SPIM

Habilitation à Diriger des Recherches

UFC

école doctorale **sciences pour l'ingénieur et microtechniques**

UNIVERSITÉ DE FRANCHE-COMTÉ

Title

FIRST NAME

SPIM

Habilitation à Diriger des Recherches

école doctorale **sciences pour l'ingénieur et microtechniques**

UFC UNIVERSITÉ DE FRANCHE-COMTÉ

HABILITATION À DIRIGER DES RECHERCHES

de l'Université de Franche-Comté

préparée au sein de l'Université de Technologie de Belfort-Montbéliard

Spécialité : **Informatique**

présentée par

FIRST NAME

Title

Soutenue publiquement le XX Mois XXXX devant le Jury composé de :

FIRST NAME	Rapporteur	Professeur à l'Université de XXX
FIRST NAME	Examinateur	Professeur à l'Université de XXX

INTRODUCTION

Blabla blabla.

I

RÉSEAUX DISCRETS

1

ITERATIONS DISCRÈTES DE RÉSEAUX BOOLÉENS

chapeau à refaire

1.1/ FORMALISATION

Chapeau chapitre à faire

On considère l'espace booléen $\mathbb{B} = \{0, 1\}$ muni des opérateurs binaires de disjonction « + », de conjonction « . » et unaire de négation « $\overline{\ }$ ».

Soit N un entier naturel. On introduit quelques notations à propos d'éléments de $\mathbb{B}^{\mathsf{N}}$. L'ensemble $\{1, \ldots, \mathsf{N}\}$ sera par la suite noté [N]. Le $i^{\text{ème}}$ composant d'un élément $x \in \mathbb{B}^{\mathsf{N}}$ s'écrit x_i. Si l'ensemble I est une partie de [N], alors $\overline{x}^I$ est l'élément $y \in \mathbb{B}^{\mathsf{N}}$ tel que $y_i = 1 - x_i$ si $i \in I$ et $y_i = x_i$ sinon. On considère les deux abréviations $\overline{x}$ pour $\overline{x}^{[\mathsf{N}]}$ (chaque composant de $\overline{x}$ est nié : c'est une négation composante à composante) et $\overline{x}^i$ pour $\overline{x}^{\{i\}}$ pour $i \in$ [N] (seul x_i est nié dans $\overline{x}$). Pour tout x et y dans $\mathbb{B}^{\mathsf{N}}$, l'ensemble $\Delta(x, y)$, contient les $i \in$ [N] tels que $x_i \neq y_i$. Soit enfin $f : \mathbb{B}^n \to \mathbb{B}^{\mathsf{N}}$. Son $i^{\text{ème}}$ composant est nommé f_i qui est une fonction de $\mathbb{B}^{\mathsf{N}}$ dans $\mathbb{B}$. Pour chaque x dans $\mathbb{B}^{\mathsf{N}}$, l'ensemble $\Delta f(x)$ est défini par $\Delta f(x) = \Delta(x, f(x))$. On peut admettre que $f(x) = \overline{x}^{\Delta f(x)}$.

Exemple. *On considère* $\mathsf{N} = 3$ *et* $f : \mathbb{B}^3 \to \mathbb{B}^3$ *telle que* $f(x) = (f_1(x), f_2(x), f_3(x))$ *avec*

$$\begin{aligned} f_1(x_1, x_2, x_3) &= (\overline{x_1} + \overline{x_2}).x_3, \\ f_2(x_1, x_2, x_3) &= x_1.x_3 \ et \\ f_3(x_1, x_2, x_3) &= x_1 + x_2 + x_3. \end{aligned}$$

La TABLE *1.1 donne l'image de chaque élément* $x \in \mathbb{B}^3$*. Pour* $x = (0, 1, 0)$ *les assertions suivantes se déduisent directement du tableau :*

- $f(x) = (0, 0, 1)$ *;*
- *pour* $I = \{1, 3\}$, $\overline{x}^I = (1, 1, 1)$ *et* $\overline{x} = (1, 0, 1)$ *;*
- $\Delta(x, f(x)) = \{2, 3\}$.

x			$f(x)$		
x_1	x_2	x_3	$f_1(x)$	$f_2(x)$	$f_3(x)$
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	0	0	1
0	1	1	1	0	1
1	0	0	0	0	1
1	0	1	1	1	1
1	1	0	0	0	1
1	1	1	0	1	1

TABLE 1.1 – Images de $(x_1, x_2, x_3) \mapsto ((\overline{x_1} + \overline{x_2}).x_3, x_1.x_3, x_1 + x_2 + x_3)$

1.1.1/ RÉSEAU BOOLÉEN

Soit n un entier naturel représentant le nombre d'éléments étudiés (gènes, protéines,...). Un réseau booléen est défini à partir d'une fonction booléenne :

$$f : \mathbb{B}^{\mathrm{N}} \to \mathbb{B}^{\mathrm{N}}, \qquad x = (x_1, \ldots, x_{\mathrm{N}}) \mapsto f(x) = (f_1(x), \ldots, f_{\mathrm{N}}(x)),$$

et un *schéma itératif* ou encore *mode de mise à jour*. À partir d'une configuration initiale $x^0 \in \mathbb{B}^{\mathrm{N}}$, la suite $(x^t)^{t\in\mathbb{N}}$ des configurations du système est construite selon l'un des schémas suivants :

— **Schéma parallèle synchrone :** basé sur la relation de récurrence $x^{t+1} = f(x^t)$. Tous les x_i, $1 \leq i \leq \mathrm{N}$, sont ainsi mis à jour à chaque itération en utilisant l'état global précédent du système x^t.

— **Schéma unaire :** ce schéma est parfois qualifié de chaotique dans la littérature. Il consiste à modifier la valeur d'un unique élément i, $1 \leq i \leq \mathrm{N}$, à chaque itération. Le choix de l'élément qui est modifié à chaque itération est défini par une suite $S = (s^t)^{t\in\mathbb{N}}$ qui est une séquence d'indices dans [N]. Cette suite est appelée *stratégie unaire*. Il est basé sur la relation définie pour tout $i \in [\mathrm{N}]$ par

$$x_i^{t+1} = \begin{cases} f_i(x^t) \text{ si } i = s^t, \\ x_i^t \text{ sinon.} \end{cases} \tag{1.1}$$

— **Schéma généralisé :** dans ce schéma, ce sont les valeurs d'un ensemble d'éléments de [N] qui sont modifiées à chaque itération. Dans le cas particulier où c'est la valeur d'un singleton $\{k\}$, $1 \leq k \leq \mathrm{N}$, qui est modifiée à chaque itération, on retrouve le mode unaire. Dans le second cas particulier où ce sont les valeurs de tous les éléments de $\{1, \ldots, \mathrm{N}\}$ qui sont modifiées à chaque itération, on retrouve le mode parallèle. Ce mode généralise donc les deux modes précédents. Plus formellement, à la $t^{\text{ème}}$ itération, seuls les éléments de la partie $s^t \in \mathcal{P}([\mathrm{N}])$ sont mis à jour. La suite $S = (s^t)^{t\in\mathbb{N}}$ est une séquence de sous-ensembles de [N] appelée *stratégie généralisée*. Il est basé sur la relation définie pour tout $i \in [\mathrm{N}]$ par

$$x_i^{t+1} = \begin{cases} f_i(x^t) \text{ si } i \in s^t, \\ x_i^t \text{ sinon.} \end{cases} \tag{1.2}$$

La section suivante détaille comment représenter graphiquement les évolutions de tels réseaux.

1.1.2/ GRAPHES DES ITÉRATIONS

Pour un entier naturel N et une fonction $f : B^N \to B^N$, plusieurs évolutions sont possibles en fonction du schéma itératif retenu. Celles-ci sont représentées par un graphe orienté dont les noeuds sont les éléments de $\mathbb{B}^N$ (voir FIGURE 1.1).

— Le *graphe des itérations synchrones* de f, noté GIS(f) est le graphe orienté de $\mathbb{B}^N$ qui contient un arc $x \to y$ si et seulement si $y = f(x)$.
— Le *graphe des itérations unaires* de f, noté GIU(f) est le graphe orienté de $\mathbb{B}^N$ qui contient un arc $x \to y$ si et seulement s'il existe $x \in \Delta f(x)$ tel que $y = \overline{x}^i$.
— Le *graphe des itérations généralisées* de f, noté GIG(f) est le graphe orienté de $\mathbb{B}^N$ qui contient un arc $x \to y$ si et seulement s'il existe un ensemble $I \subseteq \Delta f(x)$ tel que $y = \overline{x}^I$. On peut remarquer que ce graphe contient comme sous-graphe à la fois celui des itérations synchrones et celui des itérations unaires.

Exemple. *On reprend notre exemple illustratif détaillé à la page 3 avec sa table d'images (TABLE 1.1). La FIGURE 1.1 donne les trois graphes d'itérations associés à f.*

1.1.3/ ATTRACTEURS

On dit que le point $x \in \mathbb{B}^N$ est un *point fixe* de f si $x = f(x)$. Les points fixes sont particulièrement intéressants car ils correspondent aux états stables : dans chaque graphe d'itérations, le point x est un point fixe si et seulement si il est son seul successeur.

Soit un graphe d'itérations (synchrones, unaires ou généralisées) de f. Les *attracteurs* de ce graphe sont les plus petits sous-ensembles (au sens de l'inclusion) non vides $A \subseteq \mathbb{B}^N$ tels que pour tout arc $x \to y$, si x est un élément de A, alors y aussi. Un attracteur qui contient au moins deux éléments est dit *cyclique*. On en déduit qu'un attracteur cyclique ne contient pas de point fixe. En d'autres termes, lorsqu'un système entre dans un attracteur cyclique, il ne peut pas atteindre un point fixe.

On a la proposition suivante :

Théorème [1]. *Le point x est un point fixe si et seulement si $\{x\}$ est un attracteur du graphe d'itération (synchrone, unaire, généralisé). En d'autres termes, les attracteurs non cycliques de celui-ci sont les points fixes de f. Ainsi pour chaque $x \in \mathbb{B}^N$, il existe au moins un chemin depuis x qui atteint un attracteur. Ainsi tout graphe d'itérations contient toujours au moins un attracteur.*

Exemple. *Les attracteurs de* GIU(f) *et de* GIG(f) *sont le point fixe* 000 *et l'attracteur cyclique* $\{001, 101, 111, 011\}$. *Les attracteurs de* GIS(f) *sont le point fixe* 000 *et l'attracteur cyclique* $\{011, 101, 111\}$.

1.1.4/ GRAPHE D'INTERACTION

Les interactions entre les composants du système peuvent être mémorisées dans la *matrice jacobienne discrète* f'. Celle-ci est définie comme étant la fonction qui à chaque configuration $x \in \mathbb{B}^N$ associe la matrice de taille $n \times n$ telle que

$$f'(x) = (f'_{ij}(x)), \qquad f'_{ij}(x) = \frac{f_i(\overline{x}^j) - f_i(x)}{\overline{x_j} - x_j} \qquad (i, j \in [n]). \tag{1.3}$$

On note que dans l'équation donnant la valeur de $f'_{ij}(x)$, les termes $f_i(\overline{x}^j)$, $f_i(x)$, $\overline{x}^j_j$ et x_j sont considérés comme des entiers naturels égaux à 0 ou à 1 et que la différence est effectuée dans $\mathbb{Z}$. Lorsqu'on supprime les signes dans la matrice jacobienne discrète, on obtient une matrice notée $B(f)$ aussi de taille $\mathrm{N} \times \mathrm{N}$. Celle-ci mémorise uniquement l'existence d'une dépendance de tel élément vis à vis de tel élément. Elle ne mémorise pas *comment* dépendent les éléments les uns par rapport aux autres. Cette matrice est nommée *matrice d'incidence*.

Théorème [2]. *Si f_i ne dépend pas de x_j, alors pour tout $x \in [\mathrm{N}]$, $f_i(\overline{x}^j)$ est égal à $f_i(x)$, i.e, $f'_{ij}(x) = 0$. Ainsi $B(f)_{ij}$ est nulle. La réciproque est aussi vraie.*

En outre, les interactions peuvent se représenter à l'aide d'un graphe $\Gamma(f)$ orienté et signé défini ainsi : l'ensemble des sommet $[\mathrm{N}]$ et il existe un arc de j à i de signe $s \in \{-1, 1\}$, noté (j, s, i), si $f_{ij}(x) = s$ pour au moins un $x \in \mathbb{B}^{\mathrm{N}}$.

On note que la présence de deux arcs de signes opposés entre deux sommets donnés est possible. Un cycle *positif* (resp. négatif) de G est un cycle *élémentaire* qui contient un nombre pair (resp. impair) d'arcs négatifs. La *longueur* d'un cycle est le nombre d'arcs qu'il contient.

Exemple. *Pour exprimer la jacobienne discrète de la fonction donnée en exemple, pour chaque i, j dans $[3]$ on exprime $f'_{ij}(x) = \frac{f_i(\overline{x}^j) - f_i(x)}{\overline{x}_j - x_j}$ d'après l'équation (1.3). La* FIGURE *(1.2(a)) explicite la matrice jacobienne discrète de cette fonction.*

Le graphe d'interaction de f s'en déduit directement. Il est donné à la FIGURE *(1.2(b)). Il possède un cycle négatif de longueur 1 et un cycle négatif de longueur 3. Concernant les cycles positifs, il en possède un de longueur 1, deux de longueur 2 et un de longueur 3.*

La matrice d'incidence peut se déduire de la matrice d'interaction en supprimant les signes ou bien en constatant que f_1 dépend des trois éléments x_1, x_2 et x_3 et donc que la première ligne de $B(f)$ est égale à $1\ 1\ 1$. De manière similaire, f_2 (resp. f_3) dépend de x_1 et de x_3 (resp. dépend de x_1, x_2 et x_3). Ainsi la seconde ligne (resp. la troisième ligne) de $B(f)$ est $1\ 0\ 1$ (resp. est $1\ 1\ 1$). La FIGURE *(1.2(c)) donne la matrice d'incidence complète.*

Soit P une suite d'arcs de $\Gamma(f)$ de la forme

$$(i_1, s_1, i_2), (i_2, s_2, i_3), \ldots, (i_r, s_r, i_{r+1}).$$

Alors, P est dit un chemin de $\Gamma(f)$ de longueur r et de signe $\Pi_{i=1}^{r} s_i$ et i_{r+1} est dit accessible depuis i_1. P est un *circuit* si $i_{r+1} = i_1$ et si les sommets $i_1, \ldots i_r$ sont deux à deux disjoints. Un sommet i de $\Gamma(f)$ a une *boucle* positive (resp. négative) , si $\Gamma(f)$ a un arc positif (resp. un arc négatif) de i vers lui-même.

1.1.5/ CONDITIONS DE CONVERGENCE

Parmi les itérations unaires caractérisées par leurs stratégies $S = (s^t)^{t \in \mathbb{N}}$ d'éléments appartenant à $[\mathrm{N}]$, sont jugées intéressantes celles qui activent au moins une fois chacun des $i \in [\mathrm{N}]$. Dans le cas contraire, un élément n'est jamais modifié.

Plus formellement, une séquence finie $S = (s^t)^{t \in \mathbb{N}}$ est dite *complète* relativement à $[\mathrm{N}]$ si tout indice de $[\mathrm{N}]$ s'y retrouve au moins une fois.

Parmi toutes les stratégies unaires de $[\mathrm{N}]^{\mathbb{N}}$, on qualifie de :

— *périodiques* celles qui sont constituées par une répétition indéfinie d'une même séquence S complète relativement à [N]. En particulier toute séquence périodique est complète.
— *pseudo-périodiques* celles qui sont constituées par une succession indéfinie de séquences (de longueur éventuellement variable non supposée bornée) complètes. Autrement dit dans chaque stratégie pseudo-périodique, chaque indice de 1 à N revient indéfiniment.

François Robert [Robert, 1995] a énoncé en 1995 le théorème suivant de convergence dans le mode des itérations unaires.

Théorème 3. *Si le graphe $\Gamma(f)$ n'a pas de cycle et si la stratégie unaire est pseudo-périodique, alors tout chemin de* GIU(f) *atteint l'unique point fixe ζ en au plus* N *pseudo-périodes.*

Le qualificatif *pseudo-périodique* peut aisément s'étendre aux stratégies généralisées comme suit. Lorsqu'une stratégie généralisée est constituée d'une succession indéfinie de séquences de parties de [N] dont l'union est [N], cette stratégie est pseudo-périodique. J. Bahi [Bahi, 2000] a démontré le théorème suivant :

Théorème 4. *Si le graphe $\Gamma(f)$ n'a pas de cycle et si la stratégie généralisée est pseudo-périodique alors tout chemin de* GIG(f) *(et donc de* GIU(f)*) finit par atteindre l'unique point fixe ζ.*

1.2/ COMBINAISONS SYNCHRONES ET ASYNCHRONES

Pour être exécuté, le mode des itérations généralisées nécessite que chaque élément connaisse la valeur de chaque autre élément dont il dépend. Pratiquement, cela se réalise en diffusant les valeurs des éléments de proche en proche à tous les composants avant chaque itération. Dans le mode généralisé *asynchrone*, le composant n'attend pas : il met à jour sa valeur avec les dernières valeurs dont il dispose, même si celles-ci ne sont pas à jour. Cette section vise l'étude de ce mode.

Pratiquement, chaque stratégie du mode généralisé peut être mémorisée comme un nombre décimal dont la représentation en binaire donne la liste des éléments modifiés. Par exemple, pour un système à 5 éléments la stratégie définie par

$$s^t = 24 \text{ si } t \text{ est pair et } s^t = 15 \text{ sinon} \tag{1.4}$$

active successivement les deux premiers éléments (24 est 11000) et les quatre derniers éléments (15 est 01111). On dit que la stratégie est *pseudo-periodique* si tous les éléments sont activés infiniment souvent. Dans le mode asynchrone, a chaque itération t, chaque composant peut mettre à jour son état en fonction des dernières valeurs qu'il connaît des autre composants. Obtenir ou non les valeurs les plus à jours dépend du temps de calcul et du temps d'acheminement de celles-ci. On parle de latence, de délai.

Formalisons le mode les itérations asynchrone. Soit $x^0 = (x_1^0, \ldots, x_n^0)$ une configuration initiale. Soit $(D^t)^{t\in\mathbb{N}}$ la suite de matrice de taille $n \times n$ dont chaque élément D_{ij}^t représente la date (inférieure ou égale à t) à laquelle la valeur x_j produite par le composant j devient disponible au composant i. On considère que le délai entre l'émission par j et la réception par i, défini par $\delta_{ij}^t = t - D_{ij}^t$ est borné par une constante δ_0 pour tous les i, j. Le *mode*

des itérations généralisées asynchrone est défini pour chaque $i \in \{1, \ldots, n\}$ et chaque $t = 0, 1, 2, ...$ par :

$$x_i^{t+1} = \begin{cases} f_i(x_1^{D^t_{i1}}, \ldots, x_n^{D^t_{in}}) \text{ si } bin(s^t)[i] = 1 \\ x_i^t \text{ sinon} \end{cases} \tag{1.5}$$

où *bin* convertit un entier en un nombre binaire. Les itérations de f sont *convergentes* modulo une configuration initiale x^0, une stratégie s et une matrice de dates $(D^t)^{t\in\mathbb{N}}$, si la fonction atteint un point fixe. Cela revient à vérifier la propriété suivante :

$$\exists t_0 \,.\, (\forall t \,.\, t \geq t_0 \Rightarrow x^t = x^{t_0}). \tag{1.6}$$

Sinon les itérations sont dites *divergentes*. De plus, si $(x^{(t)})^{t\in\mathbb{N}}$ défini selon l'équation (1.5) satisfait (1.6) pour tous les $x^{(0)} \in E$, pour toutes les stratégies pseudo périodiques s et pour toutes les matrices de dates, $(D^{(t)})^{t\in\mathbb{N}}$, alors les itérations de f sont *universellement convergentes*.

