SPIM

Habilitation à Dirigier des Recherches

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

UNIVERSITÉ DE FRANCHE-COMTÉ

Title

FIRST NAME

Habilitation à Diriger des Recherches

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

Système Booléens

1

ITERATIONS DISCRÈTES DE SYSTÈMES DYNAMIQUES BOOLÉENS

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 $\mathbb{B}^n$. L'ensemble $\{1, \ldots, n\}$ sera par la suite noté $[n]$. Le $i^{\text{ème}}$ composant d'un élément $x \in \mathbb{B}^n$ s'écrit x_i. Si l'ensemble I est une partie de $[n]$, alors $\overline{x}^I$ est l'élément $y \in \mathbb{B}^n$ 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 $\overline{x}^{[n]}$ (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 $\mathbb{B}^n$, l'ensemble $\Delta(x, y)$, contient les $i \in [n]$ tels que $x_i \neq y_i$. Soit enfin $f : \mathbb{B}^n \to \mathbb{B}^n$. Son $i^{\text{ème}}$ composant est nommé f_i qui est une fonction de $\mathbb{B}^n$ dans $\mathbb{B}$. Pour chaque x dans $\mathbb{B}^n$, 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 $n = 3$ et $f : \mathbb{B}^3 \to \mathbb{B}^3$ telle que $f(x) = (f_1(x), f_2(x), f_3(x))$ avec*

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

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

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 :

$$f : \mathbb{B}^n \to \mathbb{B}^n, \qquad x = (x_1, \ldots, x_n) \mapsto f(x) = (f_1(x), \ldots, f_n(x)),$$

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 $(x_1, x_2, x_3) \mapsto ((\overline{x_1} + \overline{x_2}).x_3, x_1.x_3, x_1 + x_2 + x_3)$

et un *schéma itératif* ou encore *mode de mise à jour*. À partir d'une configuration initiale $x^0 \in \mathbb{B}^n$, 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, $1 \leq i \leq n$, 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, $1 \leq i \leq n$, 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 $i \in [n]$ par

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

— **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 $\{k\}$, $1 \leq k \leq n$, 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 $\{1, \ldots, n\}$ 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 $s^t \in \mathcal{P}([n])$ 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 $i \in [n]$ par

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

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 Γ un graphe d'itérations (synchrones, unaires ou généralisées) de f. Les *attracteurs* de Γ sont les plus petits sous-ensembles (au sens de l'inclusion) non vides $A \subseteq \mathbb{B}^n$ tels que pour tout arc $x \to y$ de Γ, 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 de Γ. En d'autres termes, les attracteurs non cycliques de Γ 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 Γ 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

$$f'(x) = (f'_{ij}(x)), \qquad f'_{ij}(x) = \frac{f_i(\overline{x}^j) - f_i(x)}{\overline{x_j} - x_j} \qquad (i, j \in [n]). \tag{1.1}$$

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 $n \times n$. 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 $x \in [n]$, $f_i(\overline{x}^j)$ 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 sommets est $[n]$ 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 $x \in \mathbb{B}^n$.

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 (1.1). 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,... 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 $[n]$, sont jugées intéressantes celles qui activent au moins une fois chacun des $i \in [n]$. 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 $[n]$ si tout indice de $[n]$ s'y retrouve au moins une fois.

Parmi toutes les stratégies unaires de $[n]^{\mathbb{N}}$, 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

$$s^t = 24 \text{ si } t \text{ est pair et } s^t = 15 \text{ sinon} \tag{1.2}$$

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 $x^0 = (x^0_1, \dots, x^0_n)$ une configuration initiale. Soit $(D^t)^{t\in\mathbb{N}}$ la suite de matrice de taille $n \times n$ dont chaque élément D^t_{ij} 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 $\delta^t_{ij} = t - D^t_{ij}$ est borné par une constante δ_0 pour tous les i, j. Le *mode des itérations généralisées asynchrone* est défini pour chaque $i \in \{1, \dots, n\}$ et chaque $t = 0, 1, 2, ...$ par :

$$x_i^{t+1} = \begin{cases} f_i(x_1^{D_{i1}^t}, \ldots, x_n^{D_{in}^t}) \text{ si } \mathit{bin}(s^t)[i] = 1 \\ x_i^t \text{ sinon} \end{cases} \tag{1.3}$$

