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

SYSTÈME BOOLÉENS

1

ITERATIONS DISCRÈTES DE SYSTÈMES DYNAMIQUES BOOLÉENS

Chapeau chapitre à faire

Chapeau chapitre à faire

Cette section énonce quelques notions suffisantes à la compréhension de ce document. Elle commence par formaliser ce que sont les systèmes dynamiques booléens (section 1.1) et montre comment en extraire leur graphe d'itérations (section 1.2) et d'interactions (section 1.3). Elle se termine en définissant une distance sur l'espace $[\![1;n]\!]^{\mathbb{N}} \times \mathbb{B}^n$ (section 1.4).

1.1/ SYSTÈME DYNAMIQUE BOOLÉEN

Soit n un entier naturel. Un système dynamique booléen est défini à partir d'une fonction booléenne :

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

et un *schéma des itérations* qui peuvent être parallèles, séquentielles, chaotiques, asynchrones ... Le schéma des itérations parallèles est défini comme suit : 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 à partir de 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. Le schéma qui ne modifie qu'un élément i, $1 \leq i \leq n$ à chaque itération est le schéma *chaotique*. Plus formellement, à la $t^{\text{ème}}$ itération, seul le $s_t^{\text{ème}}$ composant (entre 1 et n) est mis à jour. La suite $s = (s_t)_{t\in\mathbb{N}}$ est une séquence d'indices de $[\![1;n]\!]$ appelée *stratégie*. Formellement, dans ce mode opératoire, soit $F_f : [\![1;n]\!] \times \mathbb{B}^n$ vers $\mathbb{B}^n$ définie par

$$F_f(i, x) = (x_1, \dots, x_{i-1}, f_i(x), x_{i+1}, \dots, x_n).$$

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

$$x^{t+1} = F_f(s_t, x^t). \tag{1.1}$$

Soit alors G_f une fonction de $[\![1;n]\!]^{\mathbb{N}} \times \mathbb{B}^n$ dans lui même définie par

$$G_f(s, x) = (\sigma(s), F_f(s_0, x)),$$

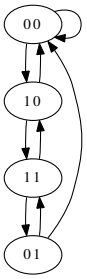

FIGURE 1.1 – $g(x_1, x_2) = (\overline{x_1}, x_1\overline{x_2})$

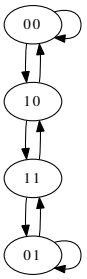

FIGURE 1.2 – $h(x_1, x_2) = (\overline{x_1}, x_1\overline{x_2} + \overline{x_1}x_2)$

FIGURE 1.3 – Graphes d'itérations de fonctions booléennes dans $\mathbb{B}^2$

où $\forall t \in \mathbb{N}, \sigma(s)_t = s_{t+1}$. En d'autres termes la fonction σ décale la stratégie fournie en argument d'un élément vers la gauche en supprimant l'élément de tête. Les itérations parallèles de G_f depuis un point initial $X^0 = (s, x^0)$ décrivent la même orbite que les itérations chaotiques de f induites par x^0 et la stratégie s.

1.2/ GRAPHE D'ITÉRATIONS

Soit f une fonction de $\mathbb{B}^n$ dans lui-même. Le *graphe des itérations chaotiques* associé à f est le graphe orienté $\Gamma(f)$ défini ainsi : l'ensemble de ses sommets est $\mathbb{B}^n$ et pour chaque $x \in \mathbb{B}^n$ et $i \in [\![1; n]\!]$, le graphe $\Gamma(f)$ contient un arc de x vers $F_f(i, x)$. La relation entre $\Gamma(f)$ et G_f est claire : il existe un chemin de x à x' dans $\Gamma(f)$ si et seulement s'il existe une stratégie s telle que les itérations parallèles G_f à partir du point (s, x) mènent au point x'.

Dans ce qui suit, et par souci de concision, le terme *graphe des itérations* est une abréviation de graphe des itérations chaotiques. La figure 1.3 donne deux exemples de graphes d'itérations pour les fonctions g et h définies dans $\mathbb{B}^2$ qui seront reprises dans la suite.