Exemple. *On considère cinq éléments à valeurs dans* $\mathbb{B}$*. Une configuration dans* $\mathbb{B}^5$ *est représentée par un entier entre 0 et 31. La* FIGURE *1.3 donne la fonction définissant la dynamique du système et son graphe d'interaction. On note que le graphe d'interaction contient cinq cycles. Les résultats connus [Bahi, 2000] de conditions suffisantes établissant la convergence du système pour les itérations généralisées sont basés sur l'absence de cycles. Ils ne peuvent donc pas être appliqués ici.*

Dans ce qui suit, les configurations sont représentées à l'aide d'entiers plutôt que nombres binaires. Le graphe des itérations synchrones est donné en FIGURE 1.4(a). Depuis n'importe quelle configuration, on constate qu'il converge vers le point fixe correspondant à l'entier 19. Un extrait du graphe des itérations unaires est donné à la FIGURE 1.4(b). Les libellés des arcs correspondent aux éléments activés. Les itérations unaires ne convergent pas pour la stratégie pseudo périodique donnée à l'équation (1.4) : le système peut infiniment boucler entre 11 et 3, entre 15 et 7.

Comme les itérations unaires ne convergent pas pour certaines stratégies, les itérations asynchrones basées sur les même stratégies peuvent ne pas converger aussi. Cependant, même si l'on considère que tous les composants sont activés à chaque itération, c'est à dire si s^t est constamment égal à $2^n - 1$, le délai peut introduire de la divergence. On considère par exemple la matrice D^t dont chaque élément vaut t sauf D^t_{12} qui vaut $t-1$ si t est impair. On a ainsi $x^{t+1} = f(x^t)$ si t est pair et

$$x^{t+1} = \left(f_1(x_1^t, x_2^{t-1}, x_3^t, x_4^t, x_5^t), f_2(x^t), \ldots, f_5(x^t)\right).$$

sinon. En démarrant de $x^0 = 00011$, le système atteint $x^1 = 01011$ et boucle entre ces deux configurations. Pour une même stratégies, les itérations asynhrones divergent alors que les synchrones convergent. Les sections suivantes de ce chapitre montrent comment résoudre ce problème.

1.2.1/ ITÉRATIONS MIXES

Introduit dans [Abbas et al., 2005] le mode d'*itérations mixes* combine synchronisme et asynchronisme. Intuitivement, les nœuds qui pourraient introduire des cycles dans les

itérations asynchrones sont regroupés. Les noeuds à l'intérieur de chaque groupe seront itérés de manière synchrone. Les itérations asynchrones sont conservées entre les groupes.

Définition 1 (Relation de Synchronisation)**.** *Soit une fonction f et $\Gamma(f)$ son graphe d'interaction. La* relation de synchronisation $\mathcal{R}$ *est définie sur l'ensemble des nœuds par : $i\mathcal{R}j$ si i et j appartiennent à la même composante fortement connexe (CFC) dans $\Gamma(F)$.*

On peut facilement démontrer que la relation de synchronisation est une relation d'équivalence sur l'ensemble des éléments. On introduit quelques notations : par la suite $\langle i \rangle$ représente la classe d'équivalence de i et $\mathcal{K}$ représente l'ensemble de toutes les classe, *i.e.*, $\mathcal{K} = \{1, \ldots, n\}/\mathcal{R}$. On peut définir les itérations mixes.

Définition 2 (Itérations mixes)**.** *Les itérations mixes d'un système discret suit l'équation (1.5) où de plus $bin(s^t)[i] = bin(s^t)[j]$ et $D^t_{ij} = D^t_{ji} = t$ si $i\mathcal{R}j$.*

Dans ce contexte, il n'y a plus de délai entre deux noeuds de la même CFC et leurs mises à jour sont synchronisées. Cependant, pour p_0 et p_1 dans la même classe $\langle p \rangle$, et q dans une autre classe $\langle q \rangle$, ce mode opératoire autorise des délais différents entre p_0 et q et entre p_1 et q. Ainsi p_1 et p_2 sont distinguables même s'ils appartiennent à la même classe. Pour gommer cette distinction, on définit le mode suivant :

Définition 3 (Itérations mixes avec delais uniformes)**.** *Le mode mixe a des* délais uniformes*si pour chaque $t = 0, 1, \ldots$ et pour chaque paire de classes $(\langle p \rangle, \langle q \rangle)$, il existe une constante d^t_{pq} telle que la propriété suivante est établie :*

$$\bigwedge_{p_k \in \langle p \rangle, q_k \in \langle q \rangle} D^t_{p_k q_k} = d^t_{pq}$$

On a alors le théorème suivant.

Théorème 5. *Soit une fonction f possédant un unique point fixe x^* et une stratégie pseudo périodique s. Si les itérations synchrones convergent vers x^* pour cette stratégie, alors les itérations mixes à délai uniforme convergent aussi vers x^* pour cette stratégie.*

La preuve de ce théorème est donnée en section A.1.

1.2.2/ DURÉES DE CONVERGENCE

Cette section donne des bornes supérieures et inférieures des durées globales de convergence pour les modes synchrones, mixes et asynchrones. Pour simplifier le discours, on considère que les itérations convergent en I étapes dans le mode synchrone et que le graphe d'interaction ne contient qu'une seule composante connexe. Les durées de convergence prennent en compte les temps de calcul et les temps de communication, ce depuis l'initialisation et jusqu'à la stabilisation.

Pour simplifier l'évaluation, nous considérons que le temps de calcul d'une itération sur un composant ainsi que celui de communication entre deux composants est constant. Ceci implique en particulier que, dans le mode asynchrone, ces derniers sont bornés. En d'autres mots, il existe un entier δ_0 tel que $0 \leq t - D^t_{ij} \leq \delta_0$ est établi pour tout couple de nœuds (i, j). Les notations utilisées sont les suivantes :

— **Taille pour coder l'information :** elle représente le nombre de bits nécessaires pour représenter l'état courant du composant i et est notée cs_i ;
— **Temps de calcul :** le composant i a besoins de cp_i unités de temps pour faire une mise à jour locale de son état ;
— **Temps de communication :** on utilise le modèle classique de communication $\beta + L\tau$ où L est le nombre de bits transférés. On définit β_{ij} et τ_{ij} comme la latence et la bande passante du lien entre i et j.

1.2.3/ LE MODE SYNCHRONE

Dans le cas synchrone, la convergence la plus rapide est obtenue lorsque le point fixe x^* est accessible en un seul pas depuis toute configuration. Le temps global de convergence est donc minoré par $T_{min}(Sync) = \max_i cp_i$ Dans le cas général, si B est la matrice d'adjacence représentant le graphe d'interaction, le temps global de convergence est

$$T(Sync) = I \times (\max_i cp_i + \max_{i,j}(B_{ji} \times (\beta_{ij} + cs_i \times \tau_{ij}))) \tag{1.7}$$

Exemple. *Intuitivement la convergence se propage selon les dépendances internes au système : un nœuds se stabilise lorsque ceux dont il dépend sont eux aussi stables. Cette stabilisation progressive est illustrée à la* FIGURE *1.5 qui représente des exécutions synchrones dans le cas d'une initialisation avec la valeur (00100). Dans cette figure et les suivantes, les blocs doublement hachurés indiquent la stabilisation du composant.*

On peut constater que la première classe $\langle 1 \rangle$ *se stabilise en deux itérations, la seconde classe* $\langle 3 \rangle$ *atteint sa valeur finale l'itération suivante tandis que la dernière classe,* $\langle 4 \rangle$*, converge en deux itérations.*

$$I = I_{\langle 1 \rangle} + I_{\langle 3 \rangle} + I_{\langle 4 \rangle} = 2 + 1 + 2 = 5 \tag{1.8}$$

1.2.4/ LE MODE MIXE

On considère $|\mathcal{K}|$ classes de composants synchronisés. (comme donné en équation (1.8)). Soit I_k le nombre d'itérations suffisants pour que la classe $\langle k \rangle \in \mathcal{K}$ se stabilise sachant toutes ses dépendances ont déjà convergé. Ainsi I vaut $\sum_{\langle k \rangle \in \mathcal{K}} I_k$. La borne inférieure pour la durée de convergence des itérations asynchrones est

$$T(Mixed) \geq \sum_{k \in \mathcal{K}} I_k(\max_{l \in k} cp_l) \tag{1.9}$$

qui apparaît lorsque tous les délais de communication sont consommés par des durées de calcul.

Concernant le majorant, celui-ci correspond au cas où les durées de communications entre les classes désynchronisées ne sont pas consommées par des calculs ou lorsque chaque classe nécessite la stabilisation de tous ses ascendants pour converger. On a dans ce cas :

$$T(Mixed) \leq \sum_{k \in \mathcal{K}} \left(I_k \times (\max_{l \in k} cp_l) + \max_{l \in k, e \in k', k \preceq k'} B_{el} \times (\beta_{le} + cs_l \tau_{le}) \right) \tag{1.10}$$

Exemple. *Une exécution du mode mixe est donnée à la* FIGURE *1.6. On peut constater que le temps d'exécution peut être plus petit que pour le mode synchrone.*

1.2.5/ LE MODE GÉNÉRALISÉ ASYNCHRONE

En terme de durée de convergence, ce mode peut être vu comme un cas particulier du mode mixe où toutes les classes sont des singletons. La borne minimale peut donc s'exprimer comme :

$$T(\mathit{Async}) \geq \max_{i=1}^{n} I_i \times \mathit{cp}_i \tag{1.11}$$

où I_i est le nombre d'itérations suffisant pour que le nœud i converge et qui est atteint si tous les nœuds sont indépendants les uns des autres. Cette borne est arbitrairement faible et n'est pas atteinte dès qu'une dépendance existe. La borne supérieure quant à elle est donnée par :

$$T(\mathit{Async}) \leq \sum_{i=1}^{n} \left(I_i \times \mathit{cp}_i + \max_{1 \leq k \leq n} B_{ki}(\beta_{ik} + \mathit{cs}_i \tau_{ik}) \right) \tag{1.12}$$

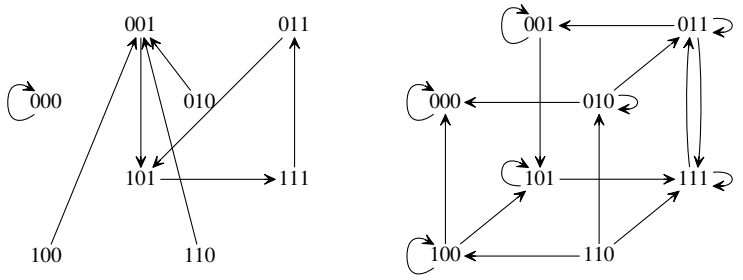
et apparaît lorsque chaque élément dépend des autres et que les calculs ne recouvrent nullement les communications.

Exemple. *La* FIGURE *1.7 présente un exemple d'exécution du mode généralisé asynchrone. Certaines communications issues de l'élément* 4 *n'ont pas été représentées pour des raisons de clarté. On constate que le temps global de convergence est plus petit que celui des deux autres modes.*

1.3/ CONCLUSION

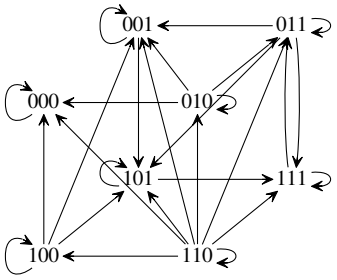
Conclusion à refaire

Introduire de l'asynchronisme peut permettre de réduire le temps d'exécution global, mais peut aussi introduire de la divergence. Dans ce chapitre, nous avons exposé comment construire un mode combinant les avantage du synchronisme en terme de convergence avec les avantages de l'asynchronisme en terme de vitesse de convergence.



(a) GIS(f)

(b) GIU(f)

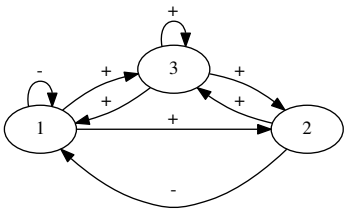


(c) GIG(f)

FIGURE 1.1 – Graphes des itérations de la fonction $f : \mathbb{B}^3 \to \mathbb{B}^3$ telle que $(x_1, x_2, x_3) \mapsto ((\overline{x_1} + \overline{x_2}).x_3, x_1.x_3, x_1 + x_2 + x_3)$. On remarque le cycle $((101, 111), (111, 011), (011, 101))$ à la FIGURE (1.1(a)).

$$\begin{pmatrix} \frac{((x_1+\overline{x_2}).x_3)-((\overline{x_1}+\overline{x_2}).x_3)}{\overline{x_1}-x_1} & \frac{((\overline{x_1}+x_2).x_3)-((\overline{x_1}+\overline{x_2}).x_3)}{\overline{x_2}-x_2} & \frac{((\overline{x_1}+\overline{x_2}).\overline{x_3})-((\overline{x_1}+\overline{x_2}).x_3)}{\overline{x_3}-x_3} \\ \frac{\overline{x_1}.x_3-x_1.x_3}{\overline{x_1}-x_1} & 0 & \frac{x_1.\overline{x_3}-x_1.x_3}{\overline{x_3}-x_3} \\ \frac{(\overline{x_1}+x_2+x_3)-(x_1+x_2+x_3)}{\overline{x_1}-x_1} & \frac{(x_1+\overline{x_2}+x_3)-(x_1+x_2+x_3)}{\overline{x_2}-x_2} & \frac{(x_1+x_2+\overline{x_3})-(x_1+x_2+x_3)}{\overline{x_3}-x_3} \end{pmatrix}$$

(a) Matrice jacobienne



(b) Graphe d'interaction

$$B(f) = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

(c) Matrice d'incidence

FIGURE 1.2 – Représentations des dépendances entre les éléments de la fonction $f : \mathbb{B}^3 \to \mathbb{B}^3$ telle que $(x_1, x_2, x_3) \mapsto ((\overline{x_1} + \overline{x_2}).x_3, x_1.x_3, x_1 + x_2 + x_3)$

$$f(x) = \begin{cases} f_1(x_1, x_2, x_3, x_4, x_5) &= x_1.\overline{x_2} + \overline{x_1}.x_2 \\ f_2(x_1, x_2, x_3, x_4, x_5) &= \overline{x_1 + x_2} \\ f_3(x_1, x_2, x_3, x_4, x_5) &= x_3.\overline{x_1} \\ f_4(x_1, x_2, x_3, x_4, x_5) &= x_5 \\ f_5(x_1, x_2, x_3, x_4, x_5) &= \overline{x_3} + x_4 \end{cases}$$

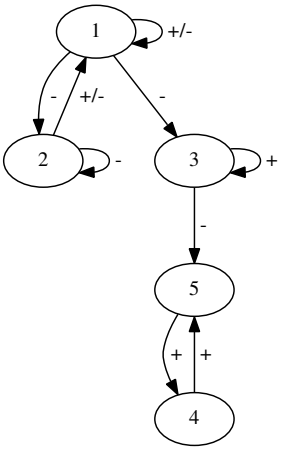
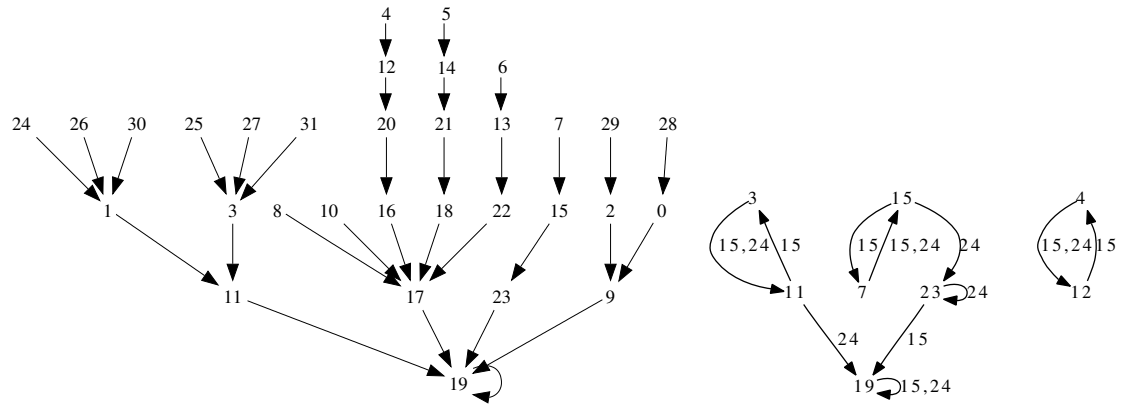


FIGURE 1.3 – Définition de $f : \mathbb{B}^5 \to \mathbb{B}^5$ et son graphe d'interaction



(a) Itérations synchrones de f. (b) Extrait des itérations unaires.

FIGURE 1.4 – Graphes des itérations de f définie à la figure 1.3

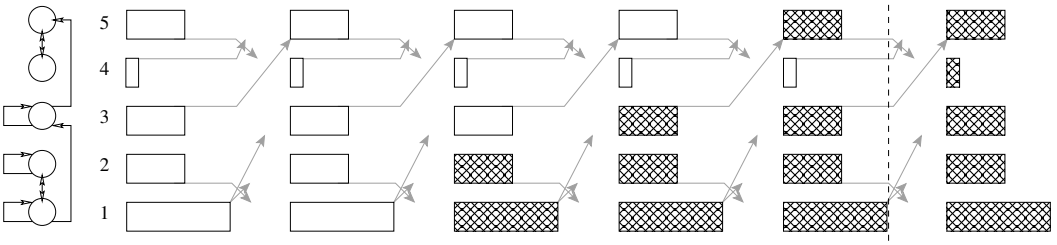


FIGURE 1.5 – Itérations synchrones

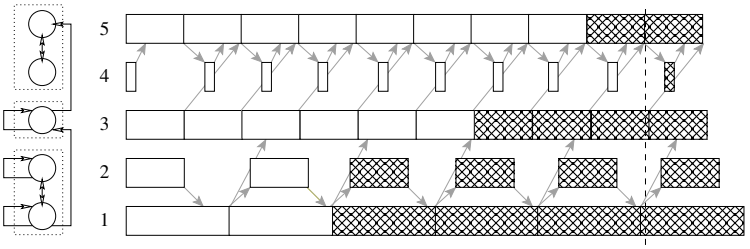


FIGURE 1.6 – Itérations mixes avec $\langle 1\rangle = \{1,2\}$, $\langle 3\rangle = \{3\}$, $\langle 4\rangle = \{4,5\}$.

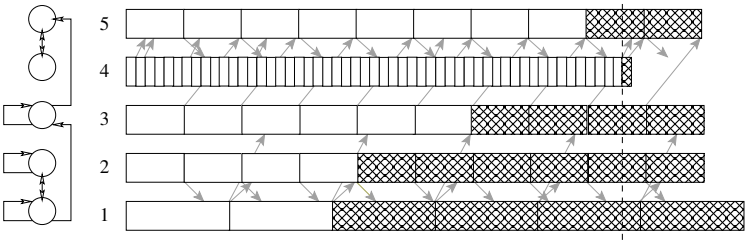


FIGURE 1.7 – Itérations asynchrones

2

PREUVE AUTOMATIQUE DE CONVERGENCE

L'étude de convergence de systèmes dynamiques discrets est simple à vérifier pratiquement pour le mode synchrone. Lorsqu'on introduit des stratégies pseudo périodiques pour les modes unaires et généralisées, le problème se complexifie. C'est pire encore lorsqu'on traite des itérations asynchrones et mixes prenant de plus en compte les délais.

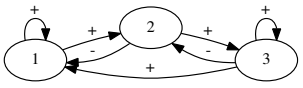
Des méthodes de simulation basées sur des stratégies et des délais générés aléatoirement ont déjà été présentées [Bahi and Michel, 1999, Bahi and Contassot-Vivier, 2002]. Cependant, comme ces implantations ne sont pas exhaustives, elles ne donnent un résultat formel que lorsqu'elles fournissent un contre-exemple. Lorsqu'elles exhibent une convergence, cela ne permet que donner une intuition de convergence, pas une preuve. Autant que nous sachions, aucune démarche de preuve formelle automatique de convergence n'a jamais été établie. Dans le travail théorique [Chandrasekaran, 2006], Chandrasekaran a montré que les itérations asynchrones sont convergentes si et seulement si on peut construire une fonction de Lyaponov décroissante, mais il ne donne pas de méthode automatique pour construire cette fonction.