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 :

$$\exists t_0 \,.\, (\forall t \,.\, t \geq t_0 \Rightarrow x^t = x^{t_0}). \tag{1.4}$$

Sinon les itérations sont dites *divergentes*. De plus, si $(x^{(t)})^{t\in\mathbb{N}}$ défini selon l'équation (1.3) satisfait (1.4) 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 (1.2) : 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 (1.3) 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

$$T(Sync) = I \times (\max_i cp_i + \max_{i,j}(B_{ji} \times (\beta_{ij} + cs_i \times \tau_{ij}))) \quad (1.5)$$

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.

$$I = I_{\langle 1 \rangle} + I_{\langle 3 \rangle} + I_{\langle 4 \rangle} = 2 + 1 + 2 = 5 \quad (1.6)$$

1.2.4/ Le mode mixe

On considère $|\mathcal{K}|$ classes de composants synchronisés. (comme donné en équation (1.6)). 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

$$T(Mixed) \geq \sum_{k \in \mathcal{K}} I_k(\max_{l \in k} cp_l) \quad (1.7)$$

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 :

$$T(Mixed) \leq \sum_{k \in \mathcal{K}} \left(I_k \times (\max_{l \in k} cp_l) + \max_{l \in k, e \in k', k \leq k'} B_{el} \times (\beta_{le} + cs_l \tau_{le}) \right) \quad (1.8)$$

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 :

$$T(Async) \geq \max_{i=1}^{n} I_i \times cp_i \tag{1.9}$$

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 :

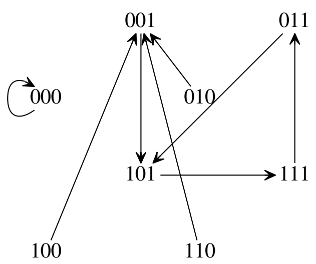
$$T(Async) \leq \sum_{i=1}^{n} \left(I_i \times cp_i + \max_{1 \leq k \leq n} B_{ki}(\beta_{ik} + cs_i \tau_{ik}) \right) \tag{1.10}$$

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

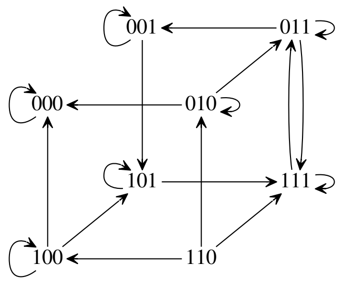
Exemple. *La* Figure *1.7 présente un exemple d'exécution du mode généralisé asynchrone. Certaines communications issues de l'élément* 4 *n'ont pas été représentées pour des raisons de clarté. On constate que le temps global de convergence est plus petit que celui des deux autres modes.*

1.3/ Conclusion

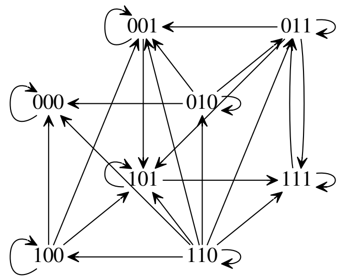
Introduire de l'asynchronisme peut permettre de réduire le temps d'exécution global, mais peut aussi introduire de la divergence. Dans ce chapitre, nous avons exposé comment construire un mode combinant les avantage du synchronisme en terme de convergence avec les avantages de l'asynchronisme en terme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



(a) GIS(f)



(b) GIU(f)

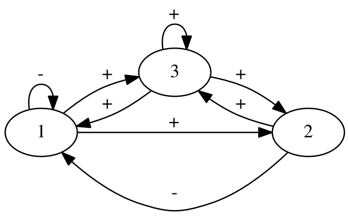


(c) GIG(f)

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 $(x_1, x_2, x_3) \mapsto ((\overline{x_1} + \overline{x_2}).x_3, x_1.x_3, x_1 + x_2 + x_3)$

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

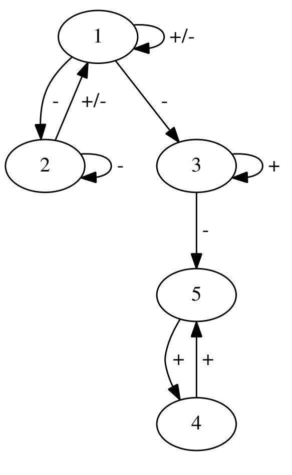
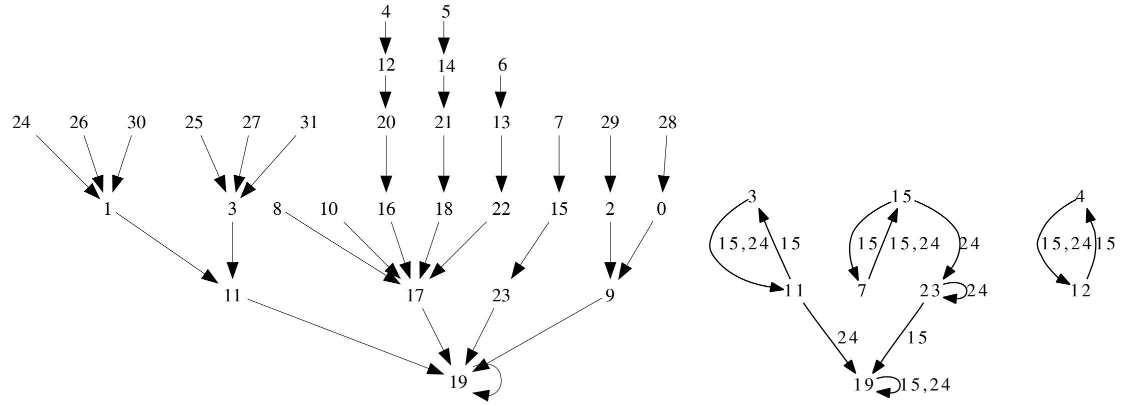


FIGURE 1.3 – Définition de $f : \mathbb{B}^5 \to \mathbb{B}^5$ et son graphe d'interaction



(a) Itérations synchrones de f.

(b) Extrait des itérations unaires.

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

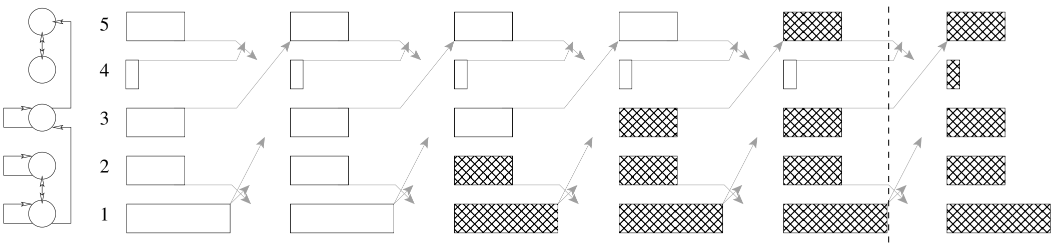


FIGURE 1.5 – Itérations synchrones

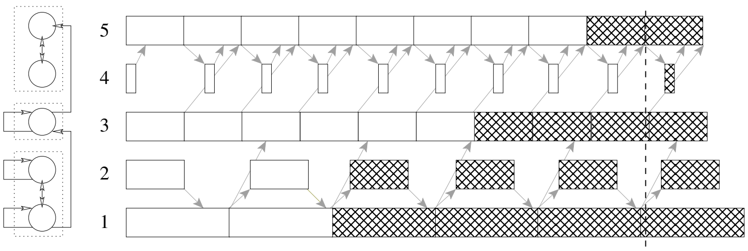


FIGURE 1.6 – Itérations mixes avec $\langle 1 \rangle = \{1, 2\}$, $\langle 3 \rangle = \{3\}$, $\langle 4 \rangle = \{4, 5\}$.

