N° d'ordre : 09/2007-M/IN

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTÈRE D'ENSEIGNEMENT SUPERIEUR ET DE LA RECHRCHE SCIENTIFIQUE

UNIVERSITÉ DES SCIENCES ET DE LA TECHNOLOGIE

"HOUARI BOUMEDIENE"

FACULTÉ DE L'INFORMATIQUE

MÉMOIRE

Présenté pour l'obtention du diplôme de MAGISTER

EN : INFORMATIQUE

Spécialité: programmation et Système

Par TAIBI Meriem

Sujet

Analyse des systèmes de grande taille

Soutenu le 17/07/2007, devant le juré composé de:

Mr- N. BADACHE, Professeur à l'USTHB, Alger.	Président.
Mr-M.C BOUKALA, Chargé de cours à l'USTHB, Alger	Dteur de Thèse
Mr- L. SEKHRI, Maître de Conférences à l'Université d'Oran	Examinateur
Mr- H. AZZOUNE, Maître de Conférences à l'USTHB, Alger.	Examinateur

Résumé

La génération de l'espace d'états ou du graphe d'accessibilité des modèles à états discrets, tel que les réseaux de Petri (RdP) est une étape nécessaire dans l'analyse des propriétés d'un système. Cependant la génération de cet espace d'états est souvent confrontée au problème d'explosion combinatoire. Pour pallier à ce problème, différentes techniques sont proposées pour la construction et le stockage de l'espace d'états, profitant de certaines propriétés telles que l'analyse de systèmes symétriques, l'analyse modulaire, l'utilisation des ensembles stubborn,...etc.

Néanmoins, ces méthodes malgré qu'elles permettent une réduction notable de l'espace des états, elles ne concernent que des systèmes particuliers (possédant la propriété de symétrie, construits en assemblant des modules ou constitués d'un ensemble de processus relativement indépendant qui synchronisent occasionnellement). La distribution de l'espace d'états est proposée comme alternative pour permettre l'analyse de modèles de grande taille. Cet espace est ainsi réparti sur un ensemble de stations interconnectées.

Notre travail consiste à étudier les algorithmes distribués permettant la génération répartie de l'espace d'états, ainsi qu'à présenter des techniques pour analyser les propriétés qualitatives et quantitatives de cet espace.

Mots clés : *Réseaux de Petri, espace d'états, vérification, propriétés, algorithmes distribués.*

Introduction générale

L'augmentation de la complexité des systèmes actuels, qui deviennent de plus en plus distribués et concurrents, a incité au développement de méthodes d'analyse, de validation et de vérification du comportement de ces systèmes.

Pour cela la modélisation de ces derniers devient une étape indispensable, qui souvent nécessite la génération de leur espace d'états. Cette génération consiste en le calcul de tous les états atteignables par le système modélisé et de leur représentation sous forme d'un graphe. La taille de cet espace croit d'une manière exponentielle par rapport à la taille du système modélisé, en particulier lorsque celui-ci présente un degré de parