solution states of the lower layer, the layer $j+1$. The basin of ψ_j is then computed. These states are the solution states of layer j, and are added to the set of all solution states, $\mathsf{solution}$.

When the computation has reached the first layer, either the initial state is not among the solution states, in which case there is no perturbation sequence for this reprogramming (line 9), or, if initial $\in$ solution$_0$, then the set of states corresponding to the biggest reprogramming sequence is returned (line 11).

Algorithm 5 inevitable reprogrammability relying on the Set Characterization

```
1: function REPROGRAMMABILITY($\mathcal{G}$, target, initial, $k$)
2:     solution$_{k+1} = \emptyset$.
3:     solution $= \emptyset$
4:     for $j = k$ to $0$ do
5:         $\psi_j = \{x \in \mathcal{N}_j \mid x \in \text{target}_j \ \vee x \in \text{pre}(\mathcal{G})(\text{solution}_{j+1})\}$
6:         solution$_j = \text{bas}_j(\psi_j)$
7:         solution $=$ solution $\bigcup$ solution$_j$
8:     if initial $\notin$ solution$_0$ then
9:         Return $\emptyset$
10:    else
11:        Return solution
```

One could be surprised by the definition of ψ_j: in solution$_j$, we use the basin of ψ_j, where the system will always reach one of the state in ψ_j, and on the other hand, in ψ_j, only the preimage of solution$_{j+1}$ is used. However, in this case, the perturbations are chosen by the controller. As a consequence, the controller only needs to choose one perturbation among all possible ones in order to reach solution$_{j+1}$.

Complexity: Given a transition graph $\mathcal{G}$ from a network with n variables, and the possibility of doing up to k perturbations, $k \times m$ can be defined as the number of states of $\mathcal{G}$, with $m = O(e^n)$.

Since the algorithm relies on the exploration of subparts of the graph, and the distance between basins of attraction and other layers, the complexity, both in space and time, will vastly depend on the graph properties.

In the worst case, all $\ln(m)$ perturbations from a state in ψ_j have to be explored, and all $k \times m$ states are in different ψ_j. Hence, the worst time complexity is in $O(k \times m \times \ln(m))$, or, if we use the size of the network as the entry, $O(k \times n \times e^n)$.

As for space complexity, the algorithm returns solution, which is the set of explored states mentioned for time complexity, giving the same complexity.

However, these worst cases happen only if there are only perturbations in the transition graph, meaning there is no inherent dynamics to the model. Typical cases would have a lower time and space complexity.

Analysis of the solution returned by the algorithm: Since algorithm 5 returns a set of states, we want to extract from it a more precise information about which variables to perturb. We could chose to return all perturbation sequences, but in big networks this set of sequences is both very big and very redundant, with only a few different perturbation valuations, and the sequences changing only in the states in which the perturbation is applied. Therefore, we designed algorithm 6 to return simplified sequences: the ordered lists of the perturbation valuations.

Algorithm 6 takes as input $\mathcal{G}$, the perturbed transition graph, $\mathsf{initial}$ and target the initial set of states and the target set of states, and $\mathsf{solution}$, a set of states, which in our case is the $\mathsf{solution}$ from algorithm 5. First, it computes the set which are in all paths from $\mathsf{initial}$ to target, by taking the intersection of all successors of $\mathsf{initial}$ and $\mathsf{solution}$. Next, this set of states is ordered in reverse topological order, and thus, the first state will be in target, by construction of $\mathsf{solution}$ in the previous algorithm. For every state in target, in every layer, a new set of lists is created, containing the empty list. This set of lists is the set of ordered perturbation valuations. One will be created for each state of the network, in order to have all perturbation valuations when reaching $\mathsf{initial}$.

For each state y, in this set of states in reverse topological order, we look at each direct predecessor x. A new set of lists is created P_x, empty, if one does not exists. If $x \to y$ is not a perturbation, then the perturbations are the same as those of the successors of x, meaning y and possibly others. If it is a perturbation, then the perturbation valuation is extracted with the function $\mathsf{pert\_valuation}$. This perturbation is added to each perturbation lists in x, and each new list is added to P_x.

After going through all states, the algorithm returns P_{initial}, which contains all lists of perturbation valuations.

Visual explanation of the set algorithm, on example 6: Example 6 with two instantaneous perturbations ($k = 2$) results in the perturbed transition graph in fig. 4.5, from which most of the perturbation transitions have been removed, to make reading easier.

We want $\mathsf{initial} = 0000$, the initial state to be inevitably reprogrammable to $\mathsf{target} = 1101$, a target single state attractor. In fig. 6, $\mathsf{initial}$ in layer 0 and target (in all layers) are in red. Algorithm 5 can be seen as a highlighting of some states with different colors. In each layer i, the states in green are the states in ψ_i, and the states in blue are the states added to ψ_i in $\mathsf{solution}_i$, meaning the set of colored states in each layer i is the set $\mathsf{solution}_i$.

This graph should be looked at from bottom to top. In layer 2, only $1101, 2$ is in ψ_2, and $\mathsf{solution}_2 = \{1101, 2; 1111, 2\}$. In layer 1, the set ψ_1 is significantly

Algorithm 6 Characterization Set Analysis

```
 1: function SOLUTION_EXTRACTION($\mathcal{G}$, initial, target, solution)
 2:     States = solution $\cap$ post$^*(\mathcal{G})$(initial)
 3:     States = reverse_topological_order(States, $\mathcal{G}$)
 4:                         ▷ returns the reverse topological order of States in $\mathcal{G}$
 5:     for $x \in$ target do
 6:         for $i \in [0, k]$ do
 7:             $x_i = x \times \{i\}$
 8:             $P_{x_i} = \{[]\}$                        ▷ Set containing an empty list
 9:     for $y \in$ States do
10:         for $x \in$ pre$(\mathcal{G})(y)$ do
11:             if $P_x$ does not exists then
12:                 solution$_x = \{\}$
13:             if $\exists j$ such that $x \in \mathcal{N}_j$ and $y \in N_j$ then
14:                                                   ▷ $x$ and $y$ are in the same layer
15:                 $P_x = P_x \cup P_y$
16:             else
17:                                                   ▷ $x$ is one layer “before” $y$
18:                 $p$ = pert_valuation(x, y)
19:                 for $l \in P_y$ do
20:                     $l_2 = l$.add($p$)
21:                     $P_x$.add($l_2$)
22:     return $P_{\text{initial}}$
23: function pert_valuation($\mathcal{G} = (V_p, E_p), x, y$)
24:     solution = new map()
25:     for $v \in V$ do                              ▷ with $V$ from def. 24
26:         if $x_v \neq y_v$ then
27:             solution.add($v = y_v$)
28:     Return solution
```

larger than ψ_2:

$$\psi_1 = \{0101, 1; 0111, 1; 1001, 1; 1011, 1; 1100, 1; 1101, 1; 1110, 1; 1111, 1\}$$

In $\mathsf{solution}_1$, two new states are added to ψ_1, 1000 and 1010. Lastly ψ_0 is the whole layer 0.

As a result, the algorithm will return the set of all colored states, $\mathsf{solution}$. Applying algorithm 6 can be seen in fig. 4.6. The set “States” is the set of highlighted states, all other states have been greyed out. Inside this set, the perturbation transitions are shown. We can see that the only solution is first the perturbation $x_1 = 1$ and then $x_2 = 1$ when the state $1011, 1$ is reached. Thus, algorithm 6 returns $\{[x_1 = 1, x_2 = 1]\}$ on this example.

In this case, there is only one perturbation sequence, $[\mathsf{pert}_I((x_1 = 1), f, 0000); \mathsf{pert}_I((x_2 = 1), f, 1011)]$, but in larger networks, there can be many perturbation sequences with the same perturbation valuations.

Existential reprogrammability: The algorithm and experiments focus on inevitable reprogrammability as it is the most relevant for applications in cellular reprogramming. However, existential reprogramming strategies can be easily found with the same algorithm by changing $\mathsf{bas}_j(X)$ to $\mathsf{pre}^*_j(X)$ on line 6.
As a reminder, in chapter 2 the definition for $\mathsf{pre}^*(\mathsf{STG}(f))(X)$ is the set of states that can lead to X. This set can then be restricted to states only being in layer j, resulting in $\mathsf{pre}^*_j(X)$.
existential reprogrammability is the existence of a path to target. By using the set of predecessors instead of the basin, the algorithm no longer guarantees the inevitability, but still guarantees the target to be reachable with the perturbations returned. Therefore, replacing $\mathsf{bas}_j(\psi_j)$ for $\mathsf{pre}^*_j(\psi)_j$ in algorithm 5 results in an existential reprogrammability algorithm.

4.3 Attractor-based Sequential Reprogrammability

Sequential reprogrammability allows the network to be perturbed in any state (transient states or states in an attractor), but this requires complete observability of the system, which is very hard to obtain experimentally.

To make sequential reprogrammability and the perturbation sequences more practical, we designed an attractor-based sequential reprogrammability algorithm, using the idea of algorithm 3 from chapter 3. This algorithm only require partial observability, we only need to know the actual state of the attractor in which the system is. At each step, we apply a set of perturbations to stir the dynamics towards a state in the basin of an intermediate attractor (or a target

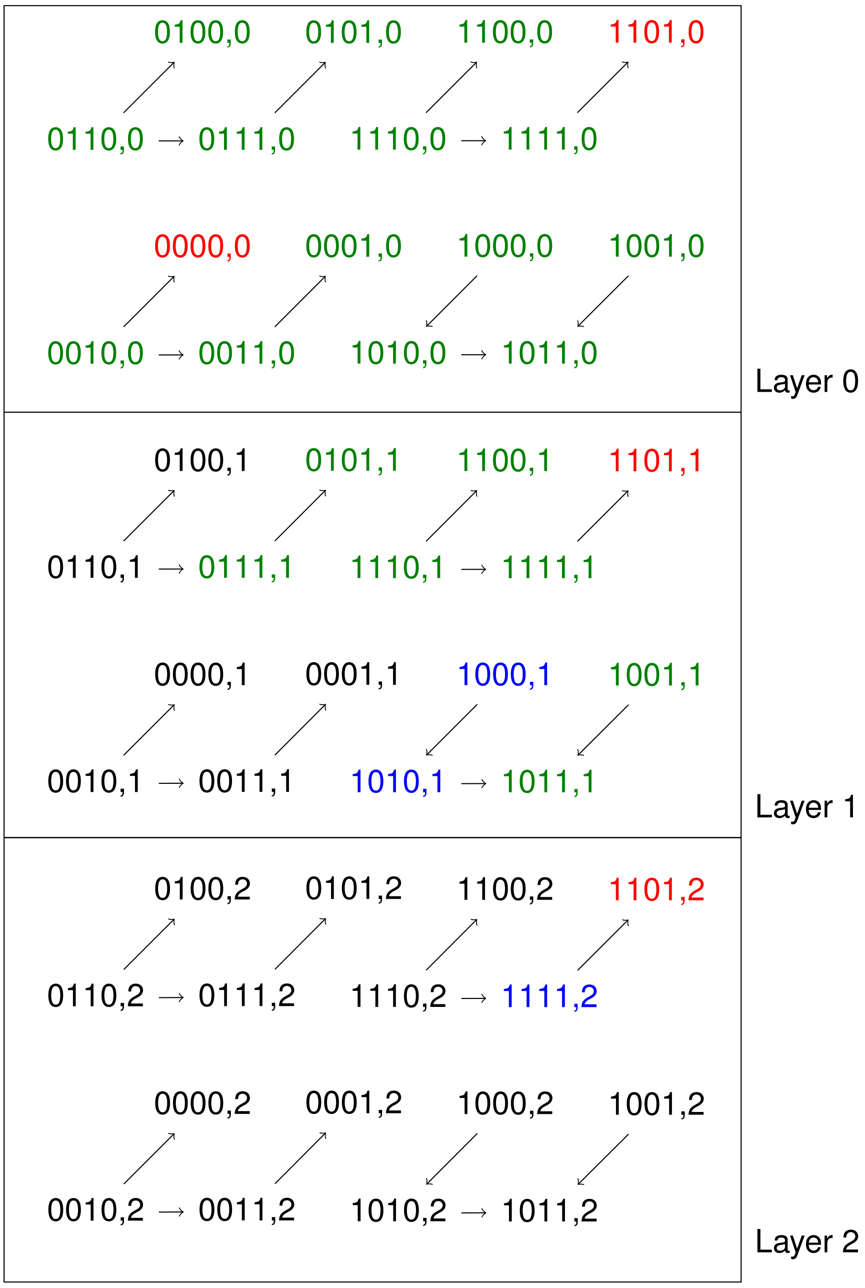

Figure 4.5: Computation of the set of nodes returned by the set characterization algorithm.

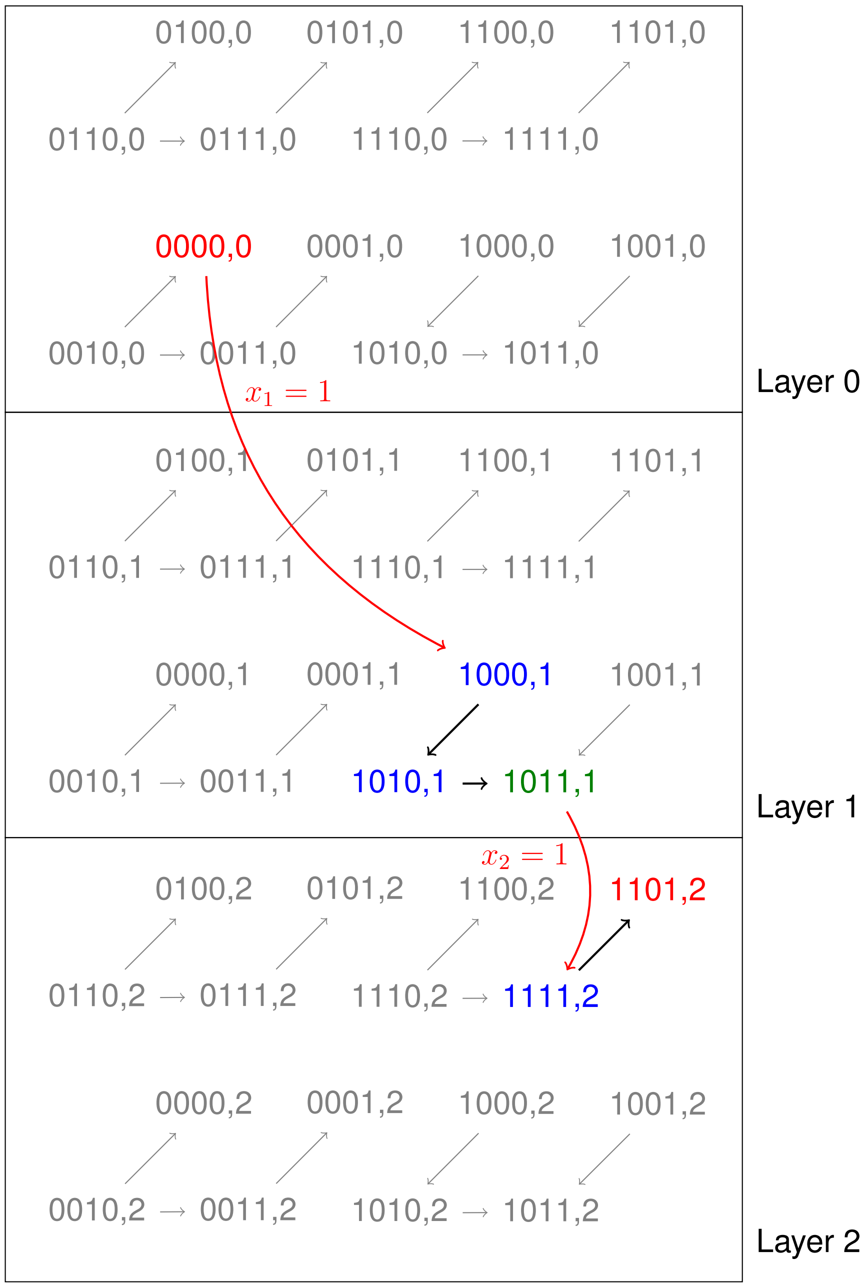

Figure 4.6: Analysis of the characterization set results.

attractor). We then let the network evolve spontaneously to the intermediate attractor (or the target attractor). We repeat the above procedure until the network reaches the target attractor.

In this section, we focus on instantaneous perturbations, while applying the perturbations longer will not affect the reachability of the target attractor. We also suppose that both initial and target are inside attractors, because cell reprogramming often consists of going from an initial phenotype (attractor in Boolean networks) to a different phenotype, our target. In practice, based on empirical experience, biologists may be able to determine how long it takes for the network to stabilize in an intermediate attractor, i.e., the timing to apply the next perturbations. In that case, if the intermediate attractors are single-state attractors, partial observability is not required. However, if the intermediate attractors are cyclic attractors, an observation of the state might still be required.

A feasible reprogrammability method has to encode practical considerations. In most cases, some variables cannot be perturbed, either because they represent an external cause the experimenter cannot change, or a set of multiple genes and proteins that would require a lot more work to perturb, or a transcription factor impacting only the gene or protein hasn't been found. Moreover, some attractors might not be suitable as intermediate states, because they lead to the death or disease of the cell.

Due to the diversity of biological networks, there does not exist one universal reprogramming strategy that suits all different networks. Hence, we develop an algorithm to compute all attractor-based perturbation sequences satisfying the following constraints:

1. the total number of perturbations is less than a threshold;
2. certain attractors can be avoided as intermediates;
3. certain variables of the network can be avoided to be perturbed.

These constraints encode the previous practical considerations and thus lead to biologically feasible perturbation sequences. We describe such an algorithm in the next subsection.

The general principle of this method can be applied to other means to compute the required perturbations for the system to reach a target attractor, given an initial state in an attractor.

4.3.1 Algorithm

Let f be a Boolean Network of size n. Let $\bar{\mathsf{V}}$ be the set of variables that cannot be perturbed, initial be an attractor of the network, which is the initial state of the system, and target be another attractor of the network, which is the target to reprogram to.

Algorithm 7 describes the algorithm to compute perturbation sequences from initial to target, using other attractors as intermediate steps. The inputs are: the Boolean Network f, the initial attractor initial, the target attractor target, the set of attractors $\mathcal{A}(f)$ of f that can act as intermediate states, and the set of variables $\bar{\mathsf{V}}$ that cannot be perturbed. We define the set $\mathcal{A}$, which is $\mathcal{A}(f)$ excluding the attractors that cannot act as intermediates, such as the attractor target and the undesired attractors[1]. In this case, since only instantaneous perturbations are considered, the Boolean network f will not change, thus allowing to use the same list of attractors.

The algorithm uses a modified Hamming distance hd_m between the states of the transition system. This modified Hamming distance is equal to the Hamming distance hd except if there is a variable v from $\bar{\mathsf{V}}$ which have a different value in the two states, in which case the distance is infinite. Between a state x and a state y, this modified Hamming distance $\mathsf{hd}_m(x, y)$ is defined as:

$$\mathsf{hd}_m(x,y) = \begin{cases} \infty & \text{if } \exists v \in \bar{\mathsf{V}}, x_v \neq y_v \\ \mathsf{hd}(x,y) & \text{otherwise} \end{cases}$$

The modified Hamming distance between two sets of states X and Y is defined as: $\mathsf{hd}_m(X, Y) = min_{x\in X, y\in Y}(\mathsf{hd}_m(x, y))$.

To compute the sequential paths from initial to target using other attractors as intermediate states, we have to compute the basin of target, which is $\mathsf{bas}(\mathsf{STG}(f), \mathsf{target})$. Since we only use the distance between a set of states and a basin, let HB_m, the distance between a set of states X and the basin of a set of states Y, be defined as $\mathsf{HB}_m(X, Y) = \mathsf{hd}_m(X, \mathsf{bas}(\mathsf{STG}(f), Y))$. Algorithm 8 describes how to compute both of these distances, as well as how to compute the argument of HB_m: the set(s) of variables that realize the minimum Hamming distance and the desired value of these variables. The distance between initial and the basin of target, $\mathsf{max\_dist} = \mathsf{HB}_m(\mathsf{initial}, \mathsf{target})$, will be used as a benchmark for the next computations: this is the maximum perturbation sequence size allowed to reach target.

An empty dictionary $\mathcal{L}_{\mathsf{initial}}$ is created, to store the possible perturbation sequences. If $\mathsf{max\_dist} < \infty$, the perturbation valuation, $a = \mathsf{arg\_HB}(f, \bar{\mathsf{V}}, \mathsf{initial}, \mathsf{target})$ is computed. The perturbation sequence, represented by a list of targets to reach in order to reach the next one, [target] is added as an entry of the dictionary $\mathcal{L}_{\mathsf{initial}}$, with the value $(\mathsf{max\_dist}, [a])$, its size and the list of perturbation valuations. This dictionary regroups $\mathcal{L}_{\mathsf{initial}}$ all perturbation sequences from initial to target, the first value is its size, and the second is the list of the perturbation

[1]We refer details on attractor detection to [MPQY18].

valuations. From an entry $[A, B, C] = (size, [M_A, M_B, M_C])$ the perturbation sequence S can be easily computed: $S = [\mathsf{pert}_I(M_A, f, A), \mathsf{pert}_I(M_B, f, B), \mathsf{pert}_I(M_C, f, C)]$ A special value is added to the dictionary, "min", which is the minimal size of all the perturbation sequences from initial to target, and it is given the value max_dist.

Then (**for** loop), for each attractor A in $\mathcal{A}$ except initial (since we already did the computations on it), the distance $d = \mathsf{HB}_m(\mathsf{A}, \mathsf{target})$ is computed. If this distance d is strictly lower than max_dist[2], then A is added to a list of attractors list and a dictionary $\mathcal{L}_\mathsf{A}$ is created. We add to $\mathcal{L}_\mathsf{A}$ the entry $[\mathsf{target}]$ to which we associate the size of the perturbation sequence, d, and the perturbation valuation, $[\mathsf{arg\_HB}(f, \bar{\mathsf{V}}, \mathsf{A}, \mathsf{target})]$. The perturbation sequence is a list of the attractors to reach in the right order. The perturbation valuation is a set, a dictionary in our case, containing the variables to perturb and the desired values. This set is put in a list: each set of the list is a perturbation valuation to apply in the current attractor to reach the next one in the sequence. A special value "min" is added to the dictionary, in the same way as for $\mathcal{L}_\mathsf{initial}$, to store the minimal length of paths from A to target.

The list list is used to recursively compute the shortest perturbation sequences. As long as list is not empty, the following steps are done (**while** loop):

1. First, create an empty list l.
2. Then, from all attractor A_1 in $\mathcal{A}$, for all attractor A_2 in list, the distance $d = \mathsf{HB}_m(\mathsf{A}_1, \mathsf{A}_2)$ is computed. If this distance plus $\mathcal{L}_{\mathsf{A}_2}[\text{"min"}]$[3] is lower than max_dist, then for every perturbation sequence seq in $\mathcal{L}_{\mathsf{A}_2}$, the total size $d + \mathcal{L}_{\mathsf{A}_2}[\mathsf{seq}][0]$[4] is computed. If this distance is lower or equal to max_dist and if $\mathsf{A}_1 \notin \mathsf{seq}$, a new entry $[\mathsf{A}_2] + \mathsf{seq}$[5] is added to $\mathcal{L}_{\mathsf{A}_1}$, with the value $(d + \mathcal{L}_{\mathsf{A}_2}[\mathsf{seq}][0], [\mathsf{arg\_HB}(\mathsf{A}_1, \mathsf{A}_2)] + \mathcal{L}_{\mathsf{A}_2}[\mathsf{seq}][1])$. The first value, the size, is the distance to go from A_1 to target using A_2 as an intermediate step, and the paths from A_2 to target already computed. The second value is the perturbation valuation, using the same principle. If the dictionary does not exist, it is created, and "min" is updated or created. Moreover, A_1 is added to l if there exists at least one new sequence, and if $\mathsf{A}_1 \neq \mathsf{initial}$ because we do not use initial as an intermediate step.
3. Lastly, the value of list is changed to match l, $\mathsf{list} = l$.

When this loop is over, all perturbation sequences are in $\mathcal{L}_\mathsf{initial}$, with the associated size and perturbation valuations, and $\mathcal{L}_\mathsf{initial}$ is returned.

[2] In this case, if $\mathsf{max\_dist} = \infty$, any non infinite distance is considered strictly lower.
[3] This value is the minimal size of any perturbation sequence from A_1 to target.
[4] As A_2 is in list, $\mathcal{L}_{\mathsf{A}_2}$ exists.
[5] Here, the $+$ is the usual concatenation for lists.