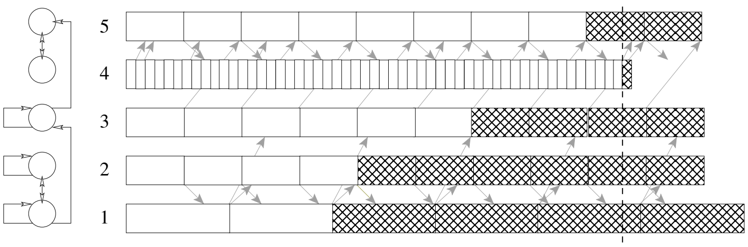


FIGURE 1.7 – Itérations asynchrones

2

Preuve automatique de convergence de systèmes booléens

donner dans les rappels les délais et les propriétés de convergence universelle

Statuer sur la taille des exemples traitables par la démarche, cf données pratiques

2.1/ Exemple jouet

Exemple. *On considère dans ce chapitre l'exemple où trois éléments dans $\mathbb{B}$. Chaque configuration est ainsi un élement de $\{0, 1\}^3$, i.e., un nombre entre 0 et 7. La* Figure *2.1 précise la fonction f considérée et la* Figure *2.2 donne son graphe d'intéraction.*

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

Figure 2.1 – Fonction à itérer

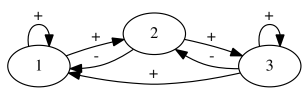


Figure 2.2 – Graphe d'intéraction

Figure 2.3 – Exemple pour SDD ≈ SPIN.

On peut facilement vérifier que toutes les itérations parallèles initialisées avec $x^0 \neq 7$ soit (111) convergent vers 2 soit (010) ; celles initialisées avec $x^0 = 7$ restent en 7. Pour les autres modes synchrones avec une stratégie pseudo périodique, les comportements selon 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.*

2.2/ 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 2.4 – Declaration des types de la traduction.

Les types primaires de PROMELA sont `bool`, `byte`, `short` et `int`. Comme dans le langage C par exemple, on peut déclarer des tableaux à une dimension de taille constante ou des nouveaux types de données (introduites par le mot clef `typedef`). Ces derniers sont utilisés pour définir des tableaux à deux dimensions.

Exemple. *Le programme donné à la* FIGURE *2.4 correspond à des déclarations de variables qui serviront dans l'exemple jouet de ce chapitre. Il définit tout d'abord :*

— *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 (le décalages d'un entier est dû à l'indexation à partir de zéro des cellules d'un tableau) ; 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 $x_{j+1}^{D^{t-1}_{i+1\,j+1}}$ pour l'itération au temps t (en d'autres termes, utile lors du calcul de x^t).*

Puisque le décalage d'un indices ne change pas fondamentalement le comportement de la version PROMELA par rapport au modèle initial et pour des raisons de clarté, on utilisera par la suite la même lettre d'indice entre les deux niveaux (pour le modèle : x_i et pour PROMELA : `X[i]`*). Cependant, ce décalage devra être conservé mémoire.*