Un outil qui construirait automatiquement toutes les transitons serait le bienvenu. Pour peu qu'on établisse la preuve de correction et de complétude de la démarche, la convergence du réseau discret ne repose alors que sur le verdict donné par l'outil. Cependant, même pour des réseaux discrets à peu d'éléments, le nombre de configurations induites explose rapidement. Les *Model-Checkers* [Holzmann, 2003, Cimatti et al., 2002, Beyer et al., 2007, Fredlund and Svensson, 2007, Robby et al., 2003] sont des classes d'outils qui addressent le problème de vérifier automatiquement qu'un modèle vérifie une propriété donnée. Pour traiter le problème d'explosion combinatoire, ces outils appliquent des méthodes d'ordre partiel, d'abstraction, de quotientage selon une relation d'équivalence.

Ce chapitre montre comment nous simulons des réseaux discrets selon toutes les sortes d'itérations pour établir formellement leur convergence (ou pas). Nous débutons par un exemple et faisons quelques rappels sur le langage PROMELA qui est le langage du model-checker SPIN [Holzmann, 2003] (Section 2.1). Nous présentons ensuite la démarche de traduction de réseaux discrets dans PROMELA (Section 2.2). Les théorèmes de correction et de complétude de la démarche sont ensuite donnés à la (Section 2.3). Des données pratiques comme la complexité et des synthèses d'expérimentation sont ensuite fournies (Section 2.4).

$$F(x) = \begin{cases} f_1(x_1, x_2, x_3) &= x_1.\overline{x_2} + x_3 \\ f_2(x_1, x_2, x_3) &= x_1 + \overline{x_3} \\ f_3(x_1, x_2, x_3) &= x_2.x_3 \end{cases}$$

(a) Fonction à itérer



(b) Graphe d'intéraction

FIGURE 2.1 – Exemple pour SDD ≈ SPIN.

Exemple. *On considère un exemple à trois éléments dans* $\mathbb{B}$*. Chaque configuration est ainsi un élement de* $\mathbb{B}^3$*, i.e., un nombre entre 0 et 7. La* FIGURE *2.1(a) précise la fonction* f *considérée et la* FIGURE *2.1(b) donne son graphe d'intéraction.*

On peut facilement vérifier que toutes les itérations synchrones initialisées avec $x^0 \neq 7$ *soit* (111) *convergent vers* 2 *soit* (010)*; celles initialisées avec* $x^0 = 7$ *restent en 7. Pour les mode unaires ou généralisés avec une stratégie pseudo périodique, on a des comportements qui dépendent de la configuration initiale :*

- *initialisée avec 7, les itérations restent en 7 ;*
- *initialisée avec 0, 2, 4 ou 6 les itérations convergent vers 2 ;*
- *initialisées avec 1, 3 ou 5, les itérations convergent vers un des deux points fixes 2 ou 7.*

2.1/ RAPPELS SUR LE LANGAGE PROMELA

Cette section rappelle les éléments fondamentaux du langage PROMELA (Process Meta Language). On peut trouver davantage de détails dans [Holzmann, 2003, Weise, 1997].

```
#define N 3
#define d_0 5

bool X [N]; bool Xp [N]; int mods [N];
typedef vals{bool v [N]};
vals Xd [N];

typedef a_send{chan sent[N]=[d_0] of {bool}};
a_send channels [N];

chan unlock_elements_update=[1] of {bool};
chan sync_mutex=[1] of {bool};
```

FIGURE 2.2 – Declaration des types de la traduction.

Comme en C, on peut déclarer des tableaux à une dimension ou des nouveaux types de données (introduites par le mot clef `typedef`).

Exemple. *Le programme donné à la* FIGURE *2.2 correspond à des déclarations de variables qui servent dans l'exemple de ce chapitre. Il définit :*

- *les constantes* `N` *et* `d_0` *qui précisent respectivement le nombre* n *d'éléments et le délais maximum* δ_0 *;*
- *les deux tableaux (*`X` *et* `Xp`*) de* `N` *variables booléennes ; les cellules* `X[i]` *et* `Xp[i]` *sont associées à la variables* x_{i+1} *d'un système dynamique discret ; elles mémorisent les valeurs de* X_{i+1} *respectivement avant et après sa mise à jour ; il suffit ainsi de comparer* `X` *et* `Xp` *pour constater si* x *à changé ou pas ;*

— *le tableau* `mods` *contient les éléments qui doivent être modifiés lors de l'itération en cours ; cela correspond naturellement à l'ensemble des éléments* s^t *;*
— *le type de données structurées* `vals` *et le tableau de tableaux* `Xd[`i`].v[`j`]` *qui vise à mémoriser* $x_{j+1}^{D_{i+1j+1}^{t-1}}$ *pour l'itération au temps* t*.*

Déclarée avec le mot clef `chan`*, une donnée de type* `channel` *permet le transfert de messages entre processus dans un ordre FIFO. Dans l'exemple précédent, on déclare successivement :*

— *un canal* `sent` *qui vise à mémoriser* `d_0` *messages de type* `bool` *; le tableau nommé* `channels` *de* `N*N` *éléments de type* `a_send` *est utilisé pour mémoriser les valeurs intermédiaires* x_j *; Il permet donc de temporiser leur emploi par d'autres elements* i*.*
— *les deux canaux* `unlock_elements_update` *et* `sync_mutex` *contenant chacun un message booléen et utilisé ensuite comme des sémaphores.*

Le langage PROMELA exploite la notion de *process* pour modéliser la concurrence au sein de systèmes. Un process est instancié soit immédiatement (lorsque sa déclaration est préfixée par le mot-clef `active`) ou bien au moment de l'exécution de l'instruction `run`. Parmi tous les process, `init` est le process initial qui permet d'initialiser les variables, lancer d'autres process...

Les instructions d'affectation sont interprétées usuellement. Les canaux sont concernés par des instructions particulières d'envoi et de réception de messages. Pour un canal `ch`, ces instructions sont respectivement notées `ch ! m` et `ch ? m`. L'instruction de réception consomme la valeur en tête du canal `ch` et l'affecte à la variable `m` (pour peu que `ch` soit initialisé et non vide). De manière similaire, l'instruction d'envoi ajoute la valeur de `m` à la queue du canal `ch` (pour peu que celui-ci soit initialisé et non rempli). Dans les cas problématiques, canal non initialisé et vide pour une réception ou bien rempli pour un envoi, le processus est bloqué jusqu'à ce que les conditions soient remplies.

La structures de contrôle `if` (resp. `do`) définit un choix non déterministe (resp. une boucle non déterministe). Que ce soit pour la conditionnelle ou la boucle, si plus d'une des conditions est établie, l'ensemble des instructions correspondantes sera choisi aléatoirement puis exécuté.

Dans le process `init` détaillé à la FIGURE 2.3, une boucle de taille N initialise aléatoirement la variable globale de type tableau `Xp`. Ceci permet par la suite de vérifier si les itérations sont convergentes pour n'importe quelle configuration initiale $x^{(0)}$.

Pour chaque élément i, si les itérations sont asynchrones

— on stocke d'abord la valeur de `Xp[i]` dans chaque `Xd[j].v[i]` puisque la matrice s^0 est égale à (0),
— puis, la valeur de i (représentée par `Xp[i]`) devrait être transmise à j s'il y a un arc de i à j dans le graphe d'incidence. Dans ce cas, c'est la fonction `hasnext` (détaillée à la FIGURE 2.5) qui mémorise ce graphe en fixant à `true` la variable `is_succ`, naturellement et à `false` dans le cas contraire. Cela permet d'envoyer la valeur de i dans le canal au travers de `channels[i].sent[j]`.

```
init{
 int i=0; int j=0; bool is_succ=0;
 do
 ::i==N->break;
 ::i< N->{
   if
   ::Xp[i]=0;
   ::Xp[i]=1;
   fi;
   j=0;
   do
   ::j==N -> break;
   ::j< N -> Xd[j].v[i]=Xp[i]; j++;
   od;
   j=0;
   do
   ::j==N -> break;
   ::j< N -> {
     hasnext(i,j);
     if
     ::(i!=j && is_succ==1) ->
     channels[i].sent[j] ! Xp[i];
     ::(i==j || is_succ==0) -> skip;
     fi;
     j++;}
   od;
   i++;}
 od;
 sync_mutex ! 1;
}
```

FIGURE 2.3 – Process init.

```
active proctype scheduler(){
 do
 ::sync_mutex ? 1 -> {
   int i=0; int j=0;
   do
   :: i==N -> break;
   :: i< N -> {
     if
     ::skip;
     ::mods[j]=i; j++;
     fi;
     i++;}
   od;
   ar_len=i;
   unlock_elements_update ! 1;
   }
 od
}
```

FIGURE 2.4 – Process scheduler pour la stratégie pseudo pérodique.

```
inline hasnext(i,j){
  if
    :: i==0 && j ==0 -> is_succ = 1
    :: i==0 && j ==1 -> is_succ = 1
    :: i==0 && j ==2 -> is_succ = 0
    :: i==1 && j ==0 -> is_succ = 1
    :: i==1 && j ==1 -> is_succ = 0
    :: i==1 && j ==2 -> is_succ = 1
    :: i==2 && j ==0 -> is_succ = 1
    :: i==2 && j ==1 -> is_succ = 1
    :: i==2 && j ==2 -> is_succ = 1
  fi
}
```

FIGURE 2.5 – Codage du graphe d'intéraction de f.

2.2/ DU SYSTÈME BOOLÉEN AU MODÈLE PROMELA

Les éléments principaux des itérations asynchrones rappelées à l'équation (1.5) sont la stratégie, la fonctions et la gestion des délais. Dans cette section, nous présentons successivement comment chacune de ces notions est traduite vers un modèle PROMELA.

2.2.1/ LA STRATÉGIE

Regardons comment une stratégie pseudo périodique peut être représentée en PROMELA. Intuitivement, un process `scheduler` (comme représenté à la FIGURE 2.4) est iterrativement appelé pour construire chaque s^t représentant les éléments possiblement mis à jour à l'itération t.

Basiquement, le process est une boucle qui est débloquée lorsque la valeur du sémaphore `sync_mutex` est 1. Dans ce cas, les éléments à modifier sont choisis aléatoirement (grâce à n choix successifs) et sont mémorisés dans le tableau `mods`, dont la taille est `ar_len`. Dans la séquence d'exécution, le choix d'un élément mis à jour est directement suivi par des mises à jour : ceci est réalisé grâce à la modification de la valeur du sémaphore `unlock_elements_updates`.

2.2.2/ ITÉRER LA FONCTION f

La mise à jour de l'ensemble $s^t = \{s_1, \ldots, s_m\}$ des éléments qui constituent la stratégie $(s^t)^{t\in\mathbb{N}}$ est implantée à l'aide du process `update_elems` fourni à la FIGURE 2.7. Ce processus actif attend jusqu'à ce qu'il soit débloqué par le process `scheduler` à l'aide du sémaphore `unlock_elements_update`. L'implantation se déroule en cinq étapes :

1. elle commence en mettant à jour la variable `X` avec les valeurs de `Xp` dans la fonction `update_X`, FIGURE 2.6

```
inline update_X(){
  int countu;
  countu = 0;
  do
    :: countu == N -> break ;
    :: countu != N ->
       X[countu] = Xp[countu];
       countu ++ ;
  od
}
```

FIGURE 2.6 – Sauvegarde de l'état courant

```
active proctype update_elems(){
 do
 ::unlock_elements_update ? 1 ->
   {
    atomic{
     bool is_succ=0;
     update_X();
     fetch_values();
     int count = 0;
     int j = 0;
     do
     ::count == ar_len -> break;
     ::count < ar_len ->
       j = mods[count];
       F(j);
       count++;
     od;
     diffuse_values(Xp);
     sync_mutex ! 1
    }
   }
 od
}
```

FIGURE 2.7 – Mise à jour des éléments.

```
inline F(){
 if
 ::j==0 -> Xp[0] =
      (Xs[j].v[0] & !Xs[j].v[1])
     |(Xs[j].v[2])
 ::j==1 -> Xp[1] =  Xs[j].v[0]
                  | !Xs[j].v[2]
 ::j==2 -> Xp[2] =  Xs[j].v[1]
                  &  Xs[j].v[2]
 fi
}
```

FIGURE 2.8 – Application de la fonction f.

2. elle mémorise dans `Xd` la valeurs disponible pour chaque élément grâce à la fonction `fetch_values` ; cette fonction est détaillée dans la section suivante ;
3. une boucle met à jour iterrativement la valeur de j (grâce à l'appel de fonction `f(j)`) pour peu que celui-ci doive être modifié, *i.e.*, pour peu qu'il soit renseigné dans `mods[count]` ; le code source de `F` est donné en FIGURE 2.8 et est une traduction directe de l'application f ;
4. les nouvelles valeurs des éléments `Xp` sont symboliquement envoyés aux autres éléments qui en dépendent grâce à la fonction `diffuse_values(Xp)` ; cette dernière fonction est aussi détaillée dans la section suivante ;
5. finalement, le process informe le scheduler de la fin de la tâche (au travers du sémaphore `sync_mutex`).

2.2.3/ GESTION DES DÉLAIS

Cette section montre comment les délais inhérents au mode asynchrone sont traduits dans le modèle PROMELA grâce à deux fonctions `fetch_values` et `diffuse_values`. Celles-ci sont données en FIGURE 2.9 et 2.10, qui récupèrent et diffusent respectivement les valeurs des elements.

La première fonction met à jour le tableau `Xd` requis pour les éléments qui doivent être modifiés. Pour chaque élément dans `mods`, identifié par la variable j, la fonction récupère les valeurs des autres éléments (dont le libellé est i) dont j dépend. Il y a deux cas.

- puisque i connaît sa dernière valeur (*i.e.*, D^t_{ii} est toujours t) `Xd[i].v[i]` est donc `Xp[i]` ;
- sinon, il y a deux sous cas qui peuvent peuvent potentiellement modifier la valeur que j a de i (et qui peuvent être choisies de manière aléatoire) :
  - depuis la perspective de j la valeur de i peut ne pas avoir changé (c'est l'instruction `skip`) ou n'est pas utile ; ce dernier cas apparaît lorsqu'il n'y a pas d'arc de i à j dans le graphe d'incidence, *i.e.*, lorsque la valeur de `is_succ` qui est calculée par `hasnext(i,j)` est 0 ; dans ce cas, la valeur de `Xd[j].v[i]` n'est pas modifiée ;

```
inline fetch_values(){
 int countv = 0;
 do
 :: countv == ar_len -> break ;
 :: countv < ar_len ->
    j = mods[countv];
    i = 0;
    do
    :: (i == N) -> break;
    :: (i < N && i == j) -> {
        Xd[j].v[i] = Xp[i] ;
        i++ }
    :: (i < N && i != j) -> {
        hasnext(i,j);
        if
        :: skip
        :: is_succ==1 &&
    nempty(channels[i].sent[j]) ->
             channels[i].sent[j] ?
                        Xd[j].v[i];
        fi;
        i++ }
    od;
    countv++
 od
}
```

FIGURE 2.9 – Récupérer les valeurs des elements

```
inline diffuse_values(values){
 int countb=0;
 do
 :: countb == ar_len -> break ;
 :: countb < ar_len ->
    j = mods[countb];
    i = 0 ;
    do
    :: (i == N) -> break;
    :: (i < N && i == j) -> i++;
    :: (i < N && i != j) -> {
        hasnext(j,i);
        if
        :: skip
        :: is_succ==1 &&
     nfull(channels[j].sent[i]) ->
            channels[j].sent[i] !
                        values[j];
        fi;
        i++ }
    od;
    countb++
 od
}
```

FIGURE 2.10 – Diffuser les valeurs des elements

— sinon, on affecte à `Xd[j].v[i]` la valeur mémorisée dans le canal `channels[i].sent[j]` (pour peu que celui-ci ne soit pas vide).

Les valeurs des éléments sont ajoutées dans ce canal au travers de la fonction `diffuse_values`. L'objectif de cette fonction est de stocker les valeurs de x (représenté dans le modèle par `Xp`) dans le canal `channels`. Il permet au modèle-checker SPIN d'exécuter le modèle PROMELA comme s'il pouvait y avoir des délais entre processus Il y a deux cas différents pour la valeur de X_j :

— soit elle est « perdue », « oubliée » pour permettre à i de ne pas tenir compte d'une des valeurs de j ; ce cas a lieu soit lors de l'instruction `skip` ou lorsqu'il n'y a pas d'arc de j à i dans le graphe d'incidence ;

— soit elle est mémorisée dans le canal `channels[j].sent[i]` (pour peu que celui-ci ne soit pas plein).

L'introduction de l'indéterminisme à la fois dans les fonctions `fetch_values` et `diffuse_values` est nécessaire dans notre contexte. Si celui-ci n'était présent que dans la fonction `fetch_values`, nous ne pourrions pas par exemple récupérer la valeur $x_i^{(t)}$ sans considérer la valeur $x_i^{(t-1)}$. De manière duale, si le non déterminisme était uniquement utilisé dans la fonction `diffuse_values`, alors chaque fois qu'une valeur serait mise dans le canal, elle serait immédiatement consommée, ce qui est contradictoire avec la notion de délai.

2.2.4/ PROPRIÉTÉ DE CONVERGENCE UNIVERSELLE

Il reste à formaliser dans le model checker SPIN le fait que les itérations d'un système dynamique à n éléments est universellement convergent.

Rappelons tout d'abord que les variables `X` et `Xp` contiennent respectivement la valeur de x avant et après la mise à jour. Ainsi, si l'on effectue une initialisation non déterministe de `Xp` et si l'on applique une stratégie pseudo périodique, il est nécessaire et suffisant de prouver la formule temporelle linéaire (LTL) suivante :

$$\diamond\,(\square \mathtt{Xp} = \mathtt{X}) \tag{2.1}$$

où les opérateur $\diamond$ et $\Box$ ont la sémantique usuelle, à savoir respectivement *éventuellement* et *toujours* dans les chemins suivants. On note que cette propriété, si elle est établie, garantit la stabilisation du système. Cependant elle ne donne aucune métrique quant à la manière dont celle-ci est obtenue. En particulier, on peut converger très lentement ou le système peut même disposer de plusieurs points fixes.

2.3/ CORRECTION ET COMPLÉTUDE DE LA DÉMARCHE

Cette section présente les théorèmes de correction et de complétude de l'approche. (Théorèmes 6 et 7). Toutes les preuves sont déplacées en annexes A.2.

Théorème 6 (Correction de la traduction vers Promela)**.** *Soit ϕ un modèle de système dynamique discret et ψ sa traduction PROMELA. Si ψ vérifie la propriété LTL (2.1) sous hypothèse d'équité faible, alors les itérations de ϕ sont universellement convergentes.*

Théorème 7 (Complétude de la traduction vers Promela)**.** *Soit ϕ un modèle de système dynamique discret et ψ sa traduction. Si ψ ne vérifie pas la propriété LTL (2.1) sous hypothèse d'équité faible, alors les itérations de ϕ ne sont pas universellement convergentes.*

2.4/ DONNÉES PRATIQUES

Cette section donne tout d'abord quelques mesures de complexité de l'approche puis présente ensuite les expérimentations issues de ce travail.

Théorème 8 (Nombre d'états)**.** *Soit ϕ un modèle de système dynamique discret à n éléments, m arcs dans le graphe d'incidence et ψ sa traduction en PROMELA. Le nombre de configurations de l'exécution en SPIN de ψ est bornée par $2^{m(\delta_0+1)+n(n+2)}$.*

Preuve. *Une configuration est une valuation des variables globales. Leur nombre ne dépend que de celles qui ne sont pas constantes.*

Les variables `Xp` *et* `X` *engendrent 2^{2n} états. La variable* `Xs` *génère 2^{n^2} états. Chaque canal de* `array_of_channels` *peut engendrer $1 + 2^1 + \ldots + 2^{\delta_0} = 2^{\delta_0+1} - 1$ états. Puisque le nombre d'arêtes du graphe d'incidence est m, il y a m canaux non constants, ce qui génère approximativement $2^{m(\delta_0+1)}$ états. Le nombre de configurations est donc borné par $2^{m(\delta_0+1)+n(n+2)}$. On remarque que cette borne est traitable par SPIN pour des valeurs raisonnables de n, m et δ_0.*

La méthode détaillée ici a pu être appliquée sur l'exemple pour prouver formellement sa convergence universelle.

On peut remarquer que SPIN n'impose l'équité faible qu'entre les process alors que les preuves des deux théorèmes précédentes reposent sur le fait que celle-ci est établie dès qu'un choix indéterministe est effectué. Naïvement, on pourrait considérer comme hypothèse la formule suivante chaque fois qu'un choix indéterministe se produit entre k événements respectivement notés $l_1, \ldots l_k$:

$$\Box \diamond (l == l_0) \Rightarrow ((\Box \diamond (l == l_1)) \wedge \ldots \wedge (\Box \diamond (l == l_k)))$$

où le libellé l_0 dénote le libellé de la ligne précédent le choix indéterministe. Cette formule traduit exactement l'équité faible. Cependant en raison de l'explosion de la taille du produit entre l'automate de Büchi issu de cette formule et celui issu du programme PROMELA, SPIN n'arrive pas à vérifier si la convergence universelle est établie ou non sur des exemples simples.

Ce problème a été pratiquement résolu en laissant SPIN générer toutes les traces d'exécution, même celles qui ne sont pas équitables, puis ensuite vérifier la propriété de convergence sur toutes celles-ci. Il reste alors à interpréter les résultats qui peuvent être de deux types. Si la convergence est établie pour toutes les traces, elle le reste en particulier pour les traces équitables. Dans le cas contraire on doit analyser le contre exemple produit par SPIN.

La méthode détaillée ici a été appliquée sur des exemples pour prouver formellement leur convergence ou leur divergence (FIGURE 2.11) avec ou sans délais. Dans ces expériences, les délais ont été bornés par $\delta_0 = 10$. Dans ce tableau, P est vrai ($\top$) si et seulement si la convergence universelle est établie et faux ($\bot$) sinon. Le nombre M est la taille de la mémoire consommée (en MB) et T est le temps d'exécution sur un Intel Centrino Dual Core 2 Duo @1.8GHz avec 2GB de mémoire vive pour établir un verdict.

	Synchrones			Généralisées		
	P	M	T	P	M	T
RE	$\top$	2.7	0.01s	$\bot$	369.371	0.509s
[Richard and Comet, 2007]	$\bot$	2.5	0.001s	$\bot$	2.5	0.01s
[Bahi and Michel, 1999]	$\top$	36.7	12s	$\top$		

(a) Sans délais

	Mode Mixe						Seulement borné					
	Synchrones			Pseudo-Périodique			Synchrones			Pseudo-Périodique		
	P	M	T	P	M	T	P	M	T	P	M	T
RE	$\top$	409	1m11s	$\bot$	370	0.54	$\bot$	374	7.7s	$\bot$	370	0.51s
AC2D	$\bot$	2.5	0.001s	$\bot$	2.5	0.01s	$\bot$	2.5	0.01s	$\bot$	2.5	0.01s
[Bahi and Michel, 1999]	$\top$			$\top$			$\bot$			$\bot$		

(b) Avec délais

FIGURE 2.11 – Résultats des simulations Promela des SDDs

L'exemple *RE* est l'exemple de ce chapitre, [Richard and Comet, 2007] concerne un réseau composé de deux gènes à valeur dans $\{0, 1, 2\}$, AC2D est un automate cellulaire avec 9 elements prenant des valeurs booléennes en fonction de de 4 voisins et [Bahi and Michel, 1999] consiste en 10 process qui modifient leurs valeurs booléennes dans un graphe d'adjacence proche du graphe complet.

L'exemple *RE* a été prouvé comme universellement convergent. *statuer sur AC2D* Comme la convergence n'est déjà pas établie pour les itérations synchrones de [Richard and Comet, 2007], il en est donc de même pour les itérations asynchrones. La FIGURE 2.12 donne une trace de la sortie de SPIN de menant à la violation de la convergence. Celle-ci correspond à une stratégie périodique qui répète $\{1,2\};\{1,2\};\{1\};\{1,2\};\{1,2\}$ et débute avec $x = (0,0)$. En raison de la dépendance forte entre les éléments de [Bahi and Michel, 1999], δ_0 est réduit à 1. Cela aboutit cependant à 2^{100} configurations dans le mode des itérations asynchrones.

Quid de ceci ? La convergence des itérations asynchrones de l'exemple [Bahi et al., 2010] n'est pas établie lorsque pour δ_0 vaut 1. Il ne peut donc y avoir convergence universelle.

2.5/ CONCLUSION

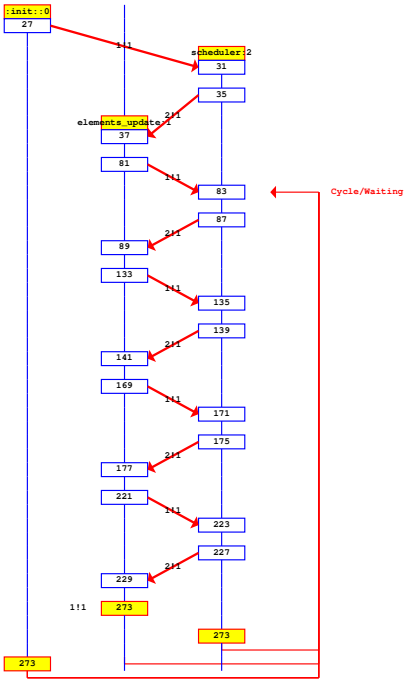


FIGURE 2.12 – Contre exemple de convergence pour 2.12

II

DES SYSTÈMES DYNAMIQUES DISCRETS AU CHAOS

3

CARACTERISATION DES SYSTÈMES DISCRETS CHAOTIQUES POUR LES SCHÉMAS UNAIRES ET GÉNÉRALISÉS

La première section rappelle ce que sont les systèmes dynamiques chaotiques. Dire que cette caractérisation dépend du type de stratégie : unaire (TIPE), généralisée (TSI). Pour chacune d'elle, on introduit une distance différente.

On montre qu'on a des résultats similaires.

3.1/ SYSTÈMES DYNAMIQUES CHAOTIQUES SELON DEVANEY

Dans cette partie, les définitions fondamentales liées au chaos dans les systèmes booléens sont rappelées et plusieurs résultats théoriques sont montrés.

Définition 4 (Chaos (Devaney))**.** *Une fonction k continue sur un espace métrique $(\mathcal{X}, d)$ est **chaotique** si elle est transitive, régulière et fortement sensible aux conditions initiales.*

Définition 5 (Transitivité)**.** *Une fonction k est **transitive** sur $(\mathcal{X}, d)$ si la propriété suivante est établie :*

$$\forall X, Y \in \mathcal{X}, \forall \epsilon > 0, \exists Z \in \mathcal{X}, \exists t \in \mathbb{N}, d(X, Z) < \epsilon \wedge k^t(Z) = Y$$

Définition 6 (Point périodique)**.** *Un point $P \in \mathcal{X}$ est dit **périodique** de période t pour une fonction k si t est un entier naturel non nul tel que $k^t(P) = P$ et pour tout n, $0 < n \leq t - 1$, on a $k^n(P) \neq P$. Par la suite, **Per(k)** dénote l'ensemble des points périodiques de k dans $\mathcal{X}$ de période quelconque.*

Définition 7 (Régularité)**.** *Une fonction k est dite **régulière** dans $(\mathcal{X}, d)$ si l'ensemble des points périodiques de k est dense dans $\mathcal{X}$, c'est-à-dire si la propriété suivante est établie :*

$$\forall X \in \mathcal{X}, \forall \epsilon > 0, \exists Y \in \mathit{Per}(k) \textit{ tel que } d(X, Y) < \epsilon.$$

Définition 8 (Forte sensibilité aux conditions initiales)**.** *Une fonction k définie sur $(\mathcal{X}, d)$ est **fortement sensible aux conditions initiales** s'il existe une valeur $\epsilon > 0$ telle que pour tout $X \in \mathcal{X}$ et pour tout $\delta > 0$, il existe $Y \in \mathcal{X}$ et $t \in \mathbb{N}$ qui vérifient $d(X, Y) < \delta$ et $d(k^t(X), k^t(Y)) > \epsilon$.*

John Banks et ses collègues ont cependant démontré que la sensibilité aux conditions initiales est une conséquence de la régularité et de la transitivité topologique [Banks et al., 1992]. On ne se focalise donc dans la suite que sur ces deux dernières propriétés pour caractériser les fonctions booléennes f rendant chaotique la fonction engendrée G_f.

3.2/ SCHÉMA UNAIRE

Soit N un entier naturel et f une fonction de $\mathbb{B}^{\mathsf{N}}$ dans lui-même.

3.2.1/ DES ITÉRATIONS UNAIRES AUX ITÉRATIONS PARALLÈLES

Dans le schéma unaire, à la $t^{\text{ème}}$ itération, seul le $s_t^{\text{ème}}$ composant (entre 1 et n) est mis à jour. Pour une stratégie $s = (s_t)_{t\in\mathbb{N}}$ (*i.e.*, une séquence d'indices de $[\![1;\mathsf{N}]\!]$), on peut définir la fonction $F_{f_u} : \mathbb{B}^{\mathsf{N}} \times [\![1;\mathsf{N}]\!]$ vers $\mathbb{B}^{\mathsf{N}}$ par

$$F_{f_u}(x,i) = (x_1,\dots,x_{i-1},f_i(x),x_{i+1},\dots,x_{\mathsf{N}}).$$

Dans le schéma des itérations unaires pour une configuration initiale $x^0 \in \mathbb{B}^{\mathsf{N}}$ et une stratégie $s \in [\![1;\mathsf{N}]\!]^{\mathbb{N}}$, les configurations x^t sont définies par la récurrence

$$x^{t+1} = F_{f_u}(x^t, s_t). \tag{3.1}$$

On peut alors construire l'espace $\mathcal{X}_u = \mathbb{B}^{\mathsf{N}} \times [\![1;\mathsf{N}]\!]^{\mathbb{N}}$ et la fonction d'iteration G_{f_u} définie de $\mathcal{X}_u$ dans lui-même par

$$G_{f_u}(x,s) = (F_{f_u}(x,s_0),\sigma(s)). \tag{3.2}$$

Dans cette définition, la fonction $\sigma : [\![1;\mathsf{N}]\!]^{\mathbb{N}} \longrightarrow [\![1;\mathsf{N}]\!]^{\mathbb{N}}$ décale la stratégie fournie en argument d'un élément vers la gauche en supprimant l'élément de tête. Ceci se formalise par

$$\sigma((u^k)_{k\in\mathbb{N}}) = (u^{k+1})_{k\in\mathbb{N}}.$$

Ainsi, effectuer des itérations unaires sur la fonction f selon une stratégie s revient à effectuer des itérations parallèles de la fonctions G_{f_u} dans $\mathcal{X}_u$. La section suivante introduit une métrique sur $\mathcal{X}_u$.

3.2.2/ UNE MÉTRIQUE POUR $\mathcal{X}_u$

Sur $\mathcal{X}_u$, on définit la distance d entre les points $X = (x,s)$ et $X' = (x',s')$ de $\mathcal{X}_u$ par

$$d(X,X') = d_H(x,x') + d_S(s,s'), \text{ où } \begin{cases} d_H(x,x') = \displaystyle\sum_{i=1}^{n} |x_i - x'_i| \\ d_S(s,s') = \dfrac{9}{n}\displaystyle\sum_{t\in\mathbb{N}} \dfrac{|s_t - s'_t|}{10^{t+1}}. \end{cases}.$$

On note que dans le calcul de $d_H(x,x')$– appelée distance de Hamming entre x et x'– les termes x_i et x'_i sont considérés comme des entiers naturels égaux à 0 ou à 1 et que le

calcul est effectué dans $\mathbb{Z}$. De plus, la partie entière $\lfloor d(X, X') \rfloor$ est égale à $d_H(x, x')$ soit la distance de Hamming entre x et x'. On remarque que la partie décimale est inférieure à 10^{-l} si et seulement si les l premiers termes des deux stratégies sont égaux. De plus, si la $(l+1)^{\text{ème}}$ décimale de $d_S(s, s')$ n'est pas nulle, alors s_l est différent de s'_l.

Se pose la question de caractériser les fonctions f telles que les itérations de G_{f_u} associées à leurs itérations unaires sont chaotiques dans $\mathcal{X}_u$. La section suivante apporte une réponse à cette question.

3.2.3/ CARACTÉRISATION DES FONCTIONS RENDANT CHAOTIQUES G_{f_u} SUR $\mathcal{X}_u$

On peut tout d'abord démontrer que pour toute fonction booléenne f, G_{f_u} est continue sur $\mathcal{X}_u$ (cf annexe B.1).

Pour charactériser les fonctions rendant chaotiques dans $\mathcal{X}_u$ les itérations de G_{f_u} on se focalise donc que sur la régularité et sur la transitivité de G_{f_u}. Ceci se réalise en établissant les relations d'inclusion entre les ensembles $\mathcal{T}$ des fonctions topologiquement transitives, $\mathcal{R}$ des fonctions régulières et $\mathcal{C}$ des fonctions chaotiques définis respectivement ci-dessous :

— $\mathcal{T} = \left\{ f : \mathbb{B}^n \to \mathbb{B}^n / G_{f_u} \text{ est transitive} \right\}$,
— $\mathcal{R} = \left\{ f : \mathbb{B}^n \to \mathbb{B}^n / G_{f_u} \text{ est régulière} \right\}$,
— $\mathcal{C} = \left\{ f : \mathbb{B}^n \to \mathbb{B}^n / G_{f_u} \text{ est chaotique} \right\}$.

On énnonce les théorèmes successifs suivants. Leur preuve est donnée en annexe B.2.

Théorème 9. *G_{f_u} est transitive si et seulement si* GIU(f) *est fortement connexe.*

Théorème 10. $\mathcal{T} \subset \mathcal{R}$.

On peut conclure que $\mathcal{C} = \mathcal{R} \cap \mathcal{T} = \mathcal{T}$. On a alors la caractérisation suivante :

Théorème 11. *Soit $f : \mathbb{B}^n \to \mathbb{B}^n$. La fonction G_{f_u} est chaotique si et seulement si* GIU(f) *est fortement connexe.*

3.3/ SCHÉMA GÉNÉRALISÉ

On reprend ici le même plan que dans la section précédente.

3.3.1/ DES ITÉRATIONS GÉNÉRALISÉES AUX ITÉRATIONS PARALLÈLES

Dans le schéma généralisé, à la $t^{\text{ème}}$ itération, c'est l'ensemble des $s_t^{\text{ème}}$ éléments (inclus dans $[n]$) qui sont mis à jour (c.f. équation (1.2)). On redéfinit la fonction la fonction $F_{f_g} : \mathbb{B}^{\mathsf{N}} \times \mathcal{P}(\{1, \ldots, \mathsf{N}\}) \to \mathbb{B}^{\mathsf{N}}$ par

$$F_{f_g}(x, s)_i = \begin{cases} f_i(x) \text{ si } i \in s\,; \\ x_i \text{ sinon.} \end{cases}$$

Dans ce schéma d'itérations généralisées, pour une configuration initiale $x^0 \in \mathbb{B}^{\mathsf{N}}$ et une stratégie $S = (s_t)_{t \in \mathbb{N}} \in \mathcal{P}(\{1, \ldots, \mathsf{N}\})^{\mathbb{N}}$, les configurations x^t sont définies par la récurrence

$$x^{t+1} = F_{f_g}(s_t, x^t). \tag{3.3}$$

Soit alors G_{f_g} une fonction de $\mathbb{B}^N \times \mathcal{P}(\{1,\dots,N\})^{\mathbb{N}}$ dans lui-même définie par

$$G_{f_g}(S,x) = (\sigma(S), F_{f_g}(s_0,x)),$$

où la fonction σ est définie comme à la section précédente. A nouveau, les itérations généralisées de f induites par x^0 et la stratégie S. décrivent la même orbite que les itérations parallèles de G_{f_g} depuis un point initial $X^0 = (x^0, S)$

On onstruit cette fois-ci l'espace $\mathcal{X}_g = \mathbb{B}^N \times \mathcal{P}(\{1,\dots,N\})^{\mathbb{N}}$

3.3.2/ UNE MÉTRIQUE POUR $\mathcal{X}_g$

Cette nouvelle distance va comparer des ensembles. On rappelle pour quelques notions ensemblistes. Pour A et B deux ensembles de l'univers Ω, on rappelle la définition de l'opérateur de *différence ensembliste* symétrique :

$$A \Delta B = (A \cap \overline{B}) \cup (\overline{A} \cap B)$$

où $\overline{B}$ désigne le complémentaire de B dans Ω.

On considère l'espace $\mathcal{X}_g = \mathcal{P}(\{1,\dots,N\})^{\mathbb{N}} \times \mathbb{B}^N$ et on définit la distance d entre les points $X = (S,x)$ et $X' = (S',x')$ de $\mathcal{X}_g$ par

$$d(X,X') = d_H(x,x') + d_S(S,S'), \text{ où } \begin{cases} d_H(x,x') = \sum_{i=1}^{N} |x_i - x'_i| \\ d_S(S,S') = \frac{9}{N} \sum_{t\in\mathbb{N}} \frac{|S_t \Delta S'_t|}{10^{t+1}}. \end{cases}.$$

La fonction d est une somme de deux fonctions. La fonction d_H est la distance de Hamming ; il est aussi établi que la somme de deux distances est une distance. Ainsi, pour montrer que d est aussi une distance, il suffit de montrer que d_S en une aussi, ce qui est fait en annexe B.3.

La section suivante caractérise les fonctions f qui sont chaotiques pour le schéma généralisées.

3.3.3/ CARACTÉRISATION DES FONCTIONS RENDANT CHAOTIQUES G_{f_g} SUR $\mathcal{X}_g$

On reprend les définitions des ensembles $\mathcal{T}$, $\mathcal{R}$ et $\mathcal{C}$ en les adaptant à G_{f_g}. On a les théorèmes suivants dont les preuves sont données en annexe B.4.

Théorème 12. *G_{f_g} est transitive si et seulement si* GIG(f) *est fortement connexe.*

Théorème 13. *$\mathcal{T} \subset \mathcal{R}$.*

Théorème 14. *Soit $f : \mathbb{B}^N \to \mathbb{B}^N$. La fonction G_{f_g} est chaotique si et seulement si* GIG(f) *est fortement connexe.*

3.4/ GÉNÉRER DES FONCTIONS CHAOTIQUES

Cette section présente deux approches permettant de générer des fonctions f dont le graphe des itérations GIU(f) est fortement connexe. La première est algorithmique mais grossière (section 3.4.1) tandis que la seconde s'appuie sur des conditions suffisantes sur le graphe d'interactions de la fonction booléenne f (Section 3.4.2).

3.4.1/ GÉNÉRATION ALGORITHMIQUE GROSSIÈRE

Cette section présente une première approche permettant de générer une fonction booléenne f dont le graphe des itérations GIU(f) est fortement connexe. La méthode est itérative et basée sur une démarche générer-tester.