Algorithm 7 inevitable reprogrammability of BNs from initial to target

```
 1: procedure COMPUTATION_OF_INEVITABLE_PATHS($f$, initial, target, $\mathcal{A}$, $\bar{V}$)
 2:     max_dist = $HB_m(f, \bar{V}, initial, target)$
 3:     $\mathcal{L}_{initial}$ = new empty dictionary
 4:     if max_dist $< \infty$ then
 5:         $a$ = arg_HB($f, \bar{V}$, initial, target)
 6:         $\mathcal{L}_{initial}$.add([target] : (max_dist, $[a]$)) ; $\mathcal{L}_{initial}$.add("min" : max_dist)
 7:     list := $\emptyset$
 8:     for A $\in (\mathcal{A} \setminus$ initial) do
 9:         $d = HB_m(f, \bar{V}, A, target)$
10:         if $d <$ max_dist then
11:             list.add(A)
12:             $\mathcal{L}_A = \mathrm{map}()$
13:             $a$ = arg_HB($f, \bar{V}$, A, target)
14:             $\mathcal{L}_A$.add([target] : $(d, [a])$) ; $\mathcal{L}_A$.add("min" : $d$)
15:     while list $\neq \emptyset$ do
16:         $l := \emptyset$
17:         for $A_1 \in \mathcal{A}$ do
18:             add_to_l = 0
19:             for $A_2 \in$ list $\setminus \{A_1\}$ do
20:                 $d = HB_m(f, \bar{V}, A_1, A_2)$
21:                 if $d \neq \infty$ and $d + \mathcal{L}_{A_2}$["min"] $\leq$ max_dist then
22:                     for seq $\in \mathcal{L}_{A_2} \setminus$ {"min"} do
23:                         $td = d + \mathcal{L}_{A_2}$[seq][0]
24:                         if $td \leq$ max_dist and $A_1 \notin$ seq then
25:                             add_to_l = 1
26:                             if $\mathcal{L}_{A_1}$ does not exists then
27:                                 $\mathcal{L}_{A_1} = \mathrm{map}()$
28:                                 $\mathcal{L}_{A_1}$.add("min" : $td$)
29:                             $a$ = arg_HB($f, \bar{V}, A_1, A_2$)
30:                             $\mathcal{L}_{A_1}$.add([$A_2$] + seq : ($td$, $[a] + \mathcal{L}_{A_2}$[seq][1]))
31:                             if $td < \mathcal{L}_{A_1}$["min"] then
32:                                 $\mathcal{L}_{A_1}$["min"] = $td$
33:                 if add_to_l = 1 and $A_1 \neq$ initial then
34:                     $l$.add($A_1$)
35:         list = $l$
36:     return $\mathcal{L}_{initial}$
```

Algorithm 8 Distance functions

```
1: function hd$_m(f, \bar{V}, x, y)$
2:     $sum = 0$
3:     for $i = 1, i \leq n, i++$ do
4:         if $x_i \neq y_i$ then
5:             if $i \in \bar{V}$ then
6:                 return $\infty$
7:             $sum = sum + |x_i - y_i|$
8:     return $sum$
9: function HB$_m(f, \bar{V}, X, Y)$
10:     $B =$ bas(STG$(f), Y)$
11: ▷ Details on the computation of basins can be found in [PSPM18, PSPM19]
12:     return $min_{x \in X, y \in Y}($hd$_m(\bar{V}, x, y))$
13: function arg_HB$(f, \bar{V}, X, Y)$
14:     $min =$ HB$_m(f, \bar{V}, X, Y)$
15:     if $min = \infty$ then
16:         Fail("infinite distance")
17:     $D = \text{valuation}()$
18:     for $x \in X$ do
19:         for $y \in Y$ do
20:             if hd$_m(f, \bar{V}, x, y) = min$ then
21:                 for $i = 1, i \leq n, i++$ do
22:                     if $x_i \neq y_i$ then
23:                                 ▷ Associate with variable $i$ its desired value $y_i$
24:                         $D$.add($i : y_i$)
25:     return $D$
```

$$\begin{tikzcd}
 & 0100 & 0101 & 1100 & \boxed{1101} \\
0110 \arrow[ur] \arrow[r] & 0111 \arrow[ur] & 1110 \arrow[ur] \arrow[r] & 1111 \arrow[ur] &
\end{tikzcd}$$

$$\begin{tikzcd}
 & \boxed{0000} & 0001 & 1000 \arrow[dl] & 1001 \arrow[dl] \\
0010 \arrow[ur] \arrow[r] & 0011 \arrow[ur] & 1010 \arrow[r] & 1011 &
\end{tikzcd}$$

Figure 4.7: Transition graph of f

4.3.2 Example

We apply algorithm 7 to example 6, with Boolean network f:
$f_1(x) = x_1$
$f_2(x) = x_2$
$f_3(x) = x_1 \wedge \neg x_2$
$f_4(x) = x_3 \vee x_4$

As a reminder, fig. 4.7 is the transition graph (which is not computed in algorithm 7). We want to find an inevitable reprogramming strategies and the resulting perturbation sequences from $\text{initial} = 0000$ (boxed in red) to $\text{target} = 1101$ (boxed in blue).

To make it more challenging, we set $\bar{\text{V}} = \{x_4\}$. Before computing the sequences, all attractors are computed. Below is the list of all attractors of f:

$$\mathcal{A}(f) = \{0000; 0001; 0100; 0101; 1011; 1100; 1101\}$$

.

For this example, $\mathcal{A} = \mathcal{A}(f) \setminus \{0100; 1101\}$ (target is always removed from $\mathcal{A}(f)$, as it will not be used as an intermediate step). Algorithm 7 first computes max_dist, here $\text{max\_dist} = \infty$ because $\text{initial}_4 \neq \text{target}_4$ and $x_4 \in \bar{\text{V}}$. Then, for each attractor A, $\text{HB}_m(f, \bar{\text{V}}, A, 1101)$ is computed, and $\mathcal{L}_A$ is created when needed.

	0001	0101	1011	1100
$\mathcal{L}_A$	$[1101] : (2, [(x_1 = 1, x_2 = 1)])$	$[1101] : (1, [(x_1 = 1)])$	$[1101] : (1, [(x_2 = 1)])$	$\emptyset$
min:	2	1	1	∞

Most of these results are obvious, except 1011. $\text{bas}(\text{STG}(f), 1101) = \{1111, 1101\}$, hence why $\text{HB}_m(f, \bar{\text{V}}, 1011, 1101) = 1$, thanks to 1111. As a result, $\text{list} = [0001, 0101, 1011]$.

While loop: For each attractor A in $\mathcal{A}$, the distance to each attractor in list is computed.

$\mathsf{HB}_m(f,\bar{\mathsf{V}},\mathsf{A}_1,\mathsf{A}_2)$	$\mathsf{A}_2 = 0001$	$\mathsf{A}_2 = 0101$	$\mathsf{A}_2 = 1011$
$\mathsf{A}_1 = 0000$	∞	∞	1
$\mathsf{A}_1 = 0001$	/	1	1
$\mathsf{A}_1 = 0101$	1	/	2
$\mathsf{A}_1 = 1011$	1	2	/
$\mathsf{A}_1 = 1100$	∞	∞	1

Since there is at least one finite distance for each attractor, and no sequence of more than one attractor yet, they are all added to the new list, meaning list $= \mathcal{A}$.

All dictionaries are either updated, or created and updated:

Dictionary	**Perturbation sequences**	**Min**
$\mathcal{L}_{0000} =$	$[1011, 1101] : (2, [(x_1 = 1), (x_2 = 1)])$	$(\text{"}min\text{"}) : 2$
$\mathcal{L}_{0001} =$	$[1101] : (2, [(x_1 = 1, x_2 = 1)]);$	
	$[0101, 1101] : (2, [(x_2 = 1), (x_1 = 1)]);$	
	$[1011, 1101] : (2, [(x_1 = 1), (x_2 = 1)])$	$(\text{"}min\text{"}) : 2$
$\mathcal{L}_{0101} =$	$[1101] : (1, [(x_1 = 1)]);$	
	$[0001, 1101] : (3, [(x_2 = 0), (x_1 = 1, x_2 = 1)]);$	
	$[1011, 1101] : (3, [(x_1 = 1, x_2 = 0), (x_2 = 1)])$	$(\text{"}min\text{"}) : 1$
$\mathcal{L}_{1011} =$	$[1101] : (1, [(x_1 = 1)]);$	
	$[0001, 1101] : (3, [(x_1 = 0), (x_1 = 1, x_2 = 1)]);$	
	$[0101, 1101] : (3, [(x_1 = 0, x_2 = 1), (x_1 = 1)]);$	$(\text{"}min\text{"}) : 1$
$\mathcal{L}_{1100} =$	$[1011, 1101] : (2, [(x_2 = 0), (x_2 = 1)];$	$(\text{"}min\text{"}) : 2$

We can note that there is already a solution in $\mathcal{L}_{0000}$, where a "one-step" reprogramming strategy cannot find one.

Once again, the while loop is executed, for each attractor in $\mathcal{A}$ and each attractor in list, the distances are computed:

$\mathsf{HB}_m(f,\bar{\mathsf{V}},\mathsf{A}_1,\mathsf{A}_2)$	$\mathsf{A}_2 = 0000$	$\mathsf{A}_2 = 0001$	$\mathsf{A}_2 = 0101$	$\mathsf{A}_2 = 1011$	$\mathsf{A}_2 = 1100$
$\mathsf{A}_1 = 0000$	/	∞	∞	1	2
$\mathsf{A}_1 = 0001$	∞	/	1	1	∞
$\mathsf{A}_1 = 0101$	∞	1	/	2	∞
$\mathsf{A}_1 = 1011$	∞	1	2	/	∞
$\mathsf{A}_1 = 1100$	2	∞	∞	1	/

To simplify further reading, we can already see a pattern here, where all "one-step" perturbation sequences from any attractor to any other have been computed. Below is the table of perturbation valuations for "one-step" perturbation sequences from A_1 to A_2, or the ∞ symbol when impossible.

$A_1 \setminus A_2$	0000	0001	0101	1011	1100	1101
0000	/	∞	∞	$(x_1 = 1)$	$(x_1 = 1, x_2 = 1)$	∞
0001	∞	/	$(x_2 = 1)$	$(x_1 = 1)$	∞	$(x_1 = 1, x_2 = 1)$
0101	∞	$(x_2 = 0)$	/	$(x_1 = 1, x_2 = 0)$	∞	$(x_1 = 1)$
1011	∞	$(x_1 = 0)$	$(x_1 = 0, x_2 = 1)$	/	∞	$(x_2 = 1)$
1100	$(x_1 = 0, x_2 = 0)$	∞	∞	$(x_2 = 0)$	/	∞

In practice, the already computed modified hamming distances and perturbation valuations should be stored, to avoid computing them each time they are needed. Now that we have this table, we will only show the sequences in the dictionaries, without the associated perturbation valuations, which can be easily found in the previous table. The "min" is also excluded, since the value will not change anymore.

Dictionary	**Perturbation sequences**
$\mathcal{L}_{0000} =$	$[1011, 1101]; [1011, 0001, 1101]; [1011, 0101, 1101];$
	$[1100, 1011, 1101]$
$\mathcal{L}_{0001} =$	$[1101];$
	$[0101, 1101]; [0101, 1011, 1101];$
	$[1011, 1101]$
$\mathcal{L}_{0101} =$	$[1101]$
	$[0001, 1101]; [0001, 1011, 1101]$
	$[1011, 1101]; 1011, 0001, 1101]$
$\mathcal{L}_{1011} =$	$[1101]$
	$[0001, 1101]; [0001, 0101, 1101];$
	$[0101, 1101]; [0101, 0001, 1101];$
$\mathcal{L}_{1100} =$	$[1011, 1101]; [1011, 0001, 1101]; [1011, 0101, 1101]$

After two more iterations, we obtain the following $\mathcal{L}_{0000}$:

$$
\begin{aligned}
\mathcal{L}_{0000} = \{ \quad & [1011, 1101] \\
& [1011, 0101, 1101]; [1011, 0101, 0001, 1101] \\
& [1011, 0001, 1101]; [1011, 0001, 0101, 1101] \\
& [1100, 1011, 1101] \\
& [1100, 1011, 0101, 1101]; [1100, 1011, 0101, 0001, 1101] \\
& [1100, 1011, 0001, 1101]; [1100, 1011, 0001, 0101, 1101] \qquad \}
\end{aligned}
$$

Which means there are ten perturbation sequences, from size 2 to size 8. If perturbations were allowed on x_4, the computation would have been much quicker, since in this case, $\text{max\_dist} = 3$.

Chapter 5

Case Studies

To show the differences between the two algorithms from chapter 4 and compare with other methods from the literature, we tested them on different networks from literature. We tested the specific case of instantaneous perturbations for inevitable reprogrammability. The first network we studied is the cardiac gene regulatory network. This rather small network contains small SCCs. As a result, the static analysis from chapter 3 returns very small perturbation sequences. The completely sequential algorithm from chapter 4 returns even smaller perturbation sequences. The special case of reprogramming from SHF to FHF is studied, in which we show that using sequential reprogramming strategies allow for smaller perturbation sequences than one-step reprogramming strategies, and that a wide range of genes can be perturbed after perturbing the first exogen of $canWnt$.

We then studied the case of the model of molecular pathways enabling tumour cell invasion and migration, which we refer as "tumour" model. In this model, attractor-based strategies return very short perturbation sequences, of size equal to the shortest perturbation sequences returned by completely sequential strategies, except in one case. We studied how constraints on the perturbations changed the results, and how permanent perturbations allowed to reach any kind of attractor with only one variable perturbed.

Lastly, we studied the PC12 cell differentiation network, and how to go from cell cycle arrest to differentiation, and how static analysis helps understanding better the results obtained for this reprogramming.

Tools used: Algorithms 5 and 6 were first implemented in python, which showed that the results were interesting. These algorithms are the ones computing the completely sequential perturbation sequences, and extracting the results.

Then, algorithms 5 and 7 were implemented and integrated in Cabean by

Cui Su under the supervision of Jun Pang from SnT Team from University of Luxembourg, using binary decision diagrams to reduce computation times. Algorithm 7 which computes attractor-based perturbation sequences was created with them, to combine the quickness of their computations for one-step perturbation sequences with the shorter size of sequential perturbation sequences. We worked with them to clarify the algorithms. The details of the implementation are given in [MSH+19, MSP+19].

The material to reproduce the case studies is available at `https://github.com/HuguesMandon/CaseStudiesCellReprogrammingBN`.

Remark: We defined the perturbations as functions with three parameters, a perturbation valuation to apply, a perturbation valuation corresponding to past modifications on the function, and a state in which to apply the perturbation. In this chapter, given the size of perturbation sequences, and the fact that multiple perturbations with the same valuations but in a different state can result in the same results, we will mostly focus on the variables perturbed. When the state is important, it will be specified.

5.1 Cardiac gene regulatory network

The cardiac gene regulatory network is a Boolean network constructed by Hermann et al. in 2012 to study the early cardiac development of the mouse, using the gene regulations to understand the FHF / SHF determination[HGZ+12]. This is a rather small network, with only 15 variables and 6 attractors, which we studied quite early on, and on which it is easier to give more complete results, such as detailed perturbation sequences. Its interaction graph is shown in fig. 5.1, and is in a topological order, increasing from top to bottom. The non-trivial SCCs are in magenta.

Attractors: The network has been designed with two specific attractors in mind, FHF and SHF. In [HGZ+12], the variable $exogen\_BMP2\_I$ is supposed to be always active, but we allowed it to change value and behave as a self-loop. In this case, the network contains 6 attractors: two which are probably artifacts of the model, one were all variables are inactive (Inactive) and one where only BMP2 and its exogens are active (ActiveBMP, which was found in the original paper but not much discussed). Two attractors are the ones the network was built for, FHF and SHF, and two are close to either FHF or SHF, with only few variables having different values. We named these last two “FHF muted” and

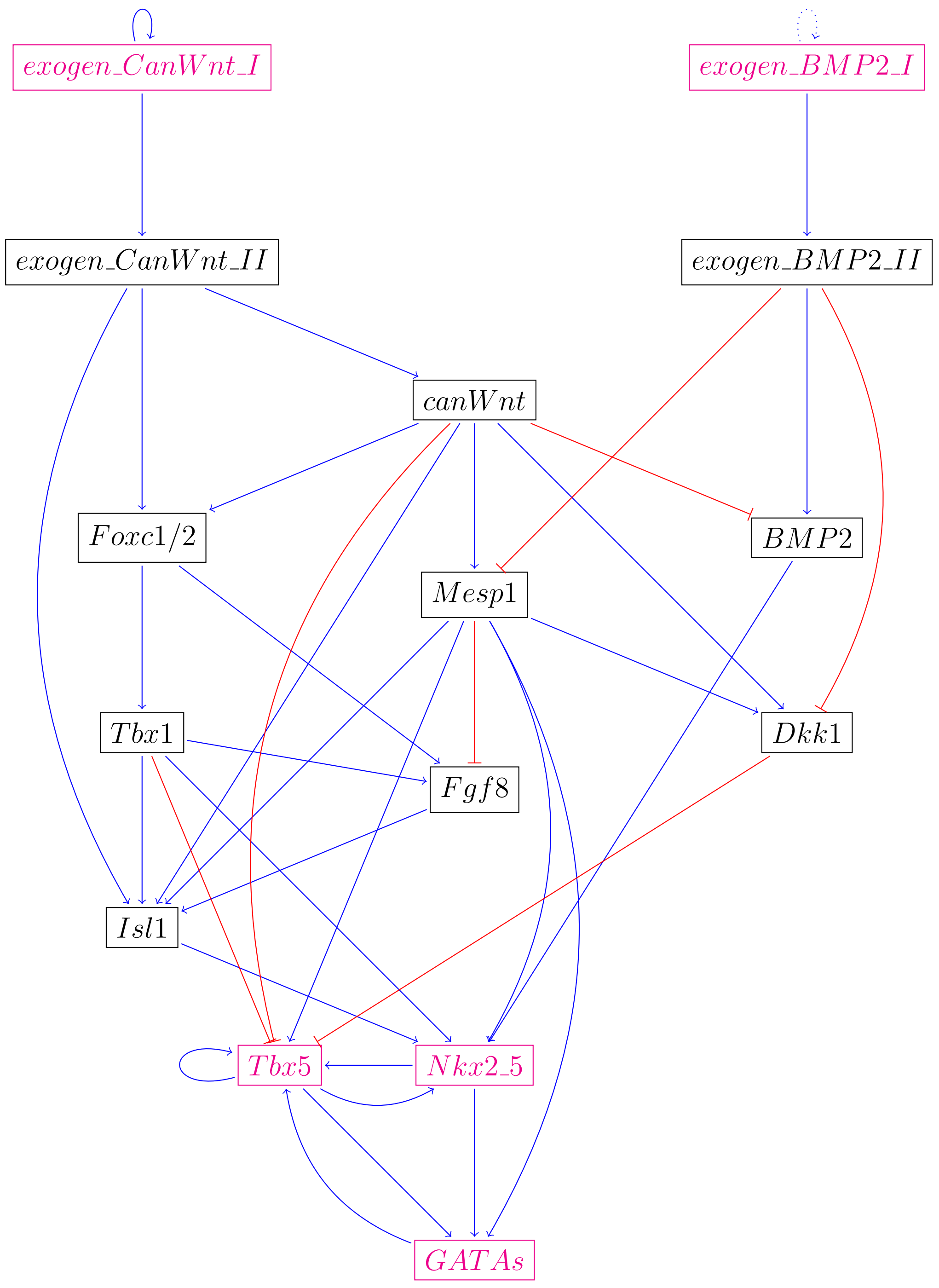

Figure 5.1: Interaction graph of the “cardiac” Boolean network

"SHF muted" which both have $exogen\_BMP2\_I$ inactive. The list of active and inactive variables in each attractor is shown in table 5.1.

Static analysis: From the attractor list and the interaction graph, we can perform static analysis of the network. There are three non-trivial SCCs, which are $A = \{exogen\_BMP2\_I\}$, $B = \{exogen\_CanWnt\_I\}$, and a larger one, $C = \{Tbx5, GATAs, Nkx2\_5\}$.

Algorithm 3 from chapter 3 computes the best sequences of non-trivial SCCs to perturb to reach a target attractor. To simplify, we will denote the sequences as lists of non-trivial SCCs, A, B or C, where the valuation is the list of variables of the SCC associated with the value they have in the target. Table 5.2 are the lists returned by the algorithm for each initial state and target attractor.

We can already see that if we use sequences with no restrictions, the maximum size of the sequences is five, because both A and B have 1 element, and C have three.

Dynamical analysis: To compare with one-step, completely sequential, and attractor-based reprogramming strategies, we computed table 5.3, which contains all three results. If there is only one number in a case, then it means all methods return the same sequences, this is often the case when only a few perturbations are needed. It also means that the result is the same as if one-step reprogramming strategies methods were used. If there are multiple results, the first one is always the size of the smallest perturbation sequence for the completely sequential reprogramming strategies, the list represent all perturbation sequences returned by the attractor-based sequential reprogramming strategies, and the last result of the list is always the size of the one-step sequence, which is returned by the attractor-based reprogramming strategies. When the first result is inside parenthesis, it means attractor-based sequential reprogramming strategies return longer perturbation sequences. The computation times for attractor-based reprogramming strategies are given in table 5.4. Computations for completely sequential strategies were made on a different computer, with times ranging from 0.006 to 0.228. All times are given in seconds.

	$exogen1$ $BMP2$	$exogen2$ $BMP2$	$exogen1$ $CanWnt$	$exogen2$ $CanWnt$	$canWnt$	$Bmp2$	$Foxc1/2$	$Mesp1$	$Dkk1$	$Tbx1$	$Fgf8$	$Isl1$	$GATAs$	$Nkx2\_5$	$Tbx5$
Inactive	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Active BMP	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0
FHF	1	1	0	0	0	1	0	0	0	0	0	0	1	1	1
FHF muted	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
SHF	1	1	1	1	1	0	1	0	0	1	1	1	1	1	0
SHF muted	0	0	1	1	1	0	1	1	1	1	0	1	1	1	0

Table 5.1: Attractors of the cardiac network

initial\target	Inactive	ActiveBMP	FHF	FHF muted	SHF	SHF muted
Inactive	$\emptyset$	$[A]$	$[A, C]; [C, A]$	$[C]$	$[A, B]; [B, A]$	$[B]$
ActiveBMP	$[A]$	$\emptyset$	$[C]$	$[A, C]; [C, A]$	$[B]$	$[A, B]; [B; A]$
FHF	$[A, C]; [C, A]$	$[C]$	$\emptyset$	$[A]$	$[B]$	$[A, B]; [B, A]$
FHF muted	$[C]$	$[A, C]; [C, A]$	$[A]$	$\emptyset$	$[A, B]; [B, A]$	$[B]$
SHF	$[A, B, C]; [B, C, A]$	$[B, C]$	$[B, C]$	$[A, B, C]; [B, C, A]$	$\emptyset$	$[A]$
SHF muted	$[B, C]$	$[A, B, C]; [B, C, A]$	$[A, B, C]; [B, C, A]$	$[B, C]$	$[A]$	$\emptyset$

Table 5.2: SCCs to perturb from static analysis for the cardiac network.

initial\target	Inactive	ActiveBMP	FHF	FHF muted	SHF	SHF muted
Inactive	0	1	2	1	2	1
ActiveBMP	1	0	1	2	1	2
FHF	3,4	3	0	1	1	2
FHF muted	2	3,4	1	0	2	1
SHF	(4) 7,8	(3) 6	(2) 4	(3) 5,7	0	1
SHF muted	(3) 8	(4) 7,9	(3) 5,6	(2) 6	1	0

Table 5.3: Total size of the perturbation sequences for the cardiac network for completely sequential, attractor-based, and one-step reprogramming strategies. Values inside parenthesis were only found with completely sequential reprogramming strategies.

initial\target	Inactive	ActiveBMP	FHF	FHF muted	SHF	SHF muted
Inactive	0	0.033	0.026	0.027	0.023	0.049
ActiveBMP	0.025	0	0.025	0.039	0.024	0.042
FHF	0.028	0.021	0	0.041	0.025	0.046
FHF muted	0.04	0.031	0.027	0	0.025	0.04
SHF	0.069	0.079	0.02	0.089	0	0.02
SHF muted	0.031	0.041	0.02	0.024	0.071	0

Table 5.4: Computation time for attractor-based strategies.

Comparison: By comparing table 5.3 and table 5.2, we can note that, in this example where SCCs are small, the static analysis algorithm returns very good results for minimal computations. If the initial state is SHF or SHF muted, all sequences returned by the static analysis algorithm are either shorter than the sequences for one-step and attractor-based reprogramming strategies, or in the worst case, as short as the best sequence returned by attractor-based reprogramming strategy. Indeed, even if the static analysis algorithm uses attractors as intermediate steps, the algorithm does not need to know which attractor will be reached after the perturbations. The attractor-based algorithm relies on knowing the intermediate steps, and thus, does not perform as well on networks with small SCCs. We can also note that when C needs to be perturbed, the static analysis often becomes under performing compared to the completely sequential algorithm. Because C is a bigger SCC, it is often possible to only perturb one or two variables inside to reprogram the system.

From SHF to FHF: We study the case of inevitable reprogrammability from SHF to FHF. They are both attractors of the network representing known phenotypes. The one-step perturbation sequence consists of perturbing $BMP2$, $canWnt$, and the two exogens of $canWnt$, thus perturbing four variables. However, smaller perturbation sequences exist, of size two, with two steps with perturbation valuations containing only one variable. The first perturbation, done in the SHF attractor, is always perturbing the first exogen of canWnt, $exogen\_canWnt\_I$.

After that, roughly a hundred (105) possible perturbation sequences are found. The second perturbation can be done on the second exogen of $BMP2$ or any gene except $canWnt$ (but not the exogens, with the exception of $exogen\_BMP2\_II$ as already mentioned). From this and the one-step perturbation sequence, we can understand that the perturbation the first exogen of canWnt will always lead to a change of value for the second one and canWnt, but that this is not enough for this strategy to be inevitable from SHF to FHF. If that was the case, the one-step perturbation sequence would only consists of $exogen\_canWnt\_I$. However, once that the perturbation from $exogen\_canWnt\_I$ propagated to $canWnt$, only one gene needs to be perturbed, depending how the system evolves.

5.2 Tumour network

An other network we studied is a mathematical modelling of molecular pathways enabling tumour cell invasion and migration, from Cohen et al. [CMR$^+$15]. This network aims at better understanding the role of individual or multiple mutations

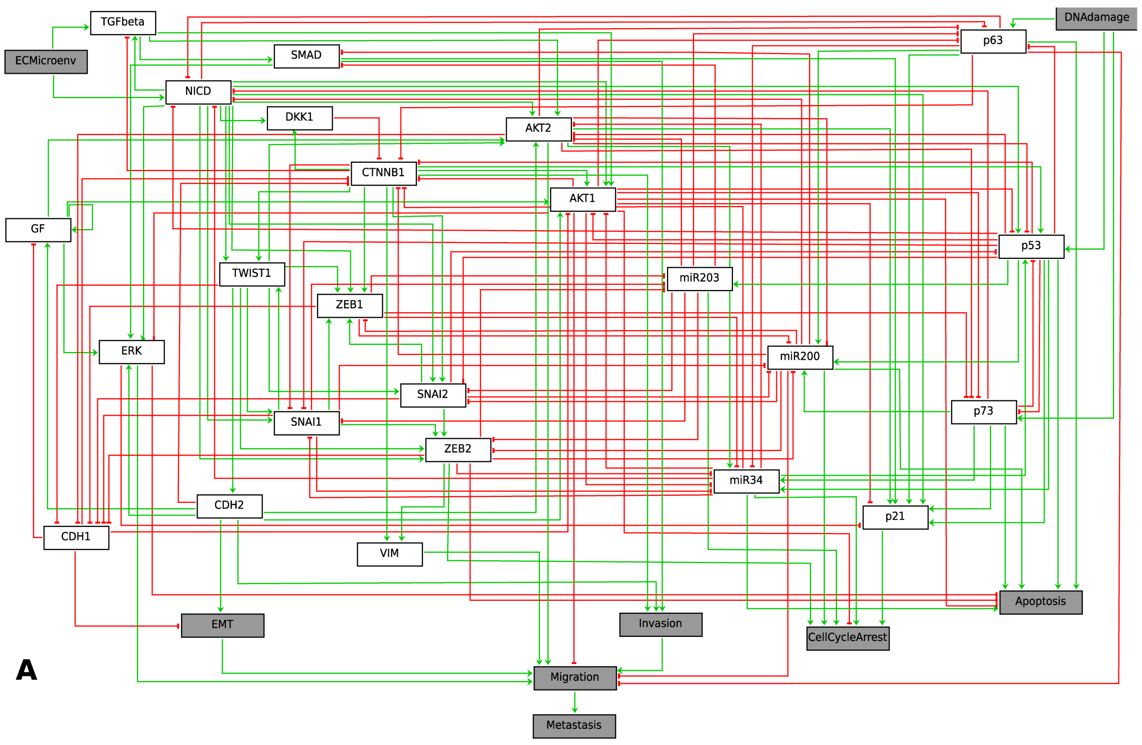

Figure 5.2: Interaction graph of the “tumour” Boolean network, from [CMR+15]

in the metastatic process, with attractors in three main categories: apoptosis, EMT, and migration. Lastly, an attractor HS represents the homeostatic state of the cell.

This model has 32 variables and 158 edges, which can be seen in fig. 5.2, and 9 attractors, 4 apoptosis, 2 EMT, 2 migration and one HS. These attractors are not listed explicitly here, since each is a list of 32 values of the variables, but a precise description of them can be found in [CMR+15].

Static analysis: This network is much more connected than the cardiac network. As a result, the SCCs are fewer and bigger. The inputs are two single state non-trivial SCCs, the outputs are all trivial SCCs, and four other variables are trivial SCCs (VIM,$p21$,ERK and $SMAD$). The remaining twenty variables all form a single non-trivial SCC. Thus, algorithm 3 will return very large perturbation sequences, except in some rare cases where only the input variables need to be modified. The only case where it happens is when trying to reach Migration 2 from any other attractor. Then either one or both inputs need to be perturbed, but no other variables.

	Apop1	Apop2	Apop3	Apop4	HS	EMT1	EMT2	Migra1	Migra2
Apop1	0	2	1	3	1	4,5,6	3,4,5,6	3,5,6,7	2
Apop2	2	0	3	1	1	(3) 4	3,4	3,5	2
Apop3	1	3	0	2	2	3,5,6,7	2,4,5,6,7	2,4,5,6	1
Apop4	3	1	2	0	2	3,5	2,4,5	2,4	1
HS	2	2	3	3	0	3,4,5	2,3	2,5	1
EMT1	3	5,7	4	6,8	4,6,7,8	0	1	1	2
EMT2	4	6,8	5	7,9	5,7	1	0	2	1
Migra1	4	6,8,9	3	5,7,8	5,7,8,9,10	1	2	0	1
Migra2	5	7,9,10	4	6,8,9	6,8,9	2	1	1	0

Table 5.5: Total size of the perturbation sequences for the tumour network for completely sequential, attractor-based, and one-step reprogramming strategies. Values inside parenthesis were only found with completely sequential reprogramming strategies.

Dynamical analysis: Table 5.5 contains the results from completely sequential reprogramming strategies, attractor-based sequential reprogramming strategies and one-step reprogramming strategies: if there is only one number in a case, then it means all methods return the same sequences, this is often the case when only a few perturbations are needed. It also means that the result is the same as if one-step reprogramming strategies were used. If there are multiple results, the first one is always the size of the smallest perturbation sequence for the completely sequential reprogramming strategies, and the list represent all perturbation sequences returned by the attractor-based sequential reprogramming strategies. When the first result is inside parenthesis, it means attractor-based sequential reprogramming strategies return longer perturbation sequences.

The time costs of computations range from 0.05 to 2.338 for attractor-based reprogramming strategies, which were mostly studied in this case, and up to 390 seconds for completely sequential strategies.

We can see that reprogramming from one attractor to an other in the same category is quite easy, the perturbation sequences are always of size 3 or less. Moreover, these perturbation sequences are always found by one-step reprogramming strategies. On the other hand, going from one category to an other is, in most cases, much more complicated. We can also see that in the best case, sequential reprogramming strategies and attractor-based sequential reprogramming strategies return sequences that can be three time smaller than one-step perturbation sequences in the case of Apoptosis 3 to EMT 2 or Migration 1.

However, looking the precise results show that the perturbed variables are often the inputs of the system, $DNAdamage$ and $ECMicroenv$, meaning it allows for better understanding of the artifacts of the model, but lacks in-depth

	Apoptosis	HS	EMT	Migration
Apoptosis	0	∞	3	4
HS	∞	0	3	∞
EMT	3	7	0	∞
Migration	3	∞	∞	0

Table 5.6: Shortest instantaneous perturbation sequences for inevitable reprogrammability with constraints on the perturbations, DNAdamage and ECMicroenv cannot be perturbed.

	Apoptosis	HS	EMT	Migration
Apoptosis	0	$\approx$33	10.158	57.01
HS	$\approx$33	0	10.828	$\approx$33
EMT	17.306	38.513	0	$\approx$33
Migration	147.963	$\approx$33	$\approx$33	0

Table 5.7: Computation times for inevitable reprogrammability with constraints on the instantaneous perturbations, DNAdamage and ECMicroenv cannot be perturbed.

insight on possible new perturbation sequences.

Constraints on the perturbations: As a result, we studied the system where these variables cannot be perturbed. We regrouped the states in the same category as a single target, meaning we only have four resulting attractor sets, Apoptosis, HS, EMT and Migration. This is because the different attractors in the same category often only differ by the values of the inputs.

We first studied inevitable reprogrammability with instantaneous perturbations, where $DNAdamage$ and $ECMicroenv$ cannot be perturbed. The shortest size of perturbation sequences is shown in table 5.6. The ∞ symbol means that the reprogramming is impossible, which is the case when $DNAdamage$ or $ECMicroenv$ has a different value in all attractors inside the initial category than in all attractors inside the target category. Computation times are given in table 5.7 in seconds. The computation when the reprogramming is impossible still allows for the 30 other variables to be perturbed, hence a computation time around 33 seconds.

Note that since attractors are grouped together, the results may seem confusing. It is possible to go from Migration to Apoptosis, and from Apoptosis to EMT, but impossible from Migration to Apoptosis. This is because $ECMicroenv$ is inactive in all attractors in EMT, but active in all attractors of Migration, pre-

	Apoptosis	HS	EMT	Migration
Apoptosis	0	∞	1	1
HS	∞	0	1	∞
EMT	1	1	0	∞
Migration	1	∞	∞	0

Table 5.8: Shortest permanent perturbation sequences for inevitable reprogrammability with constraints on the permanent perturbations.

	Apoptosis	HS	EMT	Migration
Apoptosis	0	-	1440.55	704.429
HS	-	0	1070.35	-
EMT	46239.9	1286	0	-
Migration	45369	-	-	0

Table 5.9: Computation times for inevitable reprogrammability with constraints on the permanent perturbations.

venting them to be reprogrammable to one another. However, it is active in some Apoptosis attractors, and inactive in other Apoptosis attractors, allowing both Migration and EMT to be reprogrammable to Apoptosis, and vice-versa.

We can see that most reprogrammings become harder than without the constraints, Apoptosis to Migration going from size 1 perturbation sequence to size 4 for example, or EMT to HS increasing from size 4 to size 7 perturbation sequences.

We can also see that going from a state to Apoptosis remains of the same difficulty that without the constraints. Because there are four attractors in Apoptosis, there always was one with the same inputs, where the perturbation sequences did not perturb $ECMicroenv$ or $DNAdamage$. These sequences are left unchanged by the constraints on $ECMicroenv$ and $DNAdamage$.

We then studied the network with even stronger restrictions on the perturbations, where $DNAdamage$ and $ECMicroenv$ cannot be perturbed as previously, and EMT, $Invasion$, $CellCycleArrest$, $Apoptosis$, $Migration$ and $Metastasis$ cannot be perturbed either. The results are shown in table 5.8. This shows how strong can permanent perturbations be. These perturbations often model mutations in the cell, thus showing that only a single mutation in the DNA can lead to disease. Table 5.9 shows the computation times required to compute these single perturbations, which can be very high (approximately 13 hours in some cases).

From HS to EMT We had an in-depth look at the HS to EMT reprogramming, with all the previous results. Without any constraints on the perturbations, the one-step perturbation sequence perturbs $CDH2$, $Twist1$ and $SNAI1$. The attractor-based sequence however only perturbs $ECMicroenv$, first activating it and then inhibiting it. Once we introduce constraints on the perturbations, the minimal size of the sequence is three. As a result, these sequences include the one-step perturbation sequence, which does not require to perturb any variable on which we have constraints. However, with permanent perturbations, this minimal size goes down to one, with more possibilities. If any variable among $Twist1$, $SNAI1$, $ZEB2$, $CDH1$, $AKT2$, $SNAI2$ and $ZEB1$ is permanently perturbed in HS, then EMT is the only reachable attractor.

5.3 PC12 Cell differentiation network

An other network we studied is PC12 cell differentiation network, used to understand cellular decisions towards proliferation or differentiation, explained and constructed in [OKS$^+$16]. It describes the interactions between protein signalling, transcription factor activity and gene regulatory feedback in PC12 cells after the stimulation of NGF, in order to understand more the cell differentiation.

The PC12 cell differentiation network contains 33 variables, and 62 edges and is shown in fig. 5.3. It has 7 attractors. One of which represents the differentiation, two the cell cycle arrest (CCA), and the 4 left are probably artifacts of the network, where the outcome is both Cell Cycle Arrest and Differentiation at the same time, or none.

Static Analysis: The interaction graph of the PC12 Boolean network contains three non-trivial SCCs. The first in topological order is $\{NGF\}$, the input. If set to active, only one attractor is reachable, where both cell cycle arrest and differentiation are active (arbitrarily named Both 1). The second SCC is rather big, with thirteen variables inside, that we will not list here. Lastly, the third SCC contains three variables, $\{Btg2, Klf4, CellCycleArrest\}$, with three possible configurations in attractors, one where both $Btg2$ and $Klf4$ are inactive, but $CellCycleArrest$ is active (all CCA and Both attractors), one where $Klf4$ is active, thus preventing $CellCycleArrest$ to be active (only present in the Differentiation attractor), and one where all variables are inactive (both "None" attractors).

The attractors differ only on the third SCC between CCA 1 and None 1 as well as on CCA 2 and None 2, thus we already know that the sequences will be of size 1. Moreover, the variables have approximately the same values for

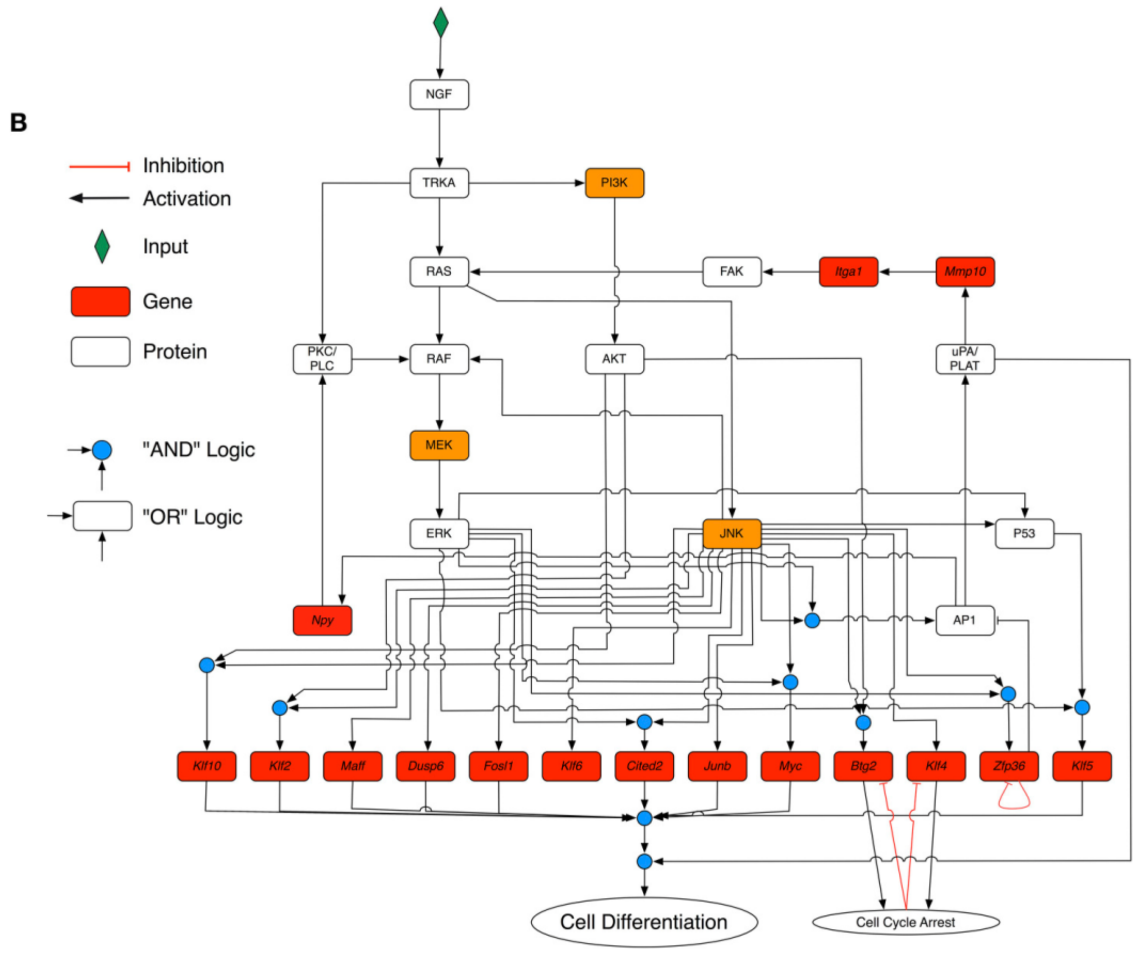

Figure 5.3: PC12 Boolean network, from [OKS+16]

these attractors, at the exception of two of them. As a result, we know that the sequences will be of size 2 maximum. We also know that only Both 1 is reachable when NGF is active, thus all sequences will consist of activating NGF and waiting for Both 1 to be reached.