Une donnée de type `channel` *permet le transfert de messages entre processus dans un ordre FIFO. Elles serait déclarée avec le mot clef* `chan` *suivi par sa capacité (qui est constante), son nom et le type des messages qui sont stockés dans ce canal. 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

```
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 2.5 – 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 2.6 – 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 2.7 – Codage du graphe d'intéraction de f.

sein de systèmes. Un process est déclaré avec le mot-clé `proctype` et 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 2.5, 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 2.7) 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]`.

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

FIGURE 2.8 – 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 2.9 – 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 2.10 – Application de la fonction f.

2.3/ DU SYSTÈME BOOLÉEN AU MODÈLE PROMELA

Les éléments principaux des itérations asynchrones rappelées à l'équation (1.3) 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.

2.3.1/ LA STRATÉGIE

Regardons comment une stratégie pseudo périodique peut être représentée en PROMELA. Intuitivement, un process `scheduler` (comme représenté à la FIGURE 2.6) 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`.

2.3.2/ ITÉRER LA FONCTION f

La mise à jour de l'ensemble $s^t = \{s_1, \dots, s_m\}$ 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 2.9. Ce process 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 2.8
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 ;

```
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 2.11 – 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 2.12 – Diffuser les valeurs des elements

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 2.10 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`).

2.3.3/ 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 2.11 et 2.12, 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 ;
  — 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.

2.3.4/ 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 :

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

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.

2.4/ 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.*

2.5/ 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. Donner un ordre de grandeur de cet ordre de grandeur

La méthode détaillée ici a pu être appliquée sur l'exemple jouet 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*faire référence à un tel exemple*.

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 2.14). 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 ($\perp$) sinon. Le nombre M est la taille de la mémoire consommée

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

FIGURE 2.13 – Expérimentations avec des itérations synchrones

	Mixed Mode						Only Bounded					
	Parallel			Pseudo-Periodic			Parallel			Pseudo-Periodic		
	P	M	T	P	M	T	P	M	T	P	M	T
RE	$\top$	409	1m11s	$\perp$	370	0.54	$\perp$	374	7.7s	$\perp$	370	0.51s
AC2D	$\perp$	2.5	0.001s	$\perp$	2.5	0.01s	$\perp$	2.5	0.01s	$\perp$	2.5	0.01s
[Bahi and Michel, 1999]	$\top$			$\top$			$\perp$			$\perp$		

FIGURE 2.14 – Expérimentations avec des itérations asynchrones

(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.

L'exemple *RE* est l'exemple jouet 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 leur valeur booléennes dans un graphe d'adjacence proche du graphe complet.

L'exemple jouet *RE* a été prouvé comme universellement convergent. *statuer sur AC2D* Comme la convergence n'est déjà pas établie pour les itérations parallèles de [Richard and Comet, 2007], il en est donc de même pour les itérations asynchrones. La FIGURE 2.15 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.

2.6/ CONCLUSION

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.

Déplacer ceci dans les perspective 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 $\lim\limits_{t \to \infty} s_{ij}^t = +\infty$. 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].

Mixage

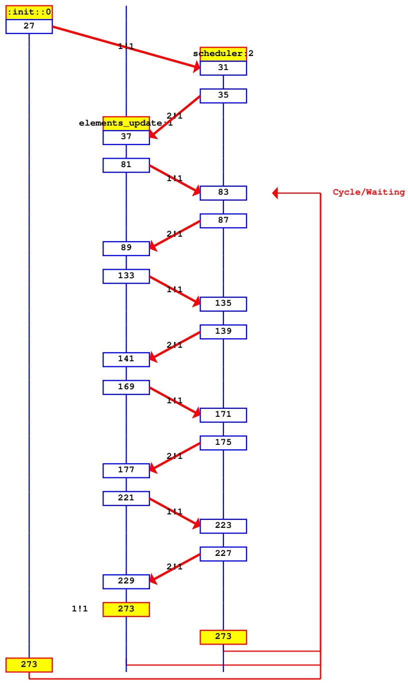


FIGURE 2.15 – Contre exemple de convergence pour 2.15

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 $\langle b_1 \rangle, \dots, \langle b_k \rangle$ en un temps t_k. Par construction, la classe $\langle b_{k+1} \rangle$ dépend uniquement de certaines classes de $\langle b_1 \rangle, \dots, \langle b_k \rangle$ 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 $p_j \in \langle b_j \rangle$, $1 \le j \le k$.*