On considère en premier lieu la fonction négation $\neg$ dont le graphe des itérations GIU($\neg$) est fortement connexe. Soit un graphe GIU, initialisé avec GIU($\neg$). L'algorithme effectue itérativement les deux étapes suivantes :

1. sélection aléatoire d'un arc du graphe d'itérations en cours, GIU, puis
2. analyse de la forte connexité du graphe d'itérations en cours auquel on enlèverait cet arc. Dans le cas positif, cet arc est enlevé de GIU,

et ce jusqu'à ce qu'un taux r d'arcs enlevés soit supérieur à un seuil donné par l'utilisateur. Si r est proche de 0% (*i.e.* peu d'arcs ont été supprimés), il reste peu ou prou $n \times 2^n$ arcs. Dans le cas contraire, si r est proche de 100%, il ne reste plus qu'environ 2^n arcs. Dans tous les cas, cette étape retourne un graphe GIU qui est fortement connexe. A partir du graphe GIU, il est alors facile de construire la fonction f.

Même si cet algorithme retourne toujours des fonctions dont le graphe des itérations est fortement connexe, il n'en est pas pour le moins efficace. Un défaut de l'algorithme est de nécessiter une vérification systématique de forte connexité sur le graphe entier composé de 2^N sommets et ce, à chaque itération ! La section suivante propose une solution à ce problème. Elle présente des conditions suffisantes sur un graphe à N sommets qui permettent d'obtenir des graphes d'itérations fortement connexes.

3.4.2/ CONDITIONS SUFFISANTES SUR LE GRAPHE D'INTERACTIONS

Cette partie énonce un théorème dont les hypothèses portent sur les interactions entre x_i et f_j et qui permet de n'engendrer que des fonctions f dont le graphe d'itérations GIU(f) est fortement connexe.

Théorème [15]. *Soit f une fonction de $\mathbb{B}^N$ vers lui-même telle que :*

1. *$\Gamma(f)$ n'a pas de cycle de longueur supérieure ou égale à deux ;*
2. *chaque sommet de $\Gamma(f)$ qui possède une boucle positive a aussi une boucle négative ;*
3. *chaque sommet de $\Gamma(f)$ est accessible depuis un sommet qui possède une boucle négative.*

Alors, GIU(f) *est fortement connexe.*

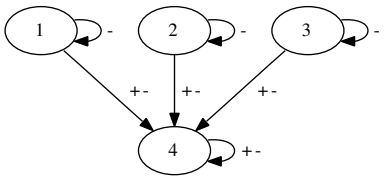


FIGURE 3.1 – Exemple de graphe d'interactions vérifiant le théorème 15

La preuve de ce théorème est donnée en annexe B.5.

Illustrons ce théorème par un exemple. On considère par le graphe d'interactions $\Gamma(f)$ donné en figure 3.1. Il vérifie le théorème 15 : toutes les fonctions f possédant un tel graphe d'interactions ont un graphe d'itérations GIU(f) fortement connexe. Pratiquement, il existe 34226 fonctions de $\mathbb{B}^4$ dans lui même qui vérifient ce graphe d'intéraction. Cependant, nombreuses sont celles qui possèdent un comportement équivalent. Deux fonctions sont equivalentes si leurs GIU sont isomorphes (au sens de l'isomorphisme de graphes). Il ne reste alors plus que 520 fonctions f non équivalentes de graphe d'interactions $\Gamma(f)$.

4

PRÉDICTION DES SYSTÈMES CHAOTIQUES

Les réseaux de neurones chaotiques ont été étudiés à de maintes reprises par le passé en raison notamment de leurs applications potentielles : les composants utils à la sécurité comme les fonctions de hachage [Li et al., 2010a], le tatouage numérique [Satish et al., 2004, Zhang et al., 2005] ou les schémas de chiffrement [Lian, 2009]. Dans tous ces cas, l'emploi de fonctions chaotiques est motivé par leur comportement imprévisibile et proche de l'aléa.

Les réseaux de neurones chaotiques peuvent être conçus selon plusieurs principes. Des neurones modifiant leur état en suivant une fonction non linéaire son par exemple appelés neurones chaotiques [Crook et al., 2007]. L'architecture de réseaux de neurones de type Perceptron multi-couches (MLP) n'iterent quant à eux, pas nécesssairement de fonctions chaotiques. Il a cependant été démontré que ce sont des approximateurs universels [Cybenko, 1989, Hornik et al., 1989]. Ils permettent, dans certains cas, de simuler des comportements physiques chaotiques comme le circuit de Chua [Dalkiran and Danisman, 2010]. Parfois [Li et al., 2010b], la fonction de transfert de cette famille de réseau celle d'initialisation sont toutes les deux définies à l'aide de fonctions chaotiques.

Ces réseaux de neurones partagent le fait qu'ils sont qualifiés de "chaotiques" sous prétexte qu'ils embarquent une fonction de ce type et ce sans aucune preuve rigoureuse. Ce chapitre caractérise la classe des réseaux de neurones MLP chaotiques. Il s'intéresse ensuite à l'étude de prévisibilité de systèmes dynamiques discrets chaotiques par cette famille de MLP.

revoir plan

The remainder of this research work is organized as follows. The next section is devoted to the basics of Devaney's chaos. Section 4.1 formally describes how to build a neural network that operates chaotically. Section 4.2 is devoted to the dual case of checking whether an existing neural network is chaotic or not. Topological properties of chaotic neural networks are discussed in Sect. **??**. The Section 4.3.1 shows how to translate such iterations into an Artificial Neural Network (ANN), in order to evaluate the capability for this latter to learn chaotic behaviors. This ability is studied in Sect. 4.3.2, where various ANNs try to learn two sets of data : the first one is obtained by chaotic iterations while the second one results from a non-chaotic system. Prediction success rates are given and discussed for the two sets. The paper ends with a conclusion section where our contribution is summed up and intended future work is exposed.

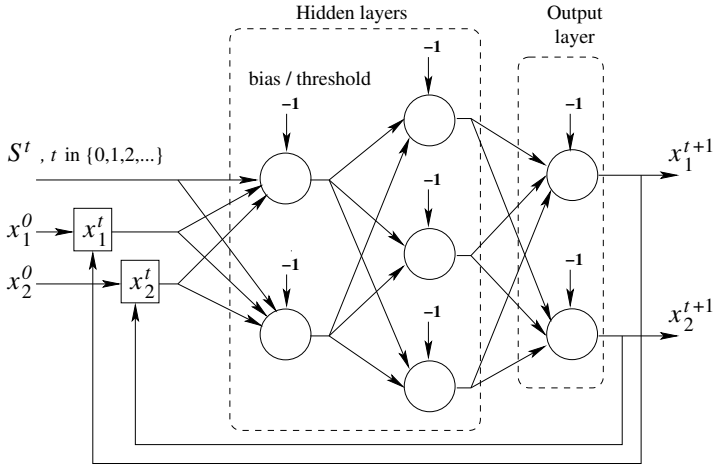


FIGURE 4.1 – Un perceptron équivalent aux itérations unitaires

4.1/ UN RÉSEAU DE NEURONES CHAOTIQUE AU SENS DE DEVANEY

On considère une fonction $f : \mathbb{B}^n \to \mathbb{B}^n$ telle que GIU(f) est fortement connexe. Ainsi G_{f_u} est chaotique d'après le théorème 21.

On considère ici le schéma unaire défini par l'équation (3.3). On construit un perceptron multi-couche associé à la fonction F_{f_u}. Plus précisément, pour chaque entrée $(x, s) \in \mathbb{B}^n \times [n]$, la couche de sortie doit générer $F_{f_u}(x, s)$. On peut ainsi lier la couche de sortie avec celle d'entrée pour représenter les dépendance entre deux itérations successives. On obtient une réseau de neurones dont le comportement est le suivant (voir Figure. 4.1) :

— Le réseau est initialisé avec le vecteur d'entrée $\left(x^0, S^0\right) \mathbb{B}^n \times [n]$ et calcule le vecteur de sortie $x^1 = F_{f_u}\left(x^0, S^0\right)$. Ce vecteur est publié comme une sortie et est aussi retournée sur la couche d'entrée à travers les liens de retours.
— Lorsque le réseau est activé à la t^{th} itération, l'état du système $x^t \in \mathbb{B}^n$ reçu depuis la couche de sortie ainsi que le premier terme de la sequence $(S^t)^{t\in\mathbb{N}}$ (*i.e.*, $S^0 \in [n]$) servent à construire le nouveau vecteur de sortie. Ce nouveau vecteur, qui représente le nouvel état du système dynamique, satisfait :

$$x^{t+1} = F_{f_u}(x^t, S^0) \in \mathbb{B}^n \ . \tag{4.1}$$

Le comportement de ce réseau de neurones est tel que lorsque l'état initial est composé de $x^0 \in \mathbb{B}^n$ et d'une séquence $(S^t)^{t\in\mathbb{N}}$, alors la séquence contenant les vecteurs successifs publiés $(x^t)^{t\in\mathbb{N}^*}$ est exactement celle produite par les itérations unaires décrites à la section 3.2. Mathématiquement, cela signifie que si on utilise les mêmes vecteurs d'entrées les deux approches génèrent successivement les mêmes sorties. En d'autres termes ce réseau de neurones modélise le comportement de G_{f_u}, dont les itérations sont chaotiques sur $\mathcal{X}_u$. On peut donc le qualifier de chaotique au sens de Devaney.

4.2/ VÉRIFIER SI UN RÉSEAU DE NEURONES EST CHAOTIQUE

On s'intéresse maintenant au cas où l'on dispose d'un réseau de neurones de type perceptron multi-couches dont on cherche à savoir s'il est chaotique (parce qu'il a par exemple été déclaré comme tel) au sens de Devaney. On considère de plus que sa topologie est la suivante : l'entrée est constituée de n bits et un entier, la sortie est constituée de n bits et chaque sortie est liée à une entrée par une boucle.

— Le réseau est initialisé avec n bits $\left(x_1^0, \ldots, x_n^0\right)$ et une valeur entière $S^0 \in [n]$.

— A l'itération t, le vecteur $\left(x_1^t, \ldots, x_n^t\right)$ permet de construire les n bits servant de sortie $\left(x_1^{t+1}, \ldots, x_n^{t+1}\right)$.

Le comportement de ce type de réseau de neurones peut être prouvé comme étant chaotique en suivant la démarche énoncée maintenant. On nomme tout d'abord $F : \mathbb{B}^n \times [n] \to \mathbb{B}^n$ la fonction qui associe au vecteur $((x_1, \ldots, x_n), s) \in \mathbb{B}^n \times [n]$ le vecteur $(y_1, \ldots, y_n) \in \mathbb{B}^n$, où $(y_1, \ldots, y_n)$ sont les sorties du réseau neuronal àaprès l'initialisation de la couche d'entrée avec $(s, (x_1, \ldots, x_n))$. Ensuite, on définie $f : \mathbb{B}^n \to \mathbb{B}^n$ telle que $f(x_1, x_2, \ldots, x_n)$ est égal à

$$(F((x_1, x_2, \ldots, x_n), 1), \ldots, F((x_1, x_2, \ldots, x_n)), n) \quad . \tag{4.2}$$

Ainsi pour chaque j, $1 \leq j \leq n$, on a $f_j(x_1, x_2, \ldots, x_n) = F((x_1, x_2, \ldots, x_n), j)$. Si ce réseau de neurones est initialisé avec $\left(x_1^0, \ldots, x_n^0\right)$ et $S \in [n]^{\mathbb{N}}$, il produit exactement les même sorties que les itérations de F_{f_u} avec une condition initiale $\left((x_1^0, \ldots, x_n^0), S\right) \in \mathbb{B}^n \times [n]^{\mathbb{N}}$. Les itérations de F_{f_u} sont donc un modèle formel de ce genre de réseau de neurones. Pour vérifier s'il est chaotique, il suffit ainsi de vérifier si le graphe d'itérations $\mathrm{GIU}(f)$ est fortement connexe.

4.3/ SUITABILITY OF FEEDFORWARD NEURAL NETWORKS FOR PREDICTING CHAOTIC AND NON-CHAOTIC BEHAVIORS

In the context of computer science different topic areas have an interest in chaos, as for steganographic techniques [Satish et al., 2004, Zhang et al., 2005]. Steganography consists in embedding a secret message within an ordinary one, while the secret extraction takes place once at destination. The reverse (*i.e.*, automatically detecting the presence of hidden messages inside media) is called steganalysis. Among the deployed strategies inside detectors, there are support vectors machines [Qiao et al., 2009], neural networks [Shaohui et al., 2003, Holoska et al., 2010], and Markov chains [Sullivan et al., 2006]. Most of these detectors give quite good results and are rather competitive when facing steganographic tools. However, to the best of our knowledge none of the considered information hiding schemes fulfills the Devaney definition of chaos [Devaney, 1989]. Indeed, one can wonder whether detectors continue to give good results when facing truly chaotic schemes. More generally, there remains the open problem of deciding whether artificial intelligence is suitable for predicting topological chaotic behaviors.

4.3.1/ Representing Chaotic Iterations for Neural Networks

The problem of deciding whether classical feedforward ANNs are suitable to approximate topological chaotic iterations may then be reduced to evaluate such neural networks on iterations of functions with Strongly Connected Component (SCC) graph of iterations. To compare with non-chaotic iterations, the experiments detailed in the following sections are carried out using both kinds of function (chaotic and non-chaotic). Let us emphasize on the difference between this kind of neural networks and the Chaotic Iterations based multilayer peceptron.

We are then left to compute two disjoint function sets that contain either functions with topological chaos properties or not, depending on the strong connectivity of their iterations graph. This can be achieved for instance by removing a set of edges from the iteration graph $\Gamma(f_0)$ of the vectorial negation function f_0. One can deduce whether a function verifies the topological chaos property or not by checking the strong connectivity of the resulting graph of iterations.

For instance let us consider the functions f and g from $\mathbb{B}^4$ to $\mathbb{B}^4$ respectively defined by the following lists :

$$[0,0,2,3,13,13,6,3,8,9,10,11,8,13,14,15]$$

$$\text{and } [11,14,13,14,11,10,1,8,7,6,5,4,3,2,1,0] \ .$$

In other words, the image of 0011 by g is 1110 : it is obtained as the binary value of the fourth element in the second list (namely 14). It is not hard to verify that $\Gamma(f)$ is not SCC (*e.g.*, $f(1111)$ is 1111) whereas $\Gamma(g)$ is. The remaining of this section shows how to translate iterations of such functions into a model amenable to be learned by an ANN. Formally, input and output vectors are pairs $((S^t)^{t\in\mathbb{N}}, x)$ and $\left(\sigma((S^t)^{t\in\mathbb{N}}), F_f(S^0, x)\right)$ as defined in Eq. (**??**).

Firstly, let us focus on how to memorize configurations. Two distinct translations are proposed. In the first case, we take one input in $\mathbb{B}$ per component ; in the second case, configurations are memorized as natural numbers. A coarse attempt to memorize configuration as natural number could consist in labeling each configuration with its translation into decimal numeral system. However, such a representation induces too many changes between a configuration labeled by a power of two and its direct previous configuration : for instance, 16 (10000) and 15 (01111) are close in a decimal ordering, but their Hamming distance is 5. This is why Gray codes [Gray, 1953] have been preferred.

Secondly, let us detail how to deal with strategies. Obviously, it is not possible to translate in a finite way an infinite strategy, even if both $(S^t)^{t\in\mathbb{N}}$ and $\sigma((S^t)^{t\in\mathbb{N}})$ belong to $\{1,\ldots,n\}^{\mathbb{N}}$. Input strategies are then reduced to have a length of size $l \in [\![2,k]\!]$, where k is a parameter of the evaluation. Notice that l is greater than or equal to 2 since we do not want the shift σ function to return an empty strategy. Strategies are memorized as natural numbers expressed in base $n+1$. At each iteration, either none or one component is modified (among the n components) leading to a radix with $n+1$ entries. Finally, we give an other input, namely $m \in [\![1, l-1]\!]$, which is the number of successive iterations that are applied starting from x. Outputs are translated with the same rules.

To address the complexity issue of the problem, let us compute the size of the data set an ANN has to deal with. Each input vector of an input-output pair is composed of a configuration x, an excerpt S of the strategy to iterate of size $l \in [\![2,k]\!]$, and a number $m \in [\![1, l-1]\!]$ of iterations that are executed.

Firstly, there are 2^n configurations x, with n^l strategies of size l for each of them. Secondly, for a given configuration there are $\omega = 1 \times n^2 + 2 \times n^3 + \ldots + (k-1) \times n^k$ ways of writing the pair (m, S). Furthermore, it is not hard to establish that

$$(n-1) \times \omega = (k-1) \times n^{k+1} - \sum_{i=2}^{k} n^i$$

then

$$\omega = \frac{(k-1) \times n^{k+1}}{n-1} - \frac{n^{k+1} - n^2}{(n-1)^2} \quad .$$

And then, finally, the number of input-output pairs for our ANNs is

$$2^n \times \left(\frac{(k-1) \times n^{k+1}}{n-1} - \frac{n^{k+1} - n^2}{(n-1)^2} \right) \quad .$$

For instance, for 4 binary components and a strategy of at most 3 terms we obtain 2304 input-output pairs.

4.3.2/ EXPERIMENTS

To study if chaotic iterations can be predicted, we choose to train the multilayer perceptron. As stated before, this kind of network is in particular well-known for its universal approximation property [Cybenko, 1989, Hornik et al., 1989]. Furthermore, MLPs have been already considered for chaotic time series prediction. For example, in [Dalkiran and Danisman, 2010] the authors have shown that a feedforward MLP with two hidden layers, and trained with Bayesian Regulation back-propagation, can learn successfully the dynamics of Chua's circuit.

In these experiments we consider MLPs having one hidden layer of sigmoidal neurons and output neurons with a linear activation function. They are trained using the Limited-memory Broyden-Fletcher-Goldfarb-Shanno quasi-newton algorithm in combination with the Wolfe linear search. The training process is performed until a maximum number of epochs is reached. To prevent overfitting and to estimate the generalization performance we use holdout validation by splitting the data set into learning, validation, and test subsets. These subsets are obtained through random selection such that their respective size represents 65%, 10%, and 25% of the whole data set.

Several neural networks are trained for both iterations coding schemes. In both cases iterations have the following layout : configurations of four components and strategies with at most three terms. Thus, for the first coding scheme a data set pair is composed of 6 inputs and 5 outputs, while for the second one it is respectively 3 inputs and 2 outputs. As noticed at the end of the previous section, this leads to data sets that consist of 2304 pairs. The networks differ in the size of the hidden layer and the maximum number of training epochs. We remember that to evaluate the ability of neural networks to predict a chaotic behavior for each coding scheme, the trainings of two data sets, one of them describing chaotic iterations, are compared.

Thereafter we give, for the different learning setups and data sets, the mean prediction success rate obtained for each output. Such a rate represents the percentage of input-output pairs belonging to the test subset for which the corresponding output value was correctly predicted. These values are computed considering 10 trainings with random

subsets construction, weights and biases initialization. Firstly, neural networks having 10 and 25 hidden neurons are trained, with a maximum number of epochs that takes its value in $\{125, 250, 500\}$ (see Tables 4.1 and 4.2). Secondly, we refine the second coding scheme by splitting the output vector such that each output is learned by a specific neural network (Table 4.3). In this last case, we increase the size of the hidden layer up to 40 neurons and we consider larger number of epochs.

TABLE 4.1 – Prediction success rates for configurations expressed as boolean vectors.