However, the second SCC needs to be perturbed for all other perturbations, thus we know that the sequences will be bigger, but none will be bigger than 14.

Dynamical analysis: Table 5.10 contains the results for attractor-based and completely sequential reprogramming strategies with instantaneous perturbations, given in sizes of the sequences: as previously, if there is only one number in a case, then it means all strategies return the same sequences, this is often the case when only a few perturbations are needed. It also means that the re-

	Differentiation	CCA 1	CCA 2	None 1	None 2	Both 1	Both 2
Differentiation	0	8	9	7	8	1	1
CCA 1	(3) 4,5,6,10,11	0	1	1	2	1	2,6,10
CCA 2	(3) 4,5,6,10,11	1	0	2	1	1	2,6,10
None 1	(3) 4,6,10	1	2	0	1	1	2,3,4,6,9,10,11
None 2	(3) 4,6,10	2	1	1	0	1	2,3,4,6,9,10,11
Both 1	2.3	8,9	9,10	9,10	10,11	0	1
Both 2	1	7	8	8	9	1	0

Table 5.10: Total size of the perturbation sequences for the PC12 network for completely sequential, attractor-based, and one-step reprogramming strategies. Values inside parenthesis were only found with completely sequential reprogramming strategies.

sult is the same as if one-step reprogramming strategies were used. If there are multiple results, the first one is always the size of the smallest perturbation sequences for the completely sequential reprogramming strategies, and the list represent all perturbation sequences returned by the attractor-based sequential reprogramming strategies. When the first result is inside parenthesis, it means attractor-based sequential reprogramming strategies return longer perturbation sequences. The computation times range from 0.011 to 0.227 for attractor-based strategies, and from 0.183 to 5.8 for completely sequential strategies.

In this network, we can see that both attractor-based sequential reprogramming strategies and total sequential reprogramming strategies return new results with much smaller perturbation sequence size than one-step reprogramming strategies.

Cell cycle arrest to cell differentiation: Going from cell cycle arrest to cell differentiation would require a one-step perturbation sequence of size 11, whereas if sequentiality is used, the size of the perturbation sequences goes down to 3 or 4. For example, going from any cell cycle arrest to cell differentiation can be reduced to perturbing only NGF, then perturbing $PI3K$ and $CellCycleArrest$ and NGF once again. Please note that even the one-step perturbation sequence requires the perturbation of CellCycleArrest. With completely sequential reprogramming strategies, only NGF and $CellCycleArrest$ are perturbed, with NGF being perturbed twice.

As remarked in the static analysis paragraph, NGF has a very strong influence on the Boolean network, due to its place as an input. Moreover, we remarked that if CellCycleArrest is active, either it needs to be perturbed, or $Klf4$ needs to be. As a result, these perturbation sequences are not surprising, given the topology of the network's interaction graph.

Chapter 6

Discussion

Conclusion

In this thesis, we started by introducing cell differentiation, what is cell reprogramming, and what kind of systems are used to model the cell. We explained why we chose to work with Boolean networks, which do not require too much in-depth knowledge of the cell, and that model closely enough the dynamics of the studied systems. We also stated how we went from studying one-step reprogramming strategies to sequential ones during the PhD, and explained the content of this thesis and what to expect in each chapter.

We then defined formally what are perturbations in the context of Boolean networks. We also extended on these with perturbation sequences, to be able to use the dynamics of the network between the perturbations, a technique that is already used in biology but not extensively, and not studied much when working with Boolean networks. We then defined reprogramming strategies, how they can be translated to perturbation sequences, and how perturbation sequences (or set of sequences) can consist in different kinds of reprogramming strategies. We clarified the reprogrammability problem, by differentiating between existential and inevitable, showing why the distinction matters. In the first case, only some cells will be reprogrammed to the target, or none if some transitions are faster than others. In the second case, all cells will be reprogrammed to the target, or there is a problem with the model.

Next, we studied the Boolean network and its interaction graph, to try to find perturbation sequences with minimal computation times. It showed that the strongly connected components of the interaction graph have a lot of influence on the behavior of the graph, and that some heuristics can be used, which are very effective in graphs where there is no big strongly connected components. We showed that if a SCC contains no loop, then it is strongly influenced by the

SCCs that contain loops and are topologically preceding it. We designed an algorithm to perturb the SCCs in a given order, using sequentiality to shorten the size of the perturbation sequences. This algorithm performs fast and well on any graph where the SCCs with loops are small. However, in graphs with big strongly connected components, we showed that an analysis of the system's dynamics is needed to get smaller perturbation sequences, and we found some cases where the choice of variables to perturb can be reduced to improve computation time.

From these results, we focused on the study of the dynamics, and how to introduce the perturbations as part of the dynamics. To do so, we constructed a new transition graph with multiple layers corresponding to the number of perturbations. We made an algorithm which returns all perturbation sequences of size lower than a k given by the user, and an algorithm to extract which perturbation valuations correspond to this set of perturbation sequences. This algorithm returns a very complete list of solutions for instantaneous and permanent perturbations, but it is very slow, and therefore can only work on small networks. Lastly, we improved on one idea from the static analysis, to use attractors as intermediate steps. This allows for a much quicker algorithm, which returns shorter perturbation sequences than other methods in some cases. However, the results are both not as many and not as short as the first algorithm.

Lastly, we studied the results of our algorithms on examples from literature. We showed how performing can be the static analysis algorithm when used on a network with small SCCs, performing better than the attractor-based algorithm in a lot of cases, and outperforming one-step reprogrammability algorithms. We studied the specific case of cell reprogramming from SHF to FHF in the cardiac gene regulatory network, where the completely sequential algorithm returns shorter perturbation sequences than one-step reprogrammability. We also studied a network modelling the tumour enabling pathways, in which we found that the inputs have a very strong influence, and that preventing them to change values results in new, longer perturbation sequences, but that permanent perturbations allow to reprogramm to any target with only one variable perturbed. We also studied the PC12 differentiation network, in which we showed how the interaction graph can give insight on the variables returned in perturbation sequences.

Perspectives

Chronologically, the first study was the static analysis, which helped understand the need for dynamics analysis. We then heavily focused on the dynamics analysis, but there might be more that can be found in the static analysis. When writing this thesis, the static analysis was explored more in-depth, using the dynamics of the network to improve results. However, we mainly use the interaction graph, which contains less information than the Boolean network. Working more closely to the Boolean network might help reduce the number of possible perturbation valuations. We also think that there some very good heuristics to choose which variables to perturb can be found in the interaction graph. We noticed that in the case of inevitable reprogrammability, there exists a strategy where each perturbation valuation contain only variables in the non-trivial SCCs. But in the general case, variables in non-trivial SCCs can be perturbed to help reach quicker the target attractor, thus forcing strategies to be inevitable. We wish we had more time to focus on the study of the Boolean network, and finding pattern in it which would help find inevitable strategies.

Chronologically, we also defined reprogramming sequences only lately, we wish we did it earlier, as it helped us understand that one of the most optimal structure for the strategies are graphs, and thus should have been the kind of results returned by the algorithms.

Concerning the dynamics, we had multiple projects to improve the algorithm for completely sequential reprogrammability. We wanted to find ways to exploit the concurrency of some parts of the Boolean network, allowing for faster algorithms. We also wanted to compute on-the-fly parts of the transition graph, once again to reduce computation times, and to allow some solutions to be returned quickly, while the algorithm is not yet over.

Algorithm 6 was roughly implemented in python as a prototype, but we lacked time to properly implement it. The version used in this paper computes all paths, which is much worse complexity-wise, preventing in-depth analysis when there are too many solutions.

We did not find an easy way to encode permanent perturbations in the case of attractor-based sequential reprogrammability. It would require to compute a specific basin, which is not easily described, both smaller than the set of predecessors and bigger than the basin as defined in chapter 2, and we found no easy way to compute this basin.

Lastly, we often considered full observability, or in the case of attractor-based reprogramming strategies, observability of the attractor when needed. Limiting observability is a very interesting problem, either when the observability is set, to find the best reprogramming strategy with this observability ; or, to find the observability required (which variables should be observed) in order to have small perturbation sequences.

Bibliography

[AJMN$^+$15] Wassim Abou-Jaoudé, Pedro T. Monteiro, Aurélien Naldi, Maximilien Grandclaudon, Vassili Soumelis, Claudine Chaouiya, and Denis Thieffry. Model checking to assess T-helper cell plasticity. *Frontiers in Bioengineering and Biotechnology*, 2, 2015.

[Ara08] Julio Aracena. Maximum number of fixed points in regulatory boolean networks. *Bulletin of Mathematical Biology*, 70(5):1398–1409, feb 2008.

[BD17] Célia Biane and Franck Delaplace. Abduction Based Drug Target Discovery Using Boolean Control Network. In *15th International Conference on Computational Methods in Systems Biology (CMSB 2017)*, volume 10545 of *Lecture Notes in Computer Science*, pages 57–73, Darmstad, Germany, September 2017. H. Koeppl and J. Feret.

[CHJ$^+$14] Thomas Chatain, Stefan Haar, Loïg Jezequel, Loïc Paulevé, and Stefan Schwoon. Characterization of reachable attractors using Petri net unfoldings. In Pedro Mendes, Joseph Dada, and Kieran Smallbone, editors, *Computational Methods in Systems Biology*, volume 8859 of *Lecture Notes in Computer Science*, pages 129–142. Springer Berlin Heidelberg, 2014.

[CMR$^+$15] David P. A. Cohen, Loredana Martignetti, Sylvie Robine, Emmanuel Barillot, Andrei Zinovyev, and Laurence Calzone. Mathematical modelling of molecular pathways enabling tumour cell invasion and migration. *PLOS Computational Biology*, 11(11):1–29, 11 2015.

[CNRT11] C. Chaouiya, A. Naldi, E. Remy, and D. Thieffry. Petri net representation of multi-valued logical regulatory graphs. *Natural Computing*, 10(2):727–750, 2011.

[CPJdS13] Isaac Crespo, Thanneer M Perumal, Wiktor Jurkowski, and Antonio del Sol. Detecting cellular reprogramming determinants by differential stability analysis of gene regulatory networks. *BMC Systems Biology*, 7(1):140, 2013.

[CSW11] Rui Chang, Robert Shoemaker, and Wei Wang. Systematic search for recipes to generate induced pluripotent stem cells. *PLoS Computational Biology*, 7(12):e1002300, Dec 2011.

[GTW03] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research (Lecture Notes in Computer Science)*. Springer, 2003.

[HAK16] R Hannam, A Annibale, and R Kuehn. Cell reprogramming modelled as transitions in a hierarchy of cell cycles. *ArXiv e-prints*, page 425601, December 2016.

[HGZ$^+$12] Franziska Herrmann, Alexander Groß, Dao Zhou, Hans A. Kestler, and Michael Kühl. A boolean model of the cardiac gene regulatory network determining first and second heart field identity. *PLOS ONE*, 7(10):1–10, 10 2012.

[KFG95] H Kulessa, J Frampton, and T Graf. Gata-1 reprograms avian myelomonocytic cell lines into eosinophils, thromboblasts, and erythroblasts. *Genes & Development*, 9(10):1250–1262, 1995.

[LLCM14] Alex H. Lang, Hu Li, James J. Collins, and Pankaj Mehta. Epigenetic landscapes explain partially reprogrammed cells and identify key reprogramming genes. *PLOS Computational Biology*, 10(8):1–13, 08 2014.

[Men06] Luis Mendoza. A network model for the control of the differentiation process in th cells. *Biosystems*, 84(2):101 – 114, 2006. Dynamical Modeling of Biological Regulatory Networks.