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 $\langle b_j \rangle$, $1 \le j \le k$, 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 2.4

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

Lemme 2 (Strategy Equivalence). *Let ϕ be a DDN with strategy $(S^t)^{t \in \mathbb{N}}$ and ψ be its translation. There exists an execution of ψ with weak fairness s.t. the scheduler makes* `update_elems` *update elements of S^t at iteration t.*

Preuve. *The proof is direct for $t = 0$. Let us suppose it is established until t is some t_0. Let us consider pseudo-periodic strategies. Thanks to the weak fairness equity property,* `update_elems` *will modify elements of S^t at iteration t.*

In what follows, let Xd^t_{ji} be the value of `Xd[`j`].v[`i`]` after the t^{th} call to the function `fetch_values`. Furthermore, let Y^k_{ij} be the element at index k in the channel `channels[i].sent[j]` of size m, $m \le \delta_0$; Y^0_{ij} and Y^{m-1}_{ij} are respectively the head and the tail of the channel. Secondly, let $(M^t_{ij})^{t \in \{1, 1.5, 2, 2.5, \dots\}}$ be a sequence such that M^t_{ij} is the partial function that associates to each k, $0 \le k \le m-1$, the tuple $(Y^k_{ij}, a^k_{ij}, c^k_{ij})$ while entering into the `update_elems` at iteration t where Y^k_{ij} is the value of the channel `channels[i].sent[j]` at index k, a^k_{ij} is the date (previous to t) when Y^k_{ij} has been added and c^k_{ij} is the first date at which the value is available on j. So, the value is removed from the channel $i \to j$ at date $c^k_{ij} + 1$. M^t_{ij} has the following signature :

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

Intuitively, M^t_{ij} is the memory of `channels[i].sent[j]` while starting the iteration t. Notice that the domain of any M^1_{ij} is $\{0\}$ and $M^1_{ij}(0) = ($`Xp[i]`$, 0, 0)$: indeed, the `init` process initializes `channels[i].sent[j]` with `Xp[i]`.

Let us show how to make the indeterminism inside the two functions `fetch_values` and `diffuse_values` compliant with (1.3). The function M_{ij}^{t+1} is obtained by the successive updates of M_{ij}^{t} through the two functions `fetch_values` and `diffuse_values`. Abusively, let $M_{ij}^{t+1/2}$ be the value of M_{ij}^{t} after the former function during iteration t.

In what follows, we consider elements i and j both in $[\![1,n]\!]$ that are updated. At iteration t, $t \geq 1$, let $(Y_{ij}^0, a_{ij}^0, c_{ij}^0)$ be the value of $M_{ij}^t(0)$ at the beginning of `fetch_values`. If t is equal to $c_{ij}^0 + 1$ then we execute the instruction that assigns Y_{ij}^0 (*i.e.*, the head value of `channels[i].sent[j]`) to Xd_{ji}^t. In that case, the function M_{ij}^t is updated as follows : $M_{ij}^{t+1/2}(k) = M_{ij}^t(k+1)$ for each k, $0 \leq k \leq m-2$ and $m-1$ is removed from the domain of $M_{ij}^{t+1/2}$. Otherwise (*i.e.*, when $t < c_{ij}^0 + 1$ or when the domain of M_{ij} is empty) the `skip` statement is executed and $M_{ij}^{t+1/2} = M_{ij}^t$.

In the function `diffuse_values`, if there exists some τ, $\tau \geq t$ such that $D_{ji}^{\tau} = t$, let c_{ij} be defined by $\min\{l \mid D_{ji}^l = t\}$. In that case, we execute the instruction that adds the value `Xp[i]` to the tail of `channels[i].sent[j]`. Then, M_{ij}^{t+1} is defined as an extension of $M_{ij}^{t+1/2}$ in m such that $M_{ij}^{t+1}(m)$ is $(\mathtt{Xp[i]}, t, c_{ij})$. Otherwise (*i.e.*, when $\forall l\,.\,l \geq t \Rightarrow D_{ji}^l \neq t$ is established) the `skip` statement is executed and $M_{ij}^{t+1} = M_{ij}^{t+1/2}$.

Lemme 3 (Existence of SPIN Execution)**.** *For any sequences $(S^t)^{t\in\mathbb{N}}$, $(D^t)^{t\in\mathbb{N}}$, for any map F there exists a SPIN execution such that for any iteration t, $t \geq 1$, for any i and j in $[\![1,n]\!]$ we have the following properties :*

If the domain of M_{ij}^t is not empty, then

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

Secondly we have :

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

Thirdly, for any $k \in S^t$. Then, the value of the computed variable `Xp[k]` at the end of the `update_elems` process is equal to X_k^t i.e., $F_k\left(X_1^{D_{k1}^{t-1}}, \ldots, X_n^{D_{kn}^{t-1}}\right)$ at the end of the t^{th} iteration.

Preuve. *The proof is done by induction on the number of iterations.*

Initial case : *For the first item, by definition of M_{ij}^t, we have $M_{ij}^1(0) = (\mathtt{Xp[i]}, 0, 0)$ that is obviously equal to $\left(X_i^{D_{ji}^0}, 0, 0\right)$.*

Next, the first call to the function `fetch_value` either assigns the head of `channels[i].sent[j]` to `Xd[j].v[i]` or does not modify `Xd[j].v[i]`. Thanks to the `init` process, both cases are equal to `Xp[i]`, i.e., X_i^0. The equation (A.2) is then established.

For the last item, let k, $0 \leq k \leq n-1$. At the end of the first execution of the `update_elems` process, the value of `Xp[k]` is $F(\mathtt{Xd}[k]\mathtt{.v[0]}, \ldots, \mathtt{Xd}[k]\mathtt{.v}[n-1])$. Thus, by definition of Xd, it is equal to $F(Xd_{k0}^1, \ldots, Xd_{kn-1}^1)$. Thanks to (A.2), we can conclude the proof.

Inductive case : *Suppose now that lemma 3 is established until iteration l.*

First, if domain of definition of the function M_{ij}^{l} is not empty, by induction hypothesis $M_{ij}^{l}(0)$ is $\left(X_i^{D_{ji}^{c}}, D_{ji}^{c}, c\right)$ where c is $\min\{k|D_{ji}^{k} > D_{ji}^{l-2}\}$.

At iteration l, if $l < c+1$ then the `skip` statement is executed in the `fetch_values` function. Thus, $M_{ij}^{l+1}(0)$ is equal to $M_{ij}^{l}(0)$. Since $c > l-1$ then $D_{ji}^{c} > D_{ji}^{l-1}$ and hence, c is $\min\{k|D_{ji}^{k} > D_{ji}^{l-1}\}$. Obviously, this implies also that $D_{ji}^{c} > D_{ji}^{l-2}$ and $c = \min\{k|D_{ji}^{k} > D_{ji}^{l-2}\}$.

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

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

$$\forall t, k \,.\, D_{ji}^{c} < t < l \wedge k \geq t \Rightarrow D_{ji}^{k} \neq t.$$

  Particularly, $D_{ji}^{c'} \notin \{D_{ji}^{c} + 1, \ldots, l-1\}$. We can apply the third item of the induction hypothesis to deduce $\texttt{Xp[i]} = X_i^{D_{ji}^{c'}}$ and we can conclude.
- *if $\{0,1\} \subseteq$ dom(M_{ij}^{l}) then $M_{ij}^{l+1}(0)$ is $M_{ij}^{l}(1)$. Let $M_{ij}^{l}(1) = \left(\texttt{Xp[i]}, a_{ij}, c_{ij}\right)$. By construction a_{ij} is $\min\{t'|t' > D_{ji}^{c} \wedge (\exists k \,.\, k \geq t' \wedge D_{ji}^{k} = t')\}$ and c_{ij} is $\min\{k|D_{ji}^{k} = a_{ij}\}$. Let us show c_{ij} is equal to $\min\{k|D_{ji}^{k} > D_{ji}^{l-1}\}$ further referred as c'. First we have $D_{ji}^{c_{ij}} = a_{ij} > D_{ji}^{c}$. Since c by definition is greater or equal to $l-1$, then $D_{ji}^{c_{ij}} > D_{ji}^{l-1}$ and then $c_{ij} \geq c'$. Next, since c is $l-1$, c' is $\min\{k|D_{ji}^{k} > D_{ji}^{c}\}$ and then $a_{ij} \leq D_{ji}^{c'}$. Thus, $c_{ij} \leq c'$ and we can conclude as in the previous part.*

The case where the domain dom(M_{ij}^{l}) is empty but the formula $\exists k \,.\, k \geq l \wedge D_{ji}^{k} = l$ 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 $\min\{k|D_{ji}^{k} > D_{ji}^{l-1}\}$. Two cases have to be considered depending on whether D_{ji}^{l} and D_{ji}^{l-1} are equal or not.

- *If $D_{ji}^{l} = D_{ji}^{l-1}$, since $D_{ji}^{c'} > D_{ji}^{l-1}$, then $D_{ji}^{c'} > D_{ji}^{l}$ and then c' is distinct from l. Thus, the SPIN execution detailed above does not modify Xd_{ji}^{l+1}. It is obvious to establish that $Xd_{ji}^{l+1} = Xd_{ji}^{l} = X_i^{D_{ji}^{l-1}} = X_i^{D_{ji}^{l}}$.*
- *Otherwise D_{ji}^{l} is greater than D_{ji}^{l-1} and c is thus l. According to (A.1) we have proved, we have $M_{ij}^{l+1}(0) = (X_i^{D_{ji}^{l}}, D_{ji}^{l}, l)$. Then the SPIN execution detailed above assigns $X_i^{D_{ji}^{l}}$ to Xd_{ji}^{l+1}, 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 $\texttt{Xp[}k\texttt{]} = F(\texttt{Xd[}k\texttt{][0]}, \ldots, \texttt{Xd[}k\texttt{][}n\texttt{-1])+}$. By definition of Xd, it is equal to $F(Xd_{k0}^{l+1}, \ldots, Xd_{kn-1}^{l+1})$. 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 $(\mathcal{X}, d)$

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 ($s^t_0 = s_0$), donc les configurations de $F_f(s^t_0, x^t)$ 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/ THÉORÈME ??

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 $\Gamma(f)^\alpha$ le sous-graphe de $\Gamma(f)$ 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 $f^\alpha_{ij}(x) = s$, et puisque $f^\alpha_{ij}(x) = f_{ij}(x,\alpha)$, 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 $f^\alpha_{ij}(x) = s$. Ainsi $G(f^\alpha)$ possède un arc arc de j vers i de signe s.*

Lemme 6. *Les graphes $\Gamma(f^\alpha)$ et $\Gamma(f)^\alpha$ 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 $\Gamma(f^\alpha)$ et $\Gamma(f)^\alpha$: $\Gamma(f^\alpha)$ possède un arc de x vers y si et seulement si $\Gamma(f)^\alpha$ a un arc de $h(x)$ vers $h(y)$.*

Preuve. *du Théorème* **??**. *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 $\Gamma(f)$ 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 $\Gamma(f^0)$ et $\Gamma(f^1)$ sont fortement connexes. Ainsi, d'après le lemme 6, $\Gamma(f)^0$ et $\Gamma(f)^1$ sont fortement connexes. Pour prouver que $\Gamma(f)$ est fortement connexe, il suffit de prouver que $\Gamma(f)$ 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.

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

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

[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.

[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.

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

[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.

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

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

Table des figures

Liste des tables

LISTE DES DÉFINITIONS

Résumé :

Blabla blabla.

■ 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

UNIVERSITÉ DE FRANCHE-COMTÉ