Networks topology : 6 inputs, 5 outputs, and one hidden layer				
Hidden neurons		10 neurons		
Epochs		125	250	500
Chaotic	Output (1)	90.92%	91.75%	91.82%
	Output (2)	69.32%	78.46%	82.15%
	Output (3)	68.47%	78.49%	82.22%
	Output (4)	91.53%	92.37%	93.4%
	Config.	36.10%	51.35%	56.85%
	Strategy (5)	1.91%	3.38%	2.43%
Non-chaotic	Output (1)	97.64%	98.10%	98.20%
	Output (2)	95.15%	95.39%	95.46%
	Output (3)	100%	100%	100%
	Output (4)	97.47%	97.90%	97.99%
	Config.	90.52%	91.59%	91.73%
	Strategy (5)	3.41%	3.40%	3.47%
Hidden neurons		25 neurons		
Epochs		125	250	500
Chaotic	Output (1)	91.65%	92.69%	93.93%
	Output (2)	72.06%	88.46%	90.5%
	Output (3)	79.19%	89.83%	91.59%
	Output (4)	91.61%	92.34%	93.47%
	Config.	48.82%	67.80%	70.97%
	Strategy (5)	2.62%	3.43%	3.78%
Non-chaotic	Output (1)	97.87%	97.99%	98.03%
	Output (2)	95.46%	95.84%	96.75%
	Output (3)	100%	100%	100%
	Output (4)	97.77%	97.82%	98.06%
	Config.	91.36%	91.99%	93.03%
	Strategy (5)	3.37%	3.44%	3.29%

Table 4.1 presents the rates obtained for the first coding scheme. For the chaotic data, it can be seen that as expected configuration prediction becomes better when the number of hidden neurons and maximum epochs increases : an improvement by a factor two is observed (from 36.10% for 10 neurons and 125 epochs to 70.97% for 25 neurons and 500 epochs). We also notice that the learning of outputs (2) and (3) is more difficult. Conversely, for the non-chaotic case the simplest training setup is enough to predict configurations. For all these feedforward network topologies and all outputs the obtained results for the non-chaotic case outperform the chaotic ones. Finally, the rates for the strategies show that the different feedforward networks are unable to learn them.

For the second coding scheme (*i.e.*, with Gray Codes) Table 4.2 shows that any network learns about five times more non-chaotic configurations than chaotic ones. As in the previous scheme, the strategies cannot be predicted. Figures 4.2 and 4.3 present the predictions given by two feedforward multilayer perceptrons that were respectively trained to

learn chaotic and non-chaotic data, using the second coding scheme. Each figure shows for each sample of the test subset (577 samples, representing 25% of the 2304 samples) the configuration that should have been predicted and the one given by the multilayer perceptron. It can be seen that for the chaotic data the predictions are far away from the expected configurations. Obviously, the better predictions for the non-chaotic data reflect their regularity.

Let us now compare the two coding schemes. Firstly, the second scheme disturbs the learning process. In fact in this scheme the configuration is always expressed as a natural number, whereas in the first one the number of inputs follows the increase of the Boolean vectors coding configurations. In this latter case, the coding gives a finer information on configuration evolution.

Unfortunately, in practical applications the number of components is usually unknown. Hence, the first coding scheme cannot be used systematically. Therefore, we provide a refinement of the second scheme : each output is learned by a different ANN. Table 4.3 presents the results for this approach. In any case, whatever the considered feedforward network topologies, the maximum epoch number, and the kind of iterations, the configuration success rate is slightly improved. Moreover, the strategies predictions rates reach almost 12%, whereas in Table 4.2 they never exceed 1.5%. Despite of this improvement, a long term prediction of chaotic iterations still appear to be an open issue.

4.4/ CONCLUSION

In this paper, we have established an equivalence between chaotic iterations, according to the Devaney's definition of chaos, and a class of multilayer perceptron neural networks. Firstly, we have described how to build a neural network that can be trained to learn a given chaotic map function. Secondly, we found a condition that allow to check whether the iterations induced by a function are chaotic or not, and thus if a chaotic map is obtained. Thanks to this condition our approach is not limited to a particular function. In the dual case, we show that checking if a neural network is chaotic consists in verifying a property on an associated graph, called the graph of iterations. These results are valid

TABLE 4.2 – Prediction success rates for configurations expressed with Gray code

Networks topology : 3 inputs, 2 outputs, and one hidden layer				
	Hidden neurons	10 neurons		
	Epochs	125	250	500
Chaotic	Config. (1)	13.29%	13.55%	13.08%
	Strategy (2)	0.50%	0.52%	1.32%
Non-Chaotic	Config. (1)	77.12%	74.00%	72.60%
	Strategy (2)	0.42%	0.80%	1.16%
	Hidden neurons	25 neurons		
	Epochs	125	250	500
Chaotic	Config. (1)	12.27%	13.15%	13.05%
	Strategy (2)	0.71%	0.66%	0.88%
Non-Chaotic	Config. (1)	73.60%	74.70%	75.89%
	Strategy (2)	0.64%	0.97%	1.23%

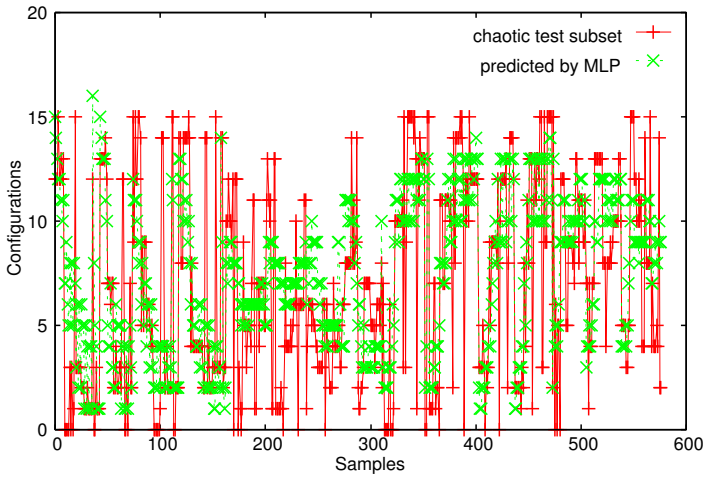


FIGURE 4.2 – Second coding scheme - Predictions obtained for a chaotic test subset.

for recurrent neural networks with a particular architecture. However, we believe that a similar work can be done for other neural network architectures. Finally, we have discovered at least one family of problems with a reasonable size, such that artificial neural networks should not be applied in the presence of chaos, due to their inability to learn chaotic behaviors in this context. Such a consideration is not reduced to a theoretical detail : this family of discrete iterations is concretely implemented in a new steganographic method [Bahi and Guyeux, 2010b]. As steganographic detectors embed tools like neural networks to distinguish between original and stego contents, our studies tend to prove that such detectors might be unable to tackle with chaos-based information hiding schemes.

In future work we intend to enlarge the comparison between the learning of truly chaotic and non-chaotic behaviors. Other computational intelligence tools such as support vector machines will be investigated too, to discover which tools are the most relevant when facing a truly chaotic phenomenon. A comparison between learning rate success and prediction quality will be realized. Concrete consequences in biology, physics, and computer science security fields will then be stated.

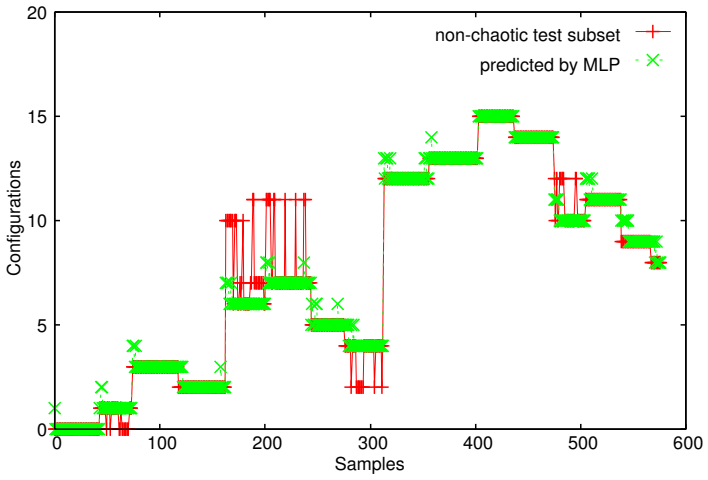


FIGURE 4.3 – Second coding scheme - Predictions obtained for a non-chaotic test subset.

TABLE 4.3 – Prediction success rates for split outputs.

Networks topology : 3 inputs, 1 output, and one hidden layer			
Epochs	125	250	500
Chaotic	Output = Configuration		
10 neurons	12.39%	14.06%	14.32%
25 neurons	13.00%	14.28%	14.58%
40 neurons	11.58%	13.47%	14.23%
Non chaotic	Output = Configuration		
10 neurons	76.01%	74.04%	78.16%
25 neurons	76.60%	72.13%	75.96%
40 neurons	76.34%	75.63%	77.50%
Chaotic/non chaotic	Output = Strategy		
10 neurons	0.76%	0.97%	1.21%
25 neurons	1.09%	0.73%	1.79%
40 neurons	0.90%	1.02%	2.15%

Epochs	1000	2500	5000
Chaotic	Output = Configuration		
10 neurons	14.51%	15.22%	15.22%
25 neurons	16.95%	17.57%	18.46%
40 neurons	17.73%	20.75%	22.62%
Non chaotic	Output = Configuration		
10 neurons	78.98%	80.02%	79.97%
25 neurons	79.19%	81.59%	81.53%
40 neurons	79.64%	81.37%	81.37%
Chaotic/non chaotic	Output = Strategy		
10 neurons	3.47%	9.98%	11.66%
25 neurons	3.92%	8.63%	10.09%
40 neurons	3.29%	7.19%	7.18%

III

CONCLUSION ET PERSPECTIVES

Perspectives pour SDD-¿Promela Among drawbacks of the method, one can argue that bounded delays is only realistic in practice for close systems. However, in real large scale distributed systems where bandwidth is weak, this restriction is too strong. In that case, one should only consider that matrix s^t follows the iterations of the system, *i.e.*, for all i, j, $1 \leq i \leq j \leq n$, we have $\lim_{t\to\infty} s^t_{ij} = +\infty$. One challenge of this work should consist in weakening this constraint. We plan as future work to take into account other automatic approaches to discharge proofs notably by deductive analysis [Couchot et al., 2005].

A

PREUVES SUR LES SDD

A.1/ CONVERGENCE DU MODE MIXE

Introduisons tout d'abord une relation d'ordre $\preceq$ entre les classes d'équivalences. Formellement, $\langle p\rangle \preceq \langle q\rangle$ s'il existe un chemin de longueur α $(0 < \alpha < |\mathcal{K}|)$ entre un élément le la classe $\langle p\rangle$ vers un élément de $\langle q\rangle$. On remarque que si la $\langle p\rangle \preceq \langle q\rangle$, il n'est alors pas possible que $\langle q\rangle \preceq \langle p\rangle$.

Lemme [1]. *Il existe un processus de renommage qui effecte un nouvel identifiant aux élément $i \in \langle p\rangle$ et $j \in \langle q\rangle$ tel que $i \leq j$ si et seulement si $\langle p\rangle \preceq \langle q\rangle$.*

Preuve. *Tout d'abord, soit $\langle p_1\rangle, \ldots, \langle p_l\rangle$ des classes contenant respectivement $n_1, \ldots, n_l$ éléments respectively qui ne dépendent d'aucune autre classe. Les éléments de $\langle p_1\rangle$ sont renommés par $1, \ldots, n_1$, les elements de $\langle p_i\rangle$, $2 \leq i \leq l$ sont renommés par $1 + \Sigma_{k=1}^{i-1} n_k$, $\ldots$, $\Sigma_{k=1}^{i} n_k$. On considère maintenant les classes $\langle p_1\rangle, \ldots, \langle p_{l'}\rangle$ dont les éléments ont été renommés et soit m le plus grand indice des elements de $\langle p_1\rangle, \ldots, \langle p_{l'}\rangle$. Soit une autre classe $\langle p\rangle$ qui dépend exclusivement d'une classe $\langle p_i\rangle$, $1 \leq i \leq l'$ et qui contient k elements. Les éléments de $\langle p\rangle$ sont renommés par $m+1, \ldots, m+k$. Ce processus a été appliqué sur $l'+1$ classes. Il se termine puisqu'il diminue le nombre d'elements auquel il reste à affecter un numéro.*

Il reste à montrer que cette méthode de renommage vérifie la propriété énoncée dans le lemme. Cette preuve se fait par induction sur la taille l du plus grand chemin de dépendance entre les classes.

Tout d'abord, si $\langle p\rangle \preceq \langle q\rangle$ et $\langle q\rangle$ dépend immédiatement de $\langle p\rangle$, i.e. le chemin le plus long entre les éléments de $\langle p\rangle$ et les elements de $\langle q\rangle$ est de longueur 1. En raison de la méthode renommage, chaque numéro d'élément $\langle q\rangle$ est plus grand que tous ceux de $\langle p\rangle$ et la preuve est établie. Soit $\langle p\rangle$ et $\langle q\rangle$ tels que le plus long chemin de dépendance entre $\langle p\rangle$ et $\langle q\rangle$ a une longueur de $l+1$. Il existe alors une classe $\langle q'\rangle$ telle que $\langle q\rangle$ dépend immédiatement de $\langle q'\rangle$ et le chemin de dépendance le plus long entre $\langle p\rangle$ et $\langle q'\rangle$ a pour longueur l. On a ainsi $\langle q'\rangle \preceq \langle q\rangle$ et pour tout k, j tels que $k \in \langle q'\rangle$ et $j \in \langle q\rangle$, $k \leq j$. Par hypothèse d'induction, $\langle p\rangle \preceq \langle q'\rangle$ et pour chaque i, k tels que $i \in \langle p\rangle$ et $k \in \langle q'\rangle$, $i \leq k$ et le résultat est établi.

On peut remarquer que ce processus de renommage est inspiré des *graphes par couches* de Golès et Salinas [Goles Ch. and Salinas, 2008].

Preuve (of Theorem 5)**.** *Le reste de la preuve est fait par induction sur le numéro de*

classe. Considérons la première classe $\langle b_1 \rangle$ de n_1 éléments i.e. la classe avec le plus petit identifiant.

D'après les hypothèses du théorème, les itérations synchrones convergent vers un point fixe en un nombre fini d'itérations. Ainsi toutes les classes sources *(indépendantes de toutes les autres classes) vont aussi converger dans le mode mixe. On peut ainsi supposer que le mode d'itération mixe avec délais uniformes fait converger les classes $\langle b_1 \rangle$, ..., $\langle b_k \rangle$ en un temps t_k. Par construction, la classe $\langle b_{k+1} \rangle$ dépend uniquement de certaines classes de $\langle b_1 \rangle$, ..., $\langle b_k \rangle$ et éventuellement d'elle-même. Il existe un nombre d'iteration suffisamment grand t_0 tel que $D^{t_0}_{p_{k+1}p_j}$ est suppérieur ou égal à t_k pour chaque $p_{k+1} \in \langle b_{k+1} \rangle$ et $p_j \in \langle b_j \rangle$, $1 \leq j \leq k$.*

Il nous reste donc des itérations synchronous entre les elements of $\langle b_{k+1} \rangle$ en démarant dans des configurations où tous les éléments de $\langle b_j \rangle$, $1 \leq j \leq k$, on des valeurs constantes. D'après les hypothèses du théorème, cela converge.

A.2/ CORRECTION ET COMPLÉTUDE DE LA VÉRIFICATION DE CONVERGENCE PAR SPIN

Voir section 2.3

Cette section donne les preuves des deux théorèmes de correction et complétude du chapitre 2.

Lemme [2] (Strategy Equivalence)**.** *Soit ϕ un système dynamique discret de stratégie $(S^t)^{t\in\mathbb{N}}$ et ψ sa traduction en promela. Il existe une exécution de ψ sous hypothèse d'équité faible telle le le scheduler met à jour les elements of S^t donnés par* `update_elems` *à l'iteration t.*

Preuve. *La preuve est directe pour $t = 0$. Supposons qu'elle est établie jusqu'en t vallant un certain t_0. On considère des stratégies pseudo périodiques. Grâce à l'hypothèse d'équité faible,* `update_elems` *modifie les éléments de S^t à l'iteration t.*

Dans ce qui suit, soit Xd^t_{ji} la valeur de `Xd[j].v[i]` après le t^{th} appel à la fonction `fetch_values`. De plus, soit Y^k_{ij} l'élément à l'indice k dans le canal `channels[i].sent[j]` de taille m, $m \leq \delta_0$; Y^0_{ij} et Y^{m-1}_{ij} sont respectivement la tête et la queue du canal. De plus, soit $(M^t_{ij})^{t\in\{1,1.5,2,2.5,...\}}$ une séquence telle que M^t_{ij} est une fonction partielle qui associe à chaque k, $0 \leq k \leq m-1$, le tuple $(Y^k_{ij}, a^k_{ij}, c^k_{ij})$ en entrant dans la fonction `update_elems` à l'itération t où Y^k_{ij} est la valeur du cannal `channels[i].sent[j]` à l'indice k, a^k_{ij} est la date (antérieure à t) mémorisant quand Y^k_{ij} est ajouté et c^k_{ij} est le premier temps où cette valeur est accessible à j. La valeur est supprimée du canal $i \to j$ à la date $c^k_{ij} + 1$. M^t_{ij} a la signature suivante :

$$\begin{array}{rccl} M^t_{ij}: & \{0,\ldots,max-1\} & \to & E_i \times \mathbb{N} \times \mathbb{N} \\ & k \in \{0,\ldots,m-1\} & \mapsto & M_{ij}(k) = (Y^k_{ij}, a^k_{ij}, c^k_{ij}). \end{array}$$

Intuitivement, M^t_{ij} est la mémoire du cannal `channels[i].sent[j]` à l'iterations t. On note que le domaine de chaque M^1_{ij} est $\{0\}$ et $M^1_{ij}(0) = (\texttt{Xp[i]}, 0, 0)$: en effet le processus `init` initialise `channels[i].sent[j]` avec `Xp[i]`.

Montrons comment l'indéterminisme des deux fonctions `fetch_values` et `diffuse_values` permet de modéliser l'équation (1.5). La function M_{ij}^{t+1} est obtenue à l'aide de mises à jour successives de M_{ij}^{t} au travers des deux functions `fetch_values` and `diffuse_values`. Par abus, soit $M_{ij}^{t+1/2}$ la valeur de M_{ij}^{t} après la première fonctions pendant l'itération t.

Dans ce qui suit, on considère les éléments i et j dans $[\![n]\!]$. A l'itération t, $t \geq 1$, soit $(Y_{ij}^0, a_{ij}^0, c_{ij}^0)$ la valeur de $M_{ij}^t(0)$ en entrant dans la fonction `fetch_values`. Si t est égal à $c_{ij}^0 + 1$ alors on exécute l'instruction qui affecte Y_{ij}^0 (*i.e.*, la valeur de tête du `channels[i].sent[j]`) à Xd_{ji}^t. Dans ce cas, la fonction M_{ij}^t est mise à jour comme suit : $M_{ij}^{t+1/2}(k) = M_{ij}^t(k+1)$ pour chaque k, $0 \leq k \leq m-2$ et $m-1$ est supprimée du domaine de $M_{ij}^{t+1/2}$. Sinon, (*i.e.*, lorsque $t < c_{ij}^0 + 1$ ou lorsque le domaine de M_{ij} est vide) l'instruction `skip` est exécutée et $M_{ij}^{t+1/2} = M_{ij}^t$.

Dans la fonction `diffuse_values`, s'il existe un τ, $\tau \geq t$ tel que $D_{ji}^{\tau} = t$, soit alors c_{ij} défini par $\min\{l \mid D_{ji}^l = t\}$. Dans ce cas, on exécution l'instruction qui ajoute la valeur `Xp[i]` dans la queue du cannal `channels[i].sent[j]`. Alors, M_{ij}^{t+1} est défini en étendant $M_{ij}^{t+1/2}$ à m de sorte que $M_{ij}^{t+1}(m)$ est $(\texttt{Xp[i]}, t, c_{ij})$. Sinon, (*i.e.*, lorsque $\forall l \,.\, l \geq t \Rightarrow D_{ji}^l \neq t$ est établie) l'instruction `skip` est exécutée et $M_{ij}^{t+1} = M_{ij}^{t+1/2}$.

Lemme [3] (Existence d'une exécution SPIN)**.** *Pour chaque sequence $(S^t)^{t\in\mathbb{N}}$, $(D^t)^{t\in\mathbb{N}}$, pour chaque fonction F, il existe une exécution SPIN telle que pour toute itération t, $t \geq 1$, et pour chaque i et j in $[\![n]\!]$ on a la propriété suivante :*

Si le domaine de M_{ij}^t n'est pas vide, alors

$$\begin{cases} M_{ij}^1(0) & = & \left(X_i^{D_{ji}^0}, 0, 0\right) \\ \textit{sit } t \geq 2 \textit{ alors } M_{ij}^t(0) & = & \left(X_i^{D_{ji}^c}, D_{ji}^c, c\right),\ c = \min\{l | D_{ji}^l > D_{ji}^{t-2}\} \end{cases} \tag{A.1}$$

De plus, on a :

$$\forall t' \,.\, 1 \leq t' \leq t \Rightarrow Xd_{ji}^{t'} = X_i^{D_{ji}^{t'-1}} \tag{A.2}$$

Enfin, pour chaque $k \in S^t$, la valeurde la variable `Xp[k]` en sortant du processus `update_elems` est égale à X_k^t i.e., $F_k\left(X_1^{D_{k1}^{t-1}}, \ldots, X_n^{D_{kn}^{t-1}}\right)$ à la fin de la t^{th} itération.

Preuve. *La preuve est faite par induction sur le nombre d'itérations.*

Situation initiale : *Pour le premier item, par definition de M_{ij}^t, on a $M_{ij}^1(0) = (\texttt{Xp[i]}, 0, 0)$ qui est égal à $\left(X_i^{D_{ji}^0}, 0, 0\right)$. Ensuite, lepremier appel à la fonction `fetch_value` soit affecte à la tête de `channels[i].sent[j]` à `Xd[j].v[i]` soit ne modifie par `Xd[j].v[i]`. Grâce au processus `init` process, les deux cas sont égaux à `Xp[i]`, i.e., X_i^0. L'equation (A.2) est ainsi établie.*

Pour le dernier item, soit k, $0 \leq k \leq n-1$. A la fin de la première exécution du processus `update_elems`, la valur de `Xp[k]` est $F(\texttt{Xd}[k]\texttt{.v[0]}, \ldots, \texttt{Xd}[k]\texttt{.v}[n-1])$. Ainsi par définition de Xd, ceci est égal à $F(Xd_{k0}^1, \ldots, Xd_{kn-1}^1)$. Grâce à l'équation (A.2), on peut conclure la preuve.

Induction : *Supposons maintenant que le lemme 3 est établi jusqu'à l'itération l.*

Tout d'abord, si le domaine de définition de la fonction M^l_{ij} n'est pas vide, par hypothèse d'induction $M^l_{ij}(0)$ est $\left(X_i^{D^c_{ji}}, D^c_{ji}, c\right)$ où c est $\min\{k|D^k_{ji} > D^{l-2}_{ji}\}$.

A l'itération l, si $l < c+1$ alors `skip` *statement is executed in the* `fetch_values` *function. Thus, $M^{l+1}_{ij}(0)$ is equal to $M^l_{ij}(0)$. Since $c > l-1$ then $D^c_{ji} > D^{l-1}_{ji}$ and hence, c is $\min\{k|D^k_{ji} > D^{l-1}_{ji}\}$. Obviously, this implies also that $D^c_{ji} > D^{l-2}_{ji}$ and $c = \min\{k|D^k_{ji} > D^{l-2}_{ji}\}$.*

We now consider that at iteration l, l is $c+1$. In other words, M_{ij} is modified depending on the domain dom(M^l_{ij}) of M^l_{ij} :

- *if dom$(M^l_{ij}) = \{0\}$ and $\forall k\,.\,k \geq l \Rightarrow D^k_{ji} \neq l$ is established then dom(M^{l+1}_{ij}) is empty and the first item of the lemma is established ;*
- *if dom$(M^l_{ij}) = \{0\}$ and $\exists k\,.\,k \geq l \wedge D^k_{ji} = l$ is established then $M^{l+1}_{ij}(0)$ is $(\texttt{Xp[i]}, l, c_{ij})$ that is added in the* `diffuse_values` *function s.t. $c_{ij} = \min\{k \mid D^k_{ji} = l\}$. Let us prove that we can express $M^{l+1}_{ij}(0)$ as $\left(X_i^{D^{c'}_{ji}}, D^{c'}_{ji}, c'\right)$ where c' is $\min\{k|D^k_{ji} > D^{l-1}_{ji}\}$. First, it is not hard to establish that $D^{c_{ij}}_{ji} = l \geq D^l_{ji} > D^{l-1}_{ji}$ and thus $c_{ij} \geq c'$. Next, since dom$(M^l_{ij}) = \{0\}$, then between iterations $D^c_{ji}+1$ and $l-1$, the* `diffuse_values` *function has not updated M_{ij}. Formally we have*

$$\forall t, k\,.\,D^c_{ji} < t < l \wedge k \geq t \Rightarrow D^k_{ji} \neq t.$$

Particularly, $D^{c'}_{ji} \notin \{D^c_{ji}+1, \ldots, l-1\}$. We can apply the third item of the induction hypothesis to deduce $\texttt{Xp[i]} = X_i^{D^{c'}_{ji}}$ and we can conclude.

- *if $\{0,1\} \subseteq$ dom(M^l_{ij}) then $M^{l+1}_{ij}(0)$ is $M^l_{ij}(1)$. Let $M^l_{ij}(1) = \left(\texttt{Xp[i]}, a_{ij}, c_{ij}\right)$. By construction a_{ij} is $\min\{t'|t' > D^c_{ji} \wedge (\exists k\,.\,k \geq t' \wedge D^k_{ji} = t')\}$ and c_{ij} is $\min\{k|D^k_{ji} = a_{ij}\}$. Let us show c_{ij} is equal to $\min\{k|D^k_{ji} > D^{l-1}_{ji}\}$ further referred as c'. First we have $D^{c_{ij}}_{ji} = a_{ij} > D^c_{ji}$. Since c by definition is greater or equal to $l-1$, then $D^{c_{ij}}_{ji} > D^{l-1}_{ji}$ and then $c_{ij} \geq c'$. Next, since c is $l-1$, c' is $\min\{k|D^k_{ji} > D^c_{ji}\}$ and then $a_{ij} \leq D^{c'}_{ji}$. Thus, $c_{ij} \leq c'$ and we can conclude as in the previous part.*

The case where the domain dom(M^l_{ij}) is empty but the formula $\exists k\,.\,k \geq l \wedge D^k_{ji} = l$ is established is equivalent to the second case given above and then is omitted.

Secondly, let us focus on the formula (A.2). At iteration $l+1$, let c' be defined as $\min\{k|D^k_{ji} > D^{l-1}_{ji}\}$. Two cases have to be considered depending on whether D^l_{ji} and D^{l-1}_{ji} are equal or not.

- *If $D^l_{ji} = D^{l-1}_{ji}$, since $D^{c'}_{ji} > D^{l-1}_{ji}$, then $D^{c'}_{ji} > D^l_{ji}$ and then c' is distinct from l. Thus, the SPIN execution detailed above does not modify Xd^{l+1}_{ji}. It is obvious to establish that $Xd^{l+1}_{ji} = Xd^l_{ji} = X_i^{D^{l-1}_{ji}} = X_i^{D^l_{ji}}$.*
- *Otherwise D^l_{ji} is greater than D^{l-1}_{ji} and c is thus l. According to (A.1) we have proved, we have $M^{l+1}_{ij}(0) = (X_i^{D^l_{ji}}, D^l_{ji}, l)$. Then the SPIN execution detailed above assigns $X_i^{D^l_{ji}}$ to Xd^{l+1}_{ji}, which ends the proof of (A.2).*

We are left to prove the induction of the third part of the lemma. Let k, $k \in S^{l+1}$. At the end of the first execution of the `update_elems` *process, we have* `Xp[`k`]` $= F($`Xd[`k`][0]`$, \ldots,$ `Xd[`k`][`n`-1]`$)+$. *By definition of Xd, it is equal to $F(Xd^{l+1}_{k0}, \ldots, Xd^{l+1}_{kn-1})$. Thanks to (A.2) we have proved, we can conclude the proof.*

Lemme 4. *Bounding the size of channels to max = δ_0 is sufficient when simulating a DDN where delays are bounded by δ_0.*

Preuve. *For any i, j, at each iteration $t+1$, thanks to bounded delays (by δ_0), element i has to know at worst δ_0 values that are $X_j^t, \ldots, X_j^{t-\delta_0+1}$. They can be stored into any channel of size δ_0.*

Théorème 6 (Correction de la traduction vers Promela)**.** *Soit ϕ un modèle de système dynamique discret et ψ sa traduction PROMELA. Si ψ vérifie la propriété LTL (2.1) sous hypothèse d'équité faible, alors les itérations de ϕ sont universellement convergentes.*

Preuve. *Let us show the contraposition of the theorem. The previous lemmas have shown that for any sequence of iterations of the DDN, there exists an execution of the PROMELA model that simulates them. If some iterations of the DDN are divergent, then they prevent the PROMELA model from stabilizing, i.e., not verifying the LTL property (2.1).*

Théorème 7 (Complétude de la traduction vers Promela)**.** *Soit ϕ un modèle de système dynamique discret et ψ sa traduction. Si ψ ne vérifie pas la propriété LTL (2.1) sous hypothèse d'équité faible, alors les itérations de ϕ ne sont pas universellement convergentes.*

Preuve. *For models ψ that do not verify the LTL property (2.1) it is easy to construct corresponding iterations of the DDN, whose strategy is pseudo-periodic since weak fairness property is taken into account.*

B

PREUVES SUR LES SYSTÈMES CHAOTIQUES

B.1/ CONTINUITÉ DE G_f DANS $(\mathcal{X}_u, d)$

Montrons que pour toute fonction booléenne f de $\mathbb{B}^n$ dans lui même, G_f est continue sur $(\mathcal{X}, d)$.

Soit donc $(s_t, x^t)^{t\in\mathbb{N}}$ une suite de points de l'espace $\mathcal{X}$ qui converge vers (s, x). Montrons que $(G_f(s_t, x^t))^{t\in\mathbb{N}}$ converge vers $G_f(s, x)$.

La distance $d((s_t, x^t), (s, x))$ tend vers 0. Il en est donc de même pour $d_H(x^t, x)$ et $d_S(s_t, s)$. Or, $d_H(x^t, x)$ ne prend que des valeurs entières. Cette distance est donc nulle à partir d'un certain t_0. Ainsi, à partir de $t > t_0$, on a $x^t = x$. De plus, $d_S(s_t, s)$ tend vers 0 donc $d_S(s_t, s) < 10^{-1}$ à partir d'un certain rang t_1. Ainsi, à partir de $t > t_1$, les suites $(s_t)_{t\in\mathbb{N}}$ ont toutes le même premier terme, qui est celui de s pour t supérieur à t_1. Pour $t > \max(t_0, t_1)$, les configurations x^t et x sont les mêmes, et les stratégies s_t et s ont le même premier terme ($s_0^t = s_0$), donc les configurations de $F_f(s_0^t, x^t)$ et de $F_f(s_0, x)$ sont égales et donc la distance entre $G_f(s_t, x^t)$ et $G_f(s, x)$ est inférieure à 1.

Montrons maintenant que la distance entre $G_f(s_t, x^t)$ et $G_f(s, x)$ tend bien vers 0 quand t tend vers $+\infty$. Soit $\epsilon > 0$.

- Si $\epsilon \geq 1$. Comme la distance $d(G_f(s_t, x^t), G_f(s, x)) < 1$ pour $t > \max(t_0, t_1)$, alors $d(G_f(s_t, x^t), G_f(s, x)) < \epsilon$
- Si $\epsilon < 1$, alors $\exists k \in \mathbb{N}$ tel que $10^{-k} > \epsilon > 10^{-(k+1)}$. Comme $d_S(s_t, s)$ tend vers 0, il existe un rang t_2 à partir duquel $\forall t > t_2, d_S(s_t, s) < 10^{-(k+2)}$: à partir de ce rang, les $k + 2$ premiers termes de s_t sont ceux de s. Donc les $k + 1$ premiers termes des stratégies de $G_f(s_t, x^t)$ et de $G_f(s, x)$ sont les mêmes (puisque G_f opère un décalage sur les stratégies), et vue la définition de d_S, la partie décimale de la distance entre les points (s_t, x^t) et (s, x) est inférieure à $10^{-(k+1)} \leq \epsilon$.

Pour conclure, pour tout $\epsilon > 0$, $\exists\ T_0 = \max(t_0, t_1, t_2) \in \mathbb{N}$ tel que $\forall t > T_0, d(Gf(s_t, x^t), G_f(s, x)) < \epsilon$.

B.2/ CARACTÉRISATION DES FONCTIONS f RENDANT CHAOTIQUE G_{f_u} DANS $(\mathcal{X}_u, d)$

On prouve les théorèmes suivants

Théorème [16]. *G_{f_u} est transitive si et seulement si* GIU(f) *est fortement connexe.*

Preuve. $\Longleftarrow$ *Supposons que* GIU(f) *soit fortement connexe. Soient (x, S) et (x', S') deux points de $\mathcal{X}_u$ et $\varepsilon > 0$. On construit la stratégie $\tilde{S}$ telle que la distance entre $(x, \tilde{S})$ et (x, S) est inférieure à ε et telle que les itérations parallèles de G_{f_u} depuis $(x, \tilde{S})$ mènent au point (x', S').*

Pour cela, on pose $t_1 = -\lfloor \log_{10}(\varepsilon) \rfloor$ et x'' la configuration de $\mathbb{B}^{\mathbb{N}}$ obtenue depuis (x, S) après t_1 itérations parallèles de G_{f_u}. Comme GIU(f) *est fortement connexe, il existe une stratégie S'' et un entier t_2 tels que x' est atteint depuis (x'', S'') après t_2 itérations de G_{f_u}.*

Considérons à présent la stratégie $\tilde{S} = (s_0, \dots, s_{t_1-1}, s''_0, \dots, s''_{t_2-1}, s'_0, s'_1, s'_2, s'_3 \dots)$. Il est évident que (x', s') est atteint depuis $(x, \tilde{S})$ après $t_1 + t_2$ itérations parallèles de G_{f_u}. Puisque $\tilde{s}_t = s_t$ pour $t < t_1$, grâce au choix de t_1, on a $d((x, S), (x, \tilde{S})) < \epsilon$. Par conséquent, G_{f_u} est transitive.

$\Longrightarrow$ *Démontrons la contraposée. Si* GIU(f) *n'est pas fortement connexe, alors il existe deux configurations x et x' telles qu'aucun chemin de* GIU(f) *ne mène de x à x'. Soient S et S' deux stratégies et $\varepsilon \in]0; 1[$. Alors, pour tout (x'', S'') tel que $d((x'', S''), (x, S)) < \varepsilon$ on a x'' qui est égal à x. Comme il n'existe aucun chemin de* GIU(f) *qui mène de x à x', les itérations de G_{f_u} à partir de $(x'', S'') = (x, S'')$ ne peuvent atteindre que des points (x''', S''') de $\mathcal{X}_u$ tels que $x''' \neq x'$, et donc ne peuvent pas atteindre (x', S'). On peut remarquer que, du fait que $x''' \neq x'$, elles n'atteignent que des points de $\mathcal{X}_u$ dont la distance à (x', S') est supérieure à 1. Pour tout entier naturel t, on a $G^t_{f_u}(x'', S'') \neq (x', S')$. Ainsi G_{f_u} n'est pas transitive et par contraposée, on a la démonstration souhaitée.*

Prouvons à présent le théorème suivant :

Théorème [17]. *$\mathcal{T} \subset \mathcal{R}$.*

Preuve. *Soit $f : \mathbb{B}^{\mathbb{N}} \to \mathbb{B}^{\mathbb{N}}$ telle que G_{f_u} est transitive (i.e. f appartient à $\mathcal{T}$). Soit $(x, S) \in \mathcal{X}_u$ et $\varepsilon > 0$. Pour prouver que f appartient à $\mathcal{R}$, il suffit de prouver qu'il existe une stratégie $\tilde{S}$ telle que la distance entre $(x, \tilde{S})$ et (x, S) est inférieure à ε et telle que $(x, \tilde{S})$ est un point périodique.*

Soit $t_1 = -\lfloor \log_{10}(\varepsilon) \rfloor$ et soit x' la configuration obtenue après t_1 itérations de G_{f_u} depuis (x, S). D'après la proposition précédente, GIU(f) *est fortement connexe. Ainsi, il existe une stratégie S' et un nombre $t_2 \in \mathbb{N}$ tels que x est atteint depuis (x', S') après t_2 itérations de G_{f_u}.*

Soit alors la stratégie $\tilde{S}$ qui alterne les t_1 premiers termes de S avec les t_2 premiers termes de S'. Ainsi $\tilde{S}$ est définie par

$$(s_0, \dots, s_{t_1-1}, s'_0, \dots, s'_{t_2-1}, s_0, \dots, s_{t_1-1}, s'_0, \dots, s'_{t_2-1}, s_0, \dots).$$

Il est évident que $(x, \tilde{S})$ s'obtient à partir de $(x, \tilde{S})$ après $t_1 + t_2$ itérations parallèles de G_{f_u}. Ainsi $(x, \tilde{S})$ est un point périodique. Puisque $\tilde{s}_t$ est égal à s_t pour $t < t_1$, d'après le choix de t_1, on a $d((x, S), (x, \tilde{S})) < \epsilon$.

On peut conclure que $\mathcal{C} = \mathcal{R} \cap \mathcal{T} = \mathcal{T}$. On a alors la caractérisation suivante :

Théorème [18]. *Soit $f : \mathbb{B}^{\mathsf{N}} \to \mathbb{B}^{\mathsf{N}}$. La fonction G_{f_u} est chaotique si et seulement si* GIU(f) *est fortement connexe.*

B.3/ PREUVE QUE d EST UNE DISTANCE SUR $\mathcal{X}_g$

Pour $S, S' \in \mathcal{P}(\{1, \dots, \mathsf{N}\})$, on définit

$$d_S(S, S') = \frac{9}{\mathsf{N}} \sum_{t \in \mathbb{N}} \frac{|S_t \Delta S'_t|}{10^{t+1}}.$$

Montrons que d_S est une distance sur $\mathcal{P}(\{1, \dots, \mathsf{N}\})$.

Soit S, S' et S'' trois parties de [N].

— De manière évidente, $d_s(S, S')$ est positive ou bien nulle si et seulement si S et S' sont égales.

— Comme la différence symétrique est commutative, la valeur de $d_S(S, S')$ est égale à celle de $d_S(S', S)$.

— On a enfin la succession d'éléments suivants :

$$\begin{aligned}
S \Delta S' &= (S \cap \overline{S'}) \cup (\overline{S} \cap S') \\
&= (S \cap \overline{S'} \cap S'') \cup (S \cap \overline{S'} \cap \overline{S''}) \cup (\overline{S} \cap S' \cap S'') \cup (\overline{S} \cap S' \cap \overline{S''}) \\
&\subseteq (S \cap \overline{S'} \cap S'') \cup (S \cap \overline{S'} \cap \overline{S''}) \cup (\overline{S} \cap S' \cap S'') \cup (\overline{S} \cap S' \cap \overline{S''}) \cup \\
&\quad (\overline{S} \cap \overline{S'} \cap S'') \cup (S \cap S' \cap \overline{S''}) \cup (\overline{S} \cap \overline{S'} \cap S'') \cup (S \cap S' \cap \overline{S''}) \\
&= (\overline{S'} \cap S'') \cup (S \cap \overline{S''}) \cup (\overline{S} \cap S'') \cup (S' \cap \overline{S''}) \\
&= (S \Delta S'') \cup (S'' \Delta S')
\end{aligned}$$

On en déduit ainsi que $|S \Delta S'| \leq |S \Delta S''| + |S'' \Delta S'|$ et donc que l'égalité triangulaire $d_S(S, S') \leq d_S(S, S'') + d_S(S'', S')$ est établie.

B.4/ CARACTÉRISATION DES FONCTIONS f RENDANT CHAOTIQUE G_{f_g} DANS $(\mathcal{X}_g, d)$

Commençons par caractériser l'ensemble $\mathcal{T}$ des fonctions transitives :

Théorème [19]. *G_{f_g} est transitive si et seulement si* GIG(f) *est fortement connexe.*

Preuve. $\Longleftarrow$ *Supposons que* GIG(f) *soit fortement connexe. Soient (x, S) et (x', S') deux points de $\mathcal{X}_g$ et $\varepsilon > 0$. On construit la stratégie $\tilde{S}$ telle que la distance entre $(x, \tilde{S})$ et (x, S) est inférieure à ε et telle que les itérations parallèles de G_{f_g} depuis $(x, \tilde{S})$ mènent au point (x', S').*

Pour cela, on pose $t_1 = -\lfloor \log_{10}(\varepsilon) \rfloor$ et x'' la configuration de $\mathbb{B}^{\mathsf{N}}$ obtenue depuis (x, S) après t_1 itérations parallèles de G_{f_g}. Comme GIG(f) *est fortement connexe, il existe une stratégie S'' et un entier t_2 tels que x' est atteint depuis (x'', S'') après t_2 itérations de G_{f_g}.*

Considérons à présent la stratégie $\tilde{S} = (s_0, \dots, s_{t_1-1}, s''_0, \dots, s''_{t_2-1}, s'_0, s'_1, s'_2, s'_3 \dots)$. Il est évident que (x', s') est atteint depuis $(x, \tilde{S})$ après $t_1 + t_2$ itérations parallèles de G_{f_g}. Puisque

$\tilde{s}_t = s_t$ pour $t < t_1$, grâce au choix de t_1, on a $d((x,S),(x,\tilde{S})) < \epsilon$. Par conséquent, G_{f_g} est transitive.

$\Longrightarrow$ Démontrons la contraposée. Si $\mathrm{GIG}(f)$ n'est pas fortement connexe, alors il existe deux configurations x et x' telles qu'aucun chemin de $\mathrm{GIG}(f)$ ne mène de x à x'. Soient S et S' deux stratégies et $\varepsilon \in]0;1[$. Alors, pour tout (x'',S'') tel que $d((x'',S''),(x,S)) < \varepsilon$ on a x'' qui est égal à x. Comme il n'existe aucun chemin de $\mathrm{GIG}(f)$ qui mène de x à x', les itérations de G_{f_g} à partir de $(x'',S'') = (x,S'')$ ne peuvent atteindre que des points (x''',S''') de $\mathcal{X}_g$ tels que $x''' \neq x'$, et donc ne peuvent pas atteindre (x',S'). On peut remarquer que, du fait que $x''' \neq x'$, elles n'atteignent que des points de $\mathcal{X}_g$ dont la distance à (x',S') est supérieure à 1. Pour tout entier naturel t, on a $G^t_{f_g}(x'',S'') \neq (x',S')$. Ainsi G_{f_g} n'est pas transitive et par contraposée, on a la démonstration souhaitée.

Prouvons à présent le théorème suivant :

Théorème 20. *$\mathcal{T} \subset \mathcal{R}$.*

Preuve. *Soit $f : \mathbb{B}^N \to \mathbb{B}^N$ telle que G_{f_g} est transitive (i.e. f appartient à $\mathcal{T}$). Soit $(x,S) \in \mathcal{X}_g$ et $\varepsilon > 0$. Pour prouver que f appartient à $\mathcal{R}$, il suffit de prouver qu'il existe une stratégie $\tilde{S}$ telle que la distance entre $(x,\tilde{S})$ et (x,S) est inférieure à ε et telle que $(x,\tilde{S})$ est un point périodique.*

Soit $t_1 = -\lfloor \log_{10}(\varepsilon) \rfloor$ et soit x' la configuration obtenue après t_1 itérations de G_{f_g} depuis (x,S). D'après la proposition précédente, $\mathrm{GIG}(f)$ est fortement connexe. Ainsi, il existe une stratégie S' et un nombre $t_2 \in \mathbb{N}$ tels que x est atteint depuis (x',S') après t_2 itérations de G_{f_g}.

Soit alors la stratégie $\tilde{S}$ qui alterne les t_1 premiers termes de S avec les t_2 premiers termes de S'. Ainsi $\tilde{S}$ est définie par

$$(s_0, \ldots, s_{t_1-1}, s'_0, \ldots, s'_{t_2-1}, s_0, \ldots, s_{t_1-1}, s'_0, \ldots, s'_{t_2-1}, s_0, \ldots).$$

Il est évident que $(x,\tilde{S})$ s'obtient à partir de $(x,\tilde{S})$ après $t_1 + t_2$ itérations parallèles de G_{f_g}. Ainsi $(x,\tilde{S})$ est un point périodique. Puisque $\tilde{s}_t$ est égal à s_t pour $t < t_1$, d'après le choix de t_1, on a $d((x,S),(x,\tilde{S})) < \epsilon$.

On peut conclure que $C = \mathcal{R} \cap \mathcal{T} = \mathcal{T}$. On a alors la caractérisation suivante :

Théorème 21. *Soit $f : \mathbb{B}^N \to \mathbb{B}^N$. La fonction G_{f_g} est chaotique si et seulement si $\mathrm{GIG}(f)$ est fortement connexe.*

B.5/ THÉORÈME 15

Soit $\alpha \in \mathbb{B}$. On nomme f^α la fonction de $\mathbb{B}^{N-1}$ dans lui-même définie pour chaque $x \in \mathbb{B}^{N-1}$ par

$$f^\alpha(x) = (f_1(x,\alpha), \ldots, f_{N-1}(x,\alpha)).$$

On nomme $\mathrm{GIU}(f)^\alpha$ le sous-graphe de $\mathrm{GIU}(f)$ engendré par le sous-ensemble $\mathbb{B}^{N-1} \times \{\alpha\}$ de $\mathbb{B}^N$.

Énonçons et prouvons tout d'abord les lemmes techniques suivants :

Lemme [5]. *$G(f^\alpha)$ est un sous-graphe de $G(f)$: chaque arc de $G(f^\alpha)$ est un arc de $G(f)$. De plus si $G(f)$ n'a pas d'arc de* N *vers un autre sommet $i \neq$ N, alors on déduit $G(f^\alpha)$ de $G(f)$ en supprimant le sommet* N *ainsi que tous les arcs dont* N *est soit l'extrémité, soit l'origine (et dans ce dernier cas, les arcs sont des boucles sur* N*).*

Preuve. *Supposons que $G(f^\alpha)$ possède un arc de j vers i de signe s. Par définition, il existe un sommet $x \in \mathbb{B}^{N-1}$ tel que $f_{ij}^\alpha(x) = s$, et puisque $f_{ij}^\alpha(x) = f_{ij}(x,\alpha)$, on en déduit que $G(f)$ possède un arc de j à i de signe s. Ceci prouve la première assertion. Pour démontrer la seconde, il suffit de prouver que si $G(f)$ a un arc de j vers i de signe s, avec $i, j \neq$ N, alors $G(f^\alpha)$ contient aussi cet arc. Ainsi, supposons que $G(f)$ a un arc de j vers i de signe s, avec $i, j \neq$ N. Alors, il existe $x \in \mathbb{B}^{N-1}$ et $\beta \in \mathbb{B}$ tels que $f_{ij}(x,\beta) = s$. Si $f_{ij}(x,\beta) \neq f_{ij}(x,\alpha)$, alors f_i dépend du $N^{ème}$ composant, ce qui est en contradiction avec les hypothèses. Ainsi $f_{ij}(x,\alpha)$ est égal à s. On a donc aussi $f_{ij}^\alpha(x) = s$. Ainsi $G(f^\alpha)$ possède un arc arc de j vers i de signe s.*

Lemme [6]. *Les graphes* GIU(f^α) *et* GIU$(f)^\alpha$ *sont isomorphes.*

Preuve. *Soit h la bijection de $\mathbb{B}^{N-1}$ vers $\mathbb{B}^{N-1} \times \{\alpha\}$ définie par $h(x) = (x,\alpha)$ pour chaque $x \in \mathbb{B}^{N-1}$. On voit facilement que h permet de définir un isomorphisme entre* GIU(f^α) *et* GIU$(f)^\alpha$ *: GIU(f^α) possède un arc de x vers y si et seulement si* GIU$(f)^\alpha$ *a un arc de $h(x)$ vers $h(y)$.*

Preuve. *du Théorème 15. La preuve se fait par induction sur* N*. Soit f une fonction de $\mathbb{B}^N$ dans lui-même et qui vérifie les hypothèses du théorème. Si* N $= 1$ *la démonstration est élémentaire : en raison du troisième point du théorème, $G(f)$ a une boucle négative ; ainsi $f(x) = \overline{x}$ et* GIU(f) *est un cycle de longueur 2. On suppose donc que* N > 1 *et que le théorème est valide pour toutes les fonctions de $\mathbb{B}^{N-1}$ dans lui-même. En raison du premier point du théorème, $G(f)$ contient au moins un sommet i tel qu'il n'existe pas dans $G(f)$ d'arc de i vers un autre sommet $j \neq i$. Sans perte de généralité, on peut considérer que ce sommet est* N*. Alors, d'après le lemme 5, f^0 et f^1 vérifient les conditions de l'hypothèse. Alors, par hypothèse d'induction* GIU(f^0) *et* GIU(f^1) *sont fortement connexes. Ainsi, d'après le lemme 6,* GIU$(f)^0$ *et* GIU$(f)^1$ *sont fortement connexes. Pour prouver que* GIU(f) *est fortement connexe, il suffit de prouver que* GIU(f) *contient un arc $x \to y$ avec $x_N = 0 < y_N$ et un arc $x \to y$ avec $x_N = 1 > y_N$. En d'autres mots, il suffit de prouver que :*

$$\forall \alpha \in \mathbb{B},\ \exists x \in \mathbb{B}^N, \qquad x_N = \alpha \neq f_N(x). \tag{$*$}$$

On suppose tout d'abord que N *a une boucle négative. Alors, d'après la définition de $G(f)$, il existe $x \in \mathbb{B}^N$ tel que $f_{NN}(x) < 0$. Ainsi si $x_N = 0$, on a $f_N(x) > f_N(\overline{x}^N)$, et donc $x_N = 0 \neq f_N(x)$ et $\overline{x}_N^N = 1 \neq f_N(\overline{x}^N)$; et si $x_N = 1$, on a $f_N(x) < f_N(\overline{x}^N)$, donc $x_N = 1 \neq f_N(x)$ et $\overline{x}_N^N = 0 \neq f_N(\overline{x}^N)$. Dans les deux cas, la condition $(*)$ est établie.*

Supposons maintenant que N *n'a pas de boucle négative. D'après la seconde hypothèse,* N *n'a pas de boucle,* i.e.*, la valeur de $f_N(x)$ ne dépend pas de la valeur de x_N. D'après la troisième hypothèse, il existe $i \in [\![1; N]\!]$ tel que $G(f)$ a un arc de i vers* N*. Ainsi, il existe $x \in \mathbb{B}^N$ tel que $f_{Ni}(x) \neq 0$ et donc f_N n'est pas constante. Ainsi, il existe $x, y \in \mathbb{B}^N$ tel que $f_N(x) = 1$ et $f_N(y) = 0$. Soit $x' = (x_1, \ldots, x_{N-1}, 0)$ et $y' = (y_1, \ldots, y_{N-1}, 1)$. Puisque la valeur de $f_N(x)$ (resp. de $f_N(y)$) ne dépend pas de la valeur de x_N (resp. de y_N), on a $f_N(x') = f_N(x) = 1 \neq x'_N$ (resp. $f_N(y') = f_N(y) = 0 \neq y'_N$). Ainsi la condition $(*)$ est établie, et le théorème est prouvé.*

BIBLIOGRAPHIE

[Abbas et al., 2005] Abbas, A., Bahi, J. M., Contassot-Vivier, S., and Salomon, M. (2005). Mixing synchronism / asynchronism in discrete-state discrete-time dynamic networks. In *4th Int. Conf. on Engineering Applications and Computational Algorithms, DCDIS'2005*, pages 524–529, Guelph, Canada. ISSN 1492-8760.

[Adler et al., 1965] Adler, R. L., Konheim, A. G., and McAndrew, M. H. (1965). Topological entropy. *Transactions of the American Mathematical Society*, 114 :309–319.

[Bahi, 2000] Bahi, J. M. (2000). Boolean totally asynchronous iterations. *International Journal of Mathematical Algorithms*, 1 :331–346.

[Bahi and Contassot-Vivier, 2002] Bahi, J. M. and Contassot-Vivier, S. (2002). Stability of fully asynchronous discrete-time discrete-state dynamic networks. *IEEE Transactions on Neural Networks*, 13(6) :1353–1363.

[Bahi et al., 2010] Bahi, J. M., Contassot-Vivier, S., and Couchot, J.-F. (2010). Convergence results of combining synchronism and asynchronism for discrete-state discrete-time dynamic network. Research Report RR2010-02, LIFC - Laboratoire d'Informatique de l'Université de Franche Comté.

[Bahi and Guyeux, 2010a] Bahi, J. M. and Guyeux, C. (2010a). Hash functions using chaotic iterations. *Journal of Algorithms & Computational Technology*, 4(2) :167–181.

[Bahi and Guyeux, 2010b] Bahi, J. M. and Guyeux, C. (2010b). A new chaos-based watermarking algorithm. In *SECRYPT'10, International Conference on Security and Cryptography*, pages 455–458, Athens, Greece. SciTePress.

[Bahi et al., 2011] Bahi, J. M., Guyeux, C., and Salomon, M. (2011). Building a chaotic proven neural network. In *ICCANS 2011, IEEE Int. Conf. on Computer Applications and Network Security*, Maldives, Maldives.

[Bahi and Michel, 1999] Bahi, J. M. and Michel, C. (1999). Simulations of asynchronous evolution of discrete systems. *Simulation Practice and Theory*, 7 :309–324.

[Banks et al., 1992] Banks, J., Brooks, J., Cairns, G., and Stacey, P. (1992). On devaney's definition of chaos. *Amer. Math. Monthly*, 99 :332–334.

[Beyer et al., 2007] Beyer, D., Henzinger, T. A., Jhala, R., and Majumdar, R. (2007). The software model checker blast. *International Journal on Software Tools for Technology Transfer*, 9(5-6) :505–525.

[Bowen, 1971] Bowen, R. (1971). Entropy for group endomorphisms and homogeneous spaces. *Transactions of the American Mathematical Society*, 153 :401–414.

[Chandrasekaran, 2006] Chandrasekaran, N. (2006). Verifying convergence of asynchronous iterative algorithms based on lyapunov functions.

[Cimatti et al., 2002] Cimatti, A., Clarke, E. M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. (2002). Nusmv 2 : An opensource tool for symbolic model checking. In Brinksma, E. and Larsen, K. G., editors, *CAV*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer.

[Couchot et al., 2005] Couchot, J.-F., Giorgetti, A., and Kosmatov, N. (2005). A uniform deductive approach for parameterized protocol safety. In Redmiles, D. F., Ellman, T., and Zisman, A., editors, *ASE*, pages 364–367. ACM.

[Crook et al., 2007] Crook, N., Goh, W. J., and Hawarat, M. (2007). Pattern recall in networks of chaotic neurons. *Biosystems*, 87(2-3) :267 – 274.

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2 :303–314.

[Dalkiran and Danisman, 2010] Dalkiran, I. and Danisman, K. (2010). Artificial neural network based chaotic generator for cryptology. *Turkish Journal of Electrical Engineering and Computer Sciences*, 18(2) :225–240.

[Devaney, 1989] Devaney, R. L. (1989). *An Introduction to Chaotic Dynamical Systems*. Redwood City : Addison-Wesley, 2nd edition.

[Fredlund and Svensson, 2007] Fredlund, L.-A. and Svensson, H. (2007). Mcerlang : a model checker for a distributed functional programming language. *SIGPLAN Not.*, 42 :125–136.

[Goles Ch. and Salinas, 2008] Goles Ch., E. and Salinas, L. (2008). Comparison between parallel and serial dynamics of boolean networks. *Theoretical Computer Science*, 396(1-3) :247–253.

[Gray, 1953] Gray, F. (1953). Pulse code communication. US Patent 2,632,058, March 17 1953,(filed November 13 1947).

[Guyeux, 2010] Guyeux, C. (2010). *Le désordre des itérations chaotiques et leur utilité en sécurité informatique*. PhD thesis, Université de Franche-Comté.

[Guyeux et al., 2010] Guyeux, C., Friot, N., and Bahi, J. M. (2010). Chaotic iterations versus spread-spectrum : chaos and stego security. In *IIH-MSP'10, 6-th International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pages 208–211, Darmstadt, Germany.

[Holoska et al., 2010] Holoska, J., Oplatkova, Z., Zelinka, I., and Senkerik, R. (2010). Comparison between neural network steganalysis and linear classification method stegdetect. *Computational Intelligence, Modelling and Simulation, International Conference on.*, 0 :15–20.

[Holzmann, 2003] Holzmann, G. J. (2003). *The SPIN Model Checker : Primer and Reference Manual*. Addison-Wesley, Pearson Education.

[Hornik et al., 1989] Hornik, K., Stinchcombe, M. B., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5) :359–366.

[Li et al., 2010a] Li, Y., Deng, S., and Xiao, D. (2010a). A novel hash algorithm construction based on chaotic neural network. *Neural Computing and Applications*, pages 1–9.

[Li et al., 2010b] Li, Y., Deng, S., and Xiao, D. (2010b). A novel hash algorithm construction based on chaotic neural network. *Neural Computing and Applications*, pages 1–9.

[Lian, 2009] Lian, S. (2009). A block cipher based on chaotic neural networks. *Neurocomputing*, 72(4-6) :1296 – 1301.

[Qiao et al., 2009] Qiao, M., Sung, A. H., and Liu, Q. (2009). Steganalysis of mp3stego. In *Proceedings of the 2009 international joint conference on Neural Networks*, IJCNN'09, pages 2723–2728. IEEE Press.

[Richard and Comet, 2007] Richard, A. and Comet, J.-P. (2007). Necessary conditions for multistationarity in discrete dynamical systems. *Discrete Applied Mathematics*, 155(18) :2403–2413.

[Robby et al., 2003] Robby, Dwyer, M. B., and Hatcliff, J. (2003). Bogor : an extensible and highly-modular software model checking framework. In *Proc. of the 11th ACM SIGSOFT Symposium on Foundations of Software Engineering 2003*, pages 267–276. ACM.

[Robert, 1995] Robert, F. (1995). *Les systèmes dynamiques discrets*, volume 19 of *Mathématiques et Applications*. Springer.

[Satish et al., 2004] Satish, K., Jayakar, T., Tobin, C., Madhavi, K., and Murali, K. (2004). Chaos based spread spectrum image steganography. *Consumer Electronics, IEEE Transactions on*, 50(2) :587 – 590.

[Shaohui et al., 2003] Shaohui, L., Hongxun, Y., and Wen, G. (2003). Neural network based steganalysis in still images. *Multimedia and Expo, IEEE International Conference on*, 2 :509–512.

[Sullivan et al., 2006] Sullivan, K., Madhow, U., Chandrasekaran, S., and Manjunath, B. S. (2006). Steganalysis for markov cover data with applications to images. *IEEE Transactions on Information Forensics and Security*, 1 :275–287.

[Weise, 1997] Weise, C. (1997). An incremental formal semantics for PROMELA. In *SPIN97, the Third SPIN Workshop*.

[Zhang et al., 2005] Zhang, L., Liao, X., and Wang, X. (2005). An image encryption approach based on chaotic maps. *Chaos, Solitons & Fractals*, 24(3) :759 – 765.

TABLE DES FIGURES

LISTE DES TABLES

LISTE DES DÉFINITIONS

Résumé :

Blabla blabla.

SPIM

■ École doctorale SPIM 16 route de Gray F - 25030 Besançon cedex

■ tél. +33 (0)3 81 66 66 02 ■ **ed-spim@univ-fcomte.fr** ■ www.ed-spim.univ-fcomte.fr

UFC
UNIVERSITÉ
DE FRANCHE-COMTÉ