[MHP16] Hugues Mandon, Stefan Haar, and Loïc Paulevé. Relationship between the reprogramming determinants of boolean networks and their interaction graph. In Eugenio Cinquemani and Alexandre Donzé, editors, *Hybrid Systems Biology*, pages 113–127, Cham, 2016. Springer International Publishing.

[MHP17] Hugues Mandon, Stefan Haar, and Loïc Paulevé. Temporal reprogramming of boolean networks. In Jérôme Feret and Heinz Koeppl, editors, *Computational Methods in Systems Biology*, pages 179–195, Cham, 2017. Springer International Publishing.

[MPQY18] Andrzej Mizera, Jun Pang, Hongyang Qu, and Qixia Yuan. Assa-pbn 3.0: Analysing context-sensitive probabilistic boolean networks. In Milan Češka and David Šafránek, editors, *Computational Methods in Systems Biology*, pages 277–284, Cham, 2018. Springer International Publishing.

[MSH$^+$19] Hugues Mandon, Cui Su, Stefan Haar, Jun Pang, and Loïc Paulevé. Sequential Reprogramming of Boolean Networks Made Practical. In *CMSB 2019 - 17th International Conference on Computational Methods in Systems Biology*, Trieste, France, September 2019.

[MSP$^+$19] H. Mandon, C. Su, J. Pang, S. Paul, S. Haar, and L. Paulevé. Algorithms for the sequential reprogramming of boolean networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pages 1–1, 2019.

[MSRSL10] Melody K. Morris, Julio Saez-Rodriguez, Peter K. Sorger, and Douglas A. Lauffenburger. Logic-based models for the analysis of cell signaling networks. *Biochemistry*, 49(15):3216–3224, 2010. PMID: 20225868.

[OKS$^+$16] B. Offermann, S. Knauer, A. Singh, M. L. Fernández-Cachón, M. Klose, S. Kowar, H. Busch, and M. Boerries. Boolean modeling reveals the necessity of transcriptional regulation for bistability in PC12 cell differentiation. *Frontiers in Genetics*, 7:44, 2016.

[PSPM18] S. Paul, C. Su, J. Pang, and A. Mizera. A decomposition-based approach towards the control of Boolean networks. In *Proc. 9th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 11–20. ACM Press, 2018.

[PSPM19] S. Paul, C. Su, J. Pang, and A. Mizera. An efficient approach towards the source-target control of boolean networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pages 1–1, 2019.

[Rod08] K.T. Rodolfa. Inducing pluripotency (september 30, 2008). In *StemBook*. The Stem Cell Research Community, 2008. StemBook, doi/10.3824/stembook.1.22.1, http://www.stembook.org.

[RPM$^+$17] Scott Ronquist, Geoff Patterson, Lindsey A. Muir, Stephen Lindsly, Haiming Chen, Markus Brown, Max S. Wicha, Anthony Bloch, Roger Brockett, and Indika Rajapakse. Algorithm for cellular reprogramming. *Proceedings of the National Academy of Sciences*, 114(45):11832–11837, 2017.

[RRT08] Élisabeth Remy, Paul Ruet, and Denis Thieffry. Graphic requirements for multistability and attractive cycles in a boolean dynamical framework. *Advances in Applied Mathematics*, 41(3):335 – 350, 2008.

[SFL$^+$09] Ozgur Sahin, Holger Frohlich, Christian Lobke, Ulrike Korf, Sara Burmester, Meher Majety, Jens Mattern, Ingo Schupp, Claudine Chaouiya, Denis Thieffry, Annemarie Poustka, Stefan Wiemann, Tim Beissbarth, and Dorit Arlt. Modeling ERBB receptor-regulated G1/S transition to find novel targets for de novo trastuzumab resistance. *BMC Systems Biology*, 3(1):1–20, 2009.

[SVKK10] Regina Samaga, Axel Von Kamp, and Steffen Klamt. Computing combinatorial intervention strategies and failure modes in signaling networks. *Journal of Computational Biology*, 17(1):39–53, Jan 2010.

[Tho73] René Thomas. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology*, 42(3):563 – 585, 1973.

[TTO$^+$07] Kazutoshi Takahashi, Koji Tanabe, Mari Ohnuki, Megumi Narita, Tomoko Ichisaka, Kiichiro Tomoda, and Shinya Yamanaka. Induction of pluripotent stem cells from adult human fibroblasts by defined factors. *Cell*, 131(5):861 – 872, 2007.

[TY16] Kazutoshi Takahashi and Shinya Yamanaka. A decade of transcription factor-mediated reprogramming to pluripotency. *Nat Rev Mol Cell Biol*, 17(3):183–193, Feb 2016.

[XYFG04] Huafeng Xie, Min Ye, Ru Feng, and Thomas Graf. Stepwise reprogramming of b cells into macrophages. *Cell*, 117(5):663 – 676, 2004.

[ZA15] Jorge G. T. Zañudo and Réka Albert. Cell fate reprogramming by control of intracellular network dynamics. *PLOS Computational Biology*, 11(4):1–24, 04 2015.

Appendix A

Encoding asynchronous Boolean networks and sequential perturbations

This appendix explains in details what Safe Petri Nets are, how they are used to model a reprogramming strategy $\mathcal{S}$ of a Boolean network, and how from this modelling, we can extract $\mathtt{STG}(\mathcal{S})$.

Definition 26 (Safe Petri Net)**.** *A* Petri net *is a tuple* (P, T, A, M_0) *where* P *and* T *are sets of nodes, called* places *and* transitions *respectively, and* $A \subseteq (P \times T) \cup (T \times P)$ *is a* flow relation *whose elements are called* arcs*. A subset* $M \subseteq P$ *of the places is called a marking, and* M_0 *is a distinguished* initial marking*.*

For any node $u \in P \cup T$*, we call* pre-set *of* u *the set* ${}^\bullet u = \{v \in P \cup T \mid (v, u) \in A\}$ *and* post-set *of* u *the set* $u^\bullet = \{v \in P \cup T \mid (u, v) \in A\}$*.*

A transition $t \in T$ *is* enabled *at a marking* M *if and only if* ${}^\bullet t \subseteq M$*. The application of such a transition leads to the new marking* $M' = (M \setminus {}^\bullet t) \cup t^\bullet$*, and is denoted by* $M \xrightarrow{t} M'$*. A marking* M' *is* reachable *if there exists a sequence of transitions* $t_1, \ldots, t_k$ *such that* $M_0 \xrightarrow{t_1} \ldots \xrightarrow{t_k} M'$*.*

A Petri net is safe *if and only if any reachable marking* M *is such that for any* $t \in T$ *that can fire from* M *leading to* M'*, the following property holds:* $\forall p \in M \cap M', p \in {}^\bullet t \cap t^\bullet \vee p \notin {}^\bullet t \cup t^\bullet$*.*

Less formally, a safe Petri Net is a Petri Net where in all reachable markings from the initial marking, all places have at most one token. A subset of places $\{p_1, \ldots, p_k\} \subseteq P$ is *mutually exclusive* if every reachable marking M contains at most one these place.

A.1 Encoding asynchronous Boolean networks

The equivalent representation of the asynchronous semantics of a Boolean network of dimension n $f = (f_1, \cdots, f_n)$ in Petri net has been addressed in [CNRT11, CHJ$^+$14]. Essentially, to each variable $i \in [1, n]$ of the Boolean network f is associated two places, i_0 and i_1, acting respectively for the variable i being inactive and active. Then, transitions are derived from clauses of the Disjunctive Normal Form (DNF; disjunction of conjunctive clauses) representation of $[\neg x_i \wedge f_i(x)]$ for i activation, and from $[x_i \wedge \neg f_i(x)]$ for i inactivation.

Given a logical formula $[e]$, we write $\mathrm{DNF}[e]$ for its DNF representation. $\mathrm{DNF}[e]$ is thus a set of clauses, where clauses are sets of literals. A literal correspond to the state of a node, and is either of the form x_i (node i is active), or $\neg x_i$ (node i is inactive). Given such a literal l, $\mathrm{place}(l)$ associates the corresponding Petri net place: $\mathrm{place}([x_i]) = i_1$ and $\mathrm{place}([\neg x_i]) = i_0$.

The safe Petri net encoding the asynchronous semantics of a Boolean network f is defined as follows.

Definition 27 $(\mathrm{PN}(f))$**.** *Given a Boolean network f of dimension n and an initial state $x \in \{0, 1\}^n$, $\mathrm{PN}(f) = (P_f, T_f, A_f, M_0)$ is the corresponding Safe Petri Net such that:*

- $P_f = \bigcup_{i \in [1,n]} \{i_0, i_1\}$ *is the set of places,*
- T_f *and* A_f *are the smallest sets which satisfy, for each* $i \in [1, n]$*,*
  - *for each clause* $c \in \mathrm{DNF}[\neg x_i \wedge f_i(x)]$*, there is a transition* $t_{i,c} \in T_f$ *with* A_f *such that* ${}^\bullet t_{i,c} = \{\mathrm{place}(l) \mid l \in c\}$ *and* $t_{i,c}{}^\bullet = \{i_1\} \cup {}^\bullet t_{i,c} \setminus \{i_0\}$*;*
  - *for each clause* $c \in \mathrm{DNF}[x_i \wedge \neg f_i(x)]$*, there is a transition* $t_{\neg i,c} \in T_f$ *with* A_f *such that* ${}^\bullet t_{\neg i,c} = \{\mathrm{place}(l) \mid l \in c\}$ *and* $t_{\neg i,c}{}^\bullet = \{i_0\} \cup {}^\bullet t_{\neg i,c} \setminus \{i_1\}$*,*
- $M_0 = \{i_{x_i} \mid i \in [1, n]\}$ *is the initial marking.*

Note that [CHJ$^+$14] also extends the encoding to multi-valued networks into 1-bounded Petri nets (contrary to the encoding of multi-valued networks of [CNRT11] which does not result in a safe Petri net). For the sake of simplicity, we restrict the presentation to Boolean networks. However, our encoding of sequential perturbations can be easily extended to multi-valued networks.

Example 16. *Fig. A.1 gives the resulting Petri net encoding of the Boolean function $f_3(x) = x_1 \wedge \neg x_2$. In this case,* $\mathrm{DNF}[\neg x_3 \wedge (x_1 \wedge \neg x_2)] = \{\{\neg x_3, x_1, \neg x_2\}\}$ *and* $\mathrm{DNF}[x_3 \wedge (\neg x_1 \vee x_2)] = \{\{x_3, \neg x_1\}, \{x_3, x_2\}\}$*.*

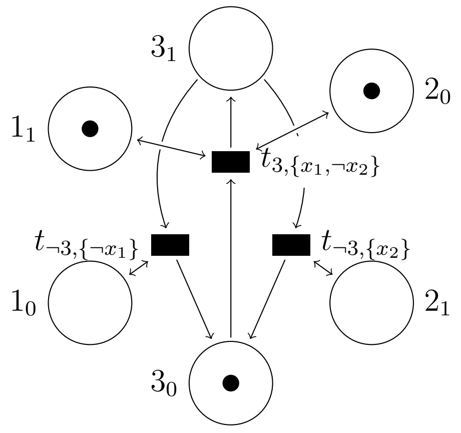

Figure A.1: Safe Petri net encoding of $f_3(x) = x_1 \wedge \neg x_2$. Places are drawn as circles and transitions as rectangles. Marked places have a dot.

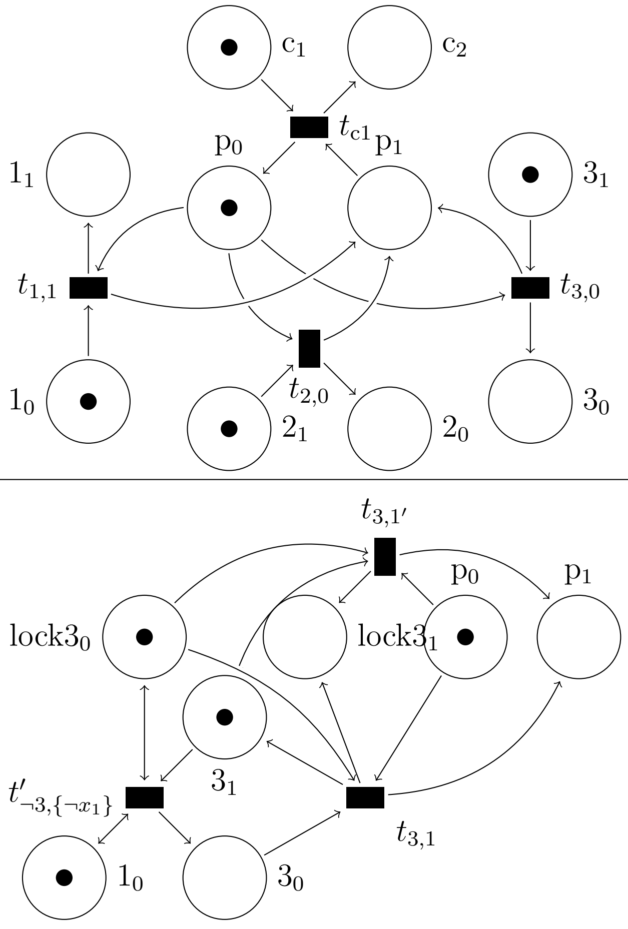

Figure A.2: (top) Excerpt of the encoding of temporary perturbations. (bottom) Excerpt of the encoding of permanent perturbations.