1.3/ GRAPHE D'INTERACTIONS

Pour $x \in \mathbb{B}^n$ et $i \in [\![1; n]\!]$, on nomme $\overline{x}^i$ la configuration obtenue en niant le $i^{\text{ème}}$ composant de x. En d'autres termes $\overline{x}^i = (x_1, \ldots, \overline{x_i}, \ldots, x_n)$. Des 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 $n \times n$

$$f'(x) = (f_{ij}(x)), \qquad f_{ij}(x) = \frac{f_i(\overline{x}^j) - f_i(x)}{\overline{x}^j_j - x_j} \qquad (i, j \in [\![1; n]\!]).$$

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 le calcul est effectué

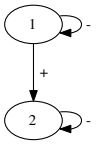

FIGURE 1.4 – $g(x_1, x_2) = (\overline{x_1}, x_1\overline{x_2})$

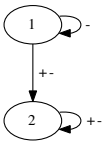

FIGURE 1.5 – $h(x_1, x_2) = (\overline{x_1}, x_1\overline{x_2} + \overline{x_1}x_2)$

FIGURE 1.6 – Graphes d'interactions de fonctions booléennes dans $\mathbb{B}^2$

dans $\mathbb{Z}$. On a le lien suivant avec la *matrice d'adjacence* A de f dans $\mathbb{B}^{n^2}$: le booléen A_{ij} vaut 1 si et seulement s'il existe $x \in \mathbb{B}^n$ tel que $f_{ij}(x)$ est différent de 0.

En outre, les interactions peuvent se représenter à l'aide d'un graphe $G(f)$ orienté et signé défini ainsi : l'ensemble des sommets est $[\![1;n]\!]$ 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.

Soit P une suite d'arcs de $G(f)$ 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 $G(f)$ 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 $G(f)$ a une *boucle* positive (resp. négative) , si $G(f)$ a un arc positif (resp. un arc négatif) de i vers lui-même.

1.4/ DISTANCE SUR L'ESPACE $[\![1;n]\!]^{\mathbb{N}} \times \mathbb{B}^n$

On considère l'espace $\mathcal{X} = [\![1;n]\!]^{\mathbb{N}} \times \mathbb{B}^n$ et on définit la distance d entre les points $X = (s, x)$ et $X' = (s', x')$ de $\mathcal{X}$ par

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

On note que dans le calcul de $d_H(x, x')$– appelée distance de Hamming (cf. glossaire) 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 (cf. glossaire) $\lfloor d(X, X') \rfloor$ 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.

On a démontré que pour toute fonction booléenne f, G_f est continue sur $\mathcal{X}$ (cf annexe A.2).

2

COMBINAISONS SYNCHRONES ET ASYNCHRONES DE SYSTÈMES BOOLÉENS

Refaire le châpeau

2.1/ GÉNÉRALISATION AU CADRE ASYNCHRONE

Dans ce chapitre une stratégie $s = (s^t)^{t\in\mathbb{N}}$ est une séquence *des éléments* qui sont mis à jour au temps t. Pratiquement, on représente ceci comme un nombre décimal dont la représentation en binaire donne la liste des éléments modifiés. Par exemple, la stratégie définie par

$$s^t = 6 \text{ si } t \text{ est pair et } s^t = 3 \text{ sinon} \tag{2.1}$$

active successivement les deux premiers éléments (6 est 110) et les trois derniers élements (3 est 011). 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 où non les valeurs les plus à jours dépent du temps de calcul et du temps d'acheminement de celles-ci. On parle de latence, de délai.

Formalisons le mode les itérations asynchrones. 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 represente 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 asynchrones* est défini pour chaque $i \in \{1, \ldots, n\}$ et chaque $t = 0, 1, 2, ...$ par :

$$x_i^{t+1} = \begin{cases} f_i(x_1^{D_{i1}^t}, \ldots, x_n^{D_{in}^t}) \text{ if } \mathit{bin}(s^t)[i] = 1 \\ x_i^t \text{ sinon} \end{cases} \tag{2.2}$$

où *bin* convertit un entier en un nombre binaire. Les itérations de f sont *convergentes* modulo une configuration initiale x^0, une stratégi 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 :

$$\exists t_0 \,.\, (\forall t \,.\, t \geq t_0 \Rightarrow x^t = x^{t_0}). \tag{2.3}$$

Sinon les itérations sont dites *divergentes*. De plus, si $(x^{(t)})^{t\in\mathbb{N}}$ défini selon l'équation (2.2) satisfait (2.3) 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*.