A.2 Encoding sequential perturbations

Perturbations are modelled as additional transitions which modify the value of variables of the BN f. These perturbations can be performed at any time during the transient dynamics, and independently of the current state of the network.

As in chapter 3, we consider three kinds of perturbations: *instantaneous perturbations* induce a change of value of some variables, but these variables can later be updated according to their Boolean function. Such perturbations can model, for instance, the transient activation of transcription factor through a signalling pathway. *Permanent perturbations* induce a permanent change of value of some variables. These perturbations model mutations (loss or gain of functions). Lastly, *temporary perturbations* induce a reversible change of value of some variables. It behaves as a permanent perturbation, but can be “unlocked” to allow the nodes to be updated according to their Boolean function. This kind of perturbations can model environmental factors, such as the constant afflux of a protein, which stops after some time.

Whichever the kind, we consider a limited amount of allowed perturbations: only up to k successive perturbations can be performed. Moreover, we only consider perturbations where one variable is perturbed. Doing so allows us to control the total size of the perturbations, and one two-variables perturbation is easy to model with two one-variable perturbations.

A.2.1 Instantaneous perturbations:

In addition to the places for the BN variable values, we add k mutually exclusive places $\mathrm{c}_1, \dots, \mathrm{c}_k$ and two mutually exclusive places p_0 and p_1. Essentially, c_j is marked if the next perturbation is the j-th; and p_0 is marked if the j-th perturbation is yet to be performed, and p_1 is marked if the j-th perturbation has been performed.

The transitions are the same as the asynchronous model, with additional transitions $t_{i,0}$ and $t_{i,1}$ for each variable $i \in [1, n]$ which set their value to 0 and 1 respectively. To be enabled, these transitions need p_0 to be marked, and after the transition, p_1 is marked. Finally, a transition $t_{\mathrm{c}j}$ re-enabling p_0 is defined for each c_j, $j \in [1, k-1]$, which moves the marking of c_j to c_{j+1}.

Definition 28. *Given a Boolean network f of dimension n, the Petri net (P, T, A, M_0) modelling its k temporary perturbations is given by*

- $P = P_f \cup \{\mathrm{p}_0, \mathrm{p}_1, \mathrm{c}_1, \dots, \mathrm{c}_k\}$,
- T *and* A *are the smallest sets which satisfy*

(a) BN transitions *$T_f \subseteq T$, $A_f \subseteq A$;*

(b) Perturbation transitions *for $i \in [1, n]$,*
$t_{i,0} \in T$ *with* $^\bullet t_{i,0} = \{i_1, \mathrm{p}_0\}$ *and* $t_{i,0}{}^\bullet = \{i_0, \mathrm{p}_1\}$
$t_{i,1} \in T$ *with* $^\bullet t_{i,1} = \{i_0, \mathrm{p}_0\}$ *and* $t_{i,1}{}^\bullet = \{i_1, \mathrm{p}_1\}$;

(c) Perturbation enabling *for $j \in [1, k-1]$,*
$t_{\mathrm{c}j} \in T$ *with* $^\bullet t_{\mathrm{c}j} = \{\mathrm{p}_1, \mathrm{c}_j\}$ *and* $t_{\mathrm{c}j}{}^\bullet = \{\mathrm{p}_0, \mathrm{c}_{j+1}\}$,

- $M_0 = \{i_{x_i} \mid i \in [1, n]\} \cup \{\mathrm{p}_0, \mathrm{c}_1\}$,

where $(P_f, T_f, A_f, M_0') = \mathrm{PN}(f)$.

Example 17. *Fig. A.2(top) shows part of the transitions added by the modelling of $k = 2$ instantaneous perturbations in the example of Fig. A.1. In the given marking, the perturbation are enabled, therefore, any of the 3 shown perturbation transitions can be applied. The application of one such transition disable the other perturbation transitions (as p_0 is no longer marked). By applying the transition $t_{\mathrm{c}1}$, the perturbations transitions are then re-enabled, allowing a second (and last) one to be applied.*

A.2.2 Permanent and temporary perturbations:

Contrary to instantaneous perturbations, once a variable has been (permanently or temporarily) perturbed, its value should no longer change. This is modelled by *locks*: if the i-th lock is active the variable i cannot perform any transition, and as such, cannot change value. In addition to the places introduced for temporary perturbations, our encoding add mutually exclusive places $\mathrm{lock}_i^0, \mathrm{lock}_i^1$ for each each variable $i \in [1, n]$, lock_i^0 being marked if the node i has not been perturbed, lock_i^1 being marked otherwise.

The transitions of the Boolean network are then modified so that a transition changing the value of variable i requires the place lock_i^0 to be marked. For each variable i, 4 perturbations transitions are defined: two for the value changes (0 to 1 and 1 to 0) also inducing the marking of lock_i^1; and two for the marking of lock_i^1 without value change: indeed, a perturbation does not necessarily have to change the current value of the variable, but it prevents any further evolution of it.

For temporary perturbations, one more perturbation transition is defined for each variable i: the “unlocking” of i, where lock_i^0 is marked, allowing the variable to be updated according to the Boolean function, once again.

Definition 29. *Given a Boolean network f of dimension n, the Petri net (P, T, A, M_0) modelling its k permanent perturbations is given by*

- $P = P_f \cup \{\mathrm{p}_0, \mathrm{p}_1, \mathrm{c}_1, \dots, \mathrm{c}_k\} \cup \bigcup_{i\in[1,n]}\{\mathrm{lock}i_0, \mathrm{lock}i_1\}$
- *T and A are the smallest sets which satisfy*

  BN transitions *$\forall t_{l,c} \in T_f$, with $l = i$ or $l = \neg i$, $i \in [1,n]$,*
  $t'_{l,c} \in T$ with $^\bullet t'_{l,c} = {}^\bullet t_{l,c} \cup \{\mathrm{lock}i_0\}$ and $t'_{l,c}{}^\bullet = t_{l,c}{}^\bullet \cup \{\mathrm{lock}i_0\}$

  Perturbation transitions *for $i \in [1,n]$,*
  $t_{i,0} \in T$ with $^\bullet t_{i,0} = \{i_1, \mathrm{p}_0, \mathrm{lock}i_0\}$ and $t_{i,0}{}^\bullet = \{i_0, \mathrm{p}_1, \mathrm{lock}i_1\}$
  $t_{i,0'} \in T$ with $^\bullet t_{i,0'} = \{i_0, \mathrm{p}_0, \mathrm{lock}i_0\}$ and $t_{i,0'}{}^\bullet = \{i_0, \mathrm{p}_1, \mathrm{lock}i_1\}$
  $t_{i,1} \in T$ with $^\bullet t_{i,1} = \{i_0, \mathrm{p}_0, \mathrm{lock}i_0\}$ and $t_{i,1}{}^\bullet = \{i_1, \mathrm{p}_1, \mathrm{lock}i_1\}$
  $t_{i,1'} \in T$ with $^\bullet t_{i,1'} = \{i_1, \mathrm{p}_0, \mathrm{lock}i_0\}$ and $t_{i,1'}{}^\bullet = \{i_1, \mathrm{p}_1, \mathrm{lock}i_1\}$

  Perturbation enabling *for $j \in [1, k-1]$,*
  $t_{\mathrm{c}j} \in T$ with $^\bullet t_{\mathrm{c}j} = \{\mathrm{p}_1, \mathrm{c}_j\}$ and $t_{\mathrm{c}j}{}^\bullet = \{\mathrm{p}_0, \mathrm{c}_{j+1}\}$
- $M_0 = \{i_{x_i} \mid i \in [1,n]\} \cup \{\mathrm{p}_0, \mathrm{c}_1\}$

where $(P_f, T_f, A_f, M'_0) = \mathrm{PN}(f)$.

Example 18. *Fig. A.2(bottom) shows part of the transitions added by the modelling of $k = 2$ permanent perturbations. The transition $t_{3,\{\neg x_1\}}$ of Fig. A.1 is modified so that it is enabled only if $\mathrm{lock}3_0$ is marked, i.e., the node 3 has not been perturbed yet. Permanent perturbation transitions $t_{3,1}$ and $t_{3,1'}$ lock the node 3 to its value 1. Once applied, none of the transitions modifying the value of node 3 can be enabled. Transitions for re-enabling perturbations are identical to the temporary case.*

A.3 State Transition Graph

Given a BN f and an initial state x, the above modelling allows to compute all the states reachable by any combination and succession of k perturbations, instantaneous, temporary or permanent.

The explicit state transition graph resulting needed for chapter 4 is composed of two classes of transitions: the transitions induced by BN f, and the transitions induced by the perturbation.

It can be remarked that our encoding uses an additional kind of transition: the transitions for re-enabling the perturbation transitions, when strictly less than k perturbations have been applied (transitions noted $t_{\mathrm{c}j}, j \in [1, k-1]$). These transitions are artefacts of the modelling, and can be skipped during the state transition graph construction.

Let us define a state transition graph among states S with two classes of transitions $\mathcal{E}$ (induced by f) and $\mathcal{M}$ (induced by the perturbations), as the smallest digraph $(S, \mathcal{E}, \mathcal{M})$ such that $M_0 \in S$, and for each $M \in S$, for each $t \in T$ such that ${}^\bullet t \subseteq M$, let $M' = (M \setminus {}^\bullet t) \cup t^\bullet$,

- if $\mathrm{p}_1 \in M$ and $\mathrm{c}_k \notin M$, then $\exists j \in [1, k] : {}^\bullet t_{\mathrm{c}j} \in M'$; let $M'' = (M' \setminus {}^\bullet t_{\mathrm{c}j}) \cup t_{\mathrm{c}j}{}^\bullet$, $M'' \in S$ and $(M, M'') \in E$,
- otherwise, $M' \in S$, and if $t = t_{l,c}$, then $(M, M') \in \mathcal{E}$, else $(M, M') \in \mathcal{M}$.

Given any marking $M \in S$ of the resulting state transition graph, the number of perturbations applied to reach M is given by $j + b$ where $\mathrm{c}_j \in M$ and $\mathrm{p}_b \in M$.

Titre: Algorithmes pour la prédiction de stratégies de reprogrammation cellulaire dans les réseaux Booléens

Mots clés: Réseaux Booléens, Reprogrammation cellulaire,Biologie des systèmes

Résumé:
Cette thèse explique ce qu'est la reprogrammation cellulaire dans le cadre des réseaux Booléens, et quelles sont différentes méthodes pour obtenir des solutions à ce problème.
Premièrement, on y établit formellement les définitions de perturbations, séquences de perturbations, stratégies de reprogrammation, inévitabilité et existentialité desdites stratégies, et ce qu'est la reprogrammabilité, dans le cadre des réseaux Booléens. De plus, il y est listé les techniques actuelles pour trouver les cibles de reprogrammation cellulaire, dans le cadre des réseaux Booléens et d'autres modèles.
On y décrit ensuite comment une analyse statique du réseau permet de comprendre mieux leur dynamique, et le rôle important des composantes fortement connexes du graphe d'interaction. À partir de ce réseau et de la donnée externe de la liste des attracteurs, un algorithme permet de trouver certaines variables à perturber, parfois nécessairement dans un ordre donné.
Ensuite, on y explique comment construire un nouveau modèle pour pouvoir faire des perturbations de manière purement séquentielle, et ainsi profiter de la dynamique du réseau Booléen entre les perturbations. Etant donnée la complexitée élevée de cette approche, on y explique également une approche intermédiaire, ou seulement les attracteurs du réseaux sont perturbables, permettant une complexité plus faible.
Enfin, une étude de cas est faite, où divers réseaux Booléens biologiques sont utilisés, avec les différentes approches expliquées au long de la thèse. On y constate que les stratégies de reprogrammation séquentielles permettent de trouver des séquences de perturbations différentes, avec des perturbations plus petites que si l'on ne perturbe qu'une seule fois.

Title: Algorithms for Cell Reprogramming Strategies in Boolean Networks

Keywords: Boolean networks, Cell Reprogramming, Systems Biology

Abstract:
This thesis explains what is cell reprogramming in Boolean networks, and what are several methods to solve this problem.
First, formal definitions of perturbations, perturbation sequences, reprogramming strategies, inevitability and existentiality of the strategies, and of reprogrammability are given, in the scope of Boolean networks. Moreover, a list of actual methods to find cell reprogramming targets is given, both in the scope of Boolean networks and outside of it.
Then, it is described how a static analysis of the networks allows for better understanding of their dynamics, and how important strongly connected components of the interaction graph are. From this network with the added information of the attractor list, an algorithm finds a list of variables to perturb, sometimes with the necessity of a precise order.
Then, how to construct a new model is explained, allowing to make perturbations sequentially, thus using the Boolean network dynamics between the perturbations. Given the high complexity of this approach, we also explain an in-between approach, where only the attractors of the network can be perturbed, thus allowing for a smaller complexity.
Lastly, a case study is done, where biological Boolean networks from literature are used, and on which the different algorithms from the thesis are applied. We show that sequential reprogramming strategies allow for new perturbation sequences, with smaller perturbations than one-step reprogramming strategies.