2.2/ EXEMPLE JOUET

On considère cinq éléments prenant à valeurs dans $\mathbb{B}$. Une configuration dans $\mathbb{B}^5$ est représentée par un entier entre 0 et 31. La FIGURE 2.1 donne la fonction définissant la dynamique du système. La FIGURE 2.2 donne le graphe d'intéraction associé à cette fonction. On note que le graphe d'intéraction contient cinq cycles. Les résultats connus [Bahi, 2000] de conditions suffisantes établissant la convergencedu système pour les itérations synchrones et asynchrones sont basés sur l'absence de cycles. Ils ne peuvent donc pas être appliqués ici.

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

FIGURE 2.1 – Fonction f de l'exemple jouet.

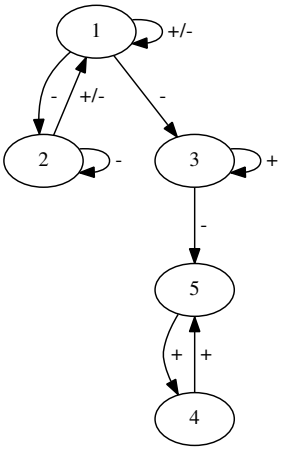

FIGURE 2.2 – Graphe d'interaction associé à f.

In what follows and for brevity reasons, configurations are represented as decimal numbers instead of binary numbers. The graph of parallel iterations is given in FIGURE 2.3. Starting from any configuration, the network converges to the fixed point corresponding to the decimal number 19.

An extract of the graph of chaotic iterations is given in FIGURE 2.4. Arc labels represent the activated elements expressed as decimal number. Iterations do not always converge with strategy depicted in (2.1) : starting from configuration 3, the network goes to configuration 11 and goes back to configuration 3.

Since chaotic iterations do not converge for some strategies, it is obvious that convergence will not be ensured for asynchronous iterations with the corresponding strategies. Indeed, even if all the components are activated, *i.e* when S^t is constantly equal to $2^n - 1$, the delays may introduce divergence. For instance, consider the matrix D^t to be equal to (t) except in D^t_{12} where it is equal to $t-1$ if t is odd. Then if t is even, x^{t+1} is $f(x^t)$; if t is odd we have

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

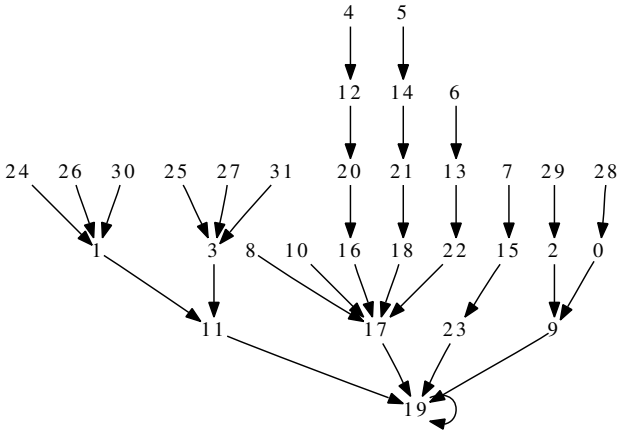

FIGURE 2.3 – Itérations parallèlles de f.

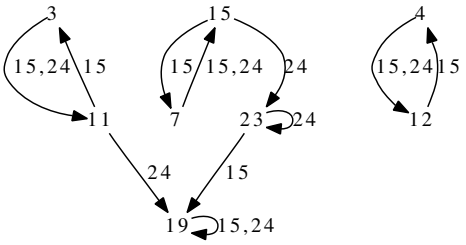

FIGURE 2.4 – Extrait d'itérations chaotiques.

Starting from $x^0 = 00011$, the system reaches $x^1 = 01011$ and enters in a cycle between these two configurations. We are then confronted to divergent asynchronous iterations whereas the synchronous ones converge. In the next section, a particular execution mode is described which enables asynchronism in iterations while guaranteeing the convergence.

3

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

3.1/ 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 *3.1 précise la fonction* f *considérée et la* FIGURE *3.2 donne son graphe d'intéraction.*

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

FIGURE 3.1 – Fonction à itérer

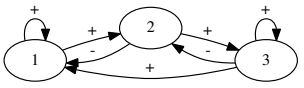

FIGURE 3.2 – Graphe d'intéraction

FIGURE 3.3 – 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.*

3.2/ 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 3.4 – 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 *3.4 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* $x_{j+1}^{D_{i+1\,j+1}^{t-1}}$ *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 3.5 – 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 3.6 – 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 3.7 – 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 3.5, 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 3.7) 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 3.8 – 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 3.9 – 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 3.10 – Application de la fonction f.

3.3/ DU SYSTÈME BOOLÉEN AU MODÈLE PROMELA

Les éléments principaux des itérations asynchrones rappelées à l'équation (2.2) 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.

3.3.1/ LA STRATÉGIE

Regardons comment une stratégie pseudo périodique peut être représentée en PROMELA. Intuitivement, un process `scheduler` (comme représenté à la FIGURE 3.6) 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`.

3.3.2/ 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 3.9. 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 3.8
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 3.11 – 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 3.12 – 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 3.10 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`).

3.3.3/ 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 3.11 et 3.12, 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.

3.3.4/ 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 :

$$\diamond(\square \mathtt{Xp} = \mathtt{X}) \tag{3.1}$$

où les opérateur $\diamond$ et $\square$ 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.

3.4/ CORRECTION ET COMPLÉTUDE DE LA DÉMARCHE

Cette section présente les théorèmes de correction et de complétude de l'approche. (Théorèmes 4 et 5). Toutes les preuves sont déplacées en annexes A.3.

Théorème 1 (Correction)**.** *Soit ϕ un modèle de système dynamique discret et ψ sa traduction PROMELA. Si ψ vérifie la propriété LTL (3.1) sous hypothèse d'équité faible, alors les itérations de ϕ sont universellement convergentes.*

Théorème 2 (Complétude)**.** *Soit ϕ un modèle de système dynamique discret et ψ sa traduction. Si ψ ne vérifie pas la propriété LTL (3.1) sous hypothèse d'équité faible, alors les itérations de ϕ ne sont pas universellement convergentes.*

3.5/ 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 [3] (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 [1]. *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 3.14). 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 ($\bot$) 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.

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

	Parallèles			Chaotiques		
	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	$\top$		

FIGURE 3.13 – 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	$\bot$	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]	$\top$			$\top$			$\bot$			$\bot$		

FIGURE 3.14 – Expérimentations avec des itérations asynchrones

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 3.15 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.

3.6/ 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 $\lim_{t\to\infty} s^t_{ij} = +\infty$. One challenge of this work should consist in weakening this constraint. We plan as future work to take into account other automatic

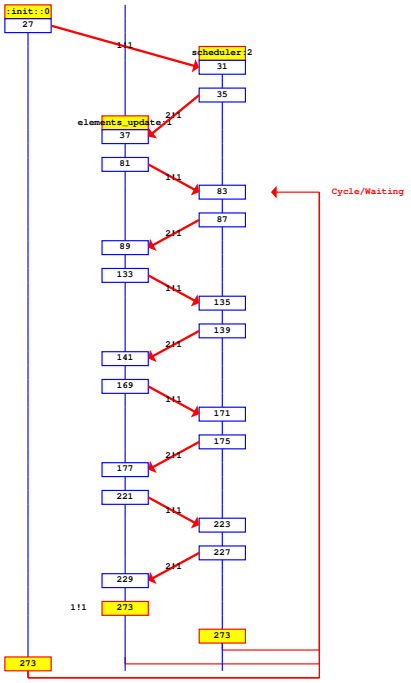

FIGURE 3.15 – Contre exemple de convergence pour 3.15

approaches to discharge proofs notably by deductive analysis [Couchot et al., 2005].

Mixage

A

PREUVES SUR LES SDD

A.1/ PREUVE DU 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 1. *$G(f^\alpha)$ 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 2. *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 2. *Les graphes $\Gamma(f^\alpha)$ et $\Gamma(f)^\alpha$ sont isomorphes.*

Preuve 3. *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 4. *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 1, 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 2, $\Gamma(f)^0$ 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.*

A.2/ PREUVE DE 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 $\forall t > T_0, d(Gf(s_t, x^t), G_f(s, x)) < \epsilon$.

A.3/ PREUVE DE CORRECTION ET DE COMPLÉTUDE DE L'APPROCHE DE VÉRIFICATION DE CONVERGENCE À L'AIDE DE SPIN

Voir section 3.4

Cette section donne les preuves des deux théorèmes de correction et complétude du chapitre 3.

Lemme [3] (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 [5]**.** *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, $m \leq \delta_0$; Y^0_{ij} and Y^{m-1}_{ij} are respectively the head and the tail of the channel. Secondly, let $(M^t_{ij})^{t\in\{1,1.5,2,2.5,\ldots\}}$ be a sequence such that M^t_{ij} is the partial function that associates to each k, $0 \leq k \leq m-1$, 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 :

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

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 $M^1_{ij}(0) = (\texttt{Xp[i]}, 0, 0)$: 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 (2.2). The function M^{t+1}_{ij} is obtained by the successive updates of M^t_{ij} through the two functions `fetch_values` and `diffuse_values`. Abusively, let $M^{t+1/2}_{ij}$ be the value of M^t_{ij} 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 $(Y^0_{ij}, a^0_{ij}, c^0_{ij})$ be the value of $M^t_{ij}(0)$ at the beginning of `fetch_values`. If t is equal to $c^0_{ij} + 1$ then we execute the instruction that assigns Y^0_{ij} (*i.e.*, the head value of `channels[i].sent[j]`) to Xd^t_{ji}. In that case, the function M^t_{ij} is updated as follows : $M^{t+1/2}_{ij}(k) = M^t_{ij}(k+1)$ for each k, $0 \leq k \leq m-2$ and $m-1$ is removed from the domain of $M^{t+1/2}_{ij}$. Otherwise (*i.e.*, when $t < c^0_{ij} + 1$ or when the domain of M_{ij} is empty) the `skip` statement is executed and $M^{t+1/2}_{ij} = M^t_{ij}$.

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

Lemme 4 (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^{t}_{ij} is not empty, then

$$\begin{cases} M^{1}_{ij}(0) & = & \left(X_i^{D^{0}_{ji}}, 0, 0\right) \\ \text{if } t \geq 2 \text{ then } M^{t}_{ij}(0) & = & \left(X_i^{D^{c}_{ji}}, D^{c}_{ji}, c\right),\ c = \min\{l | D^{l}_{ji} > D^{t-2}_{ji}\} \end{cases} \tag{A.1}$$

Secondly we have :

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

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^{t}_{k} i.e., $F_k\left(X_1^{D^{t-1}_{k1}}, \ldots, X_n^{D^{t-1}_{kn}}\right)$ at the end of the t^{th} iteration.

Preuve 6. *The proof is done by induction on the number of iterations.*

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

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^{0}_{i}. 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 $F(\texttt{Xd[}k\texttt{].v[0]}, \ldots, \texttt{Xd[}k\texttt{].v[}n-1\texttt{]})$. Thus, by definition of Xd, it is equal to $F(Xd^{1}_{k\,0}, \ldots, Xd^{1}_{k\,n-1})$. Thanks to (A.2), we can conclude the proof.

Inductive case : *Suppose now that lemma 4 is established until iteration l.*

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

At iteration l, if $l < c+1$ then the `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 $\mathrm{dom}(M^{l}_{ij})$ of M^{l}_{ij} :

- *if $\mathrm{dom}(M^{l}_{ij}) = \{0\}$ and $\forall k \,.\, k \geq l \Rightarrow D^{k}_{ji} \neq l$ is established then $\mathrm{dom}(M^{l+1}_{ij})$ 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* $\texttt{Xp}[k] = F(\texttt{Xd}[k][0], \ldots, \texttt{Xd}[k][n-1])+$. *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 5. *Bounding the size of channels to max =* δ_0 *is sufficient when simulating a DDN where delays are bounded by* δ_0.

Preuve 7. *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^t_j, ..., $X^{t-\delta_0+1}_j$. *They can be stored into any channel of size* δ_0.

Théorème 4 (Soundness wrt universal convergence property)**.** *Let* ϕ *be a DDN model and* ψ *be its translation. If* ψ *verifies the LTL property (3.1) under weak fairness property, then iterations of* ϕ *are universally convergent.*

Preuve 8. *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 (3.1).

Théorème [5] (Completeness wrt universal convergence property)**.** *Let ϕ be a DDN model and ψ be its translation. If ψ does not verify the LTL property (3.1) under weak fairness property then the iterations of ϕ are divergent.*

Preuve [9]. *For models ψ that do not verify the LTL property (3.1) it is easy to construct corresponding iterations of the DDN, whose strategy is pseudo-periodic since weak fairness property is taken into account.*

BIBLIOGRAPHIE

[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.

[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.

[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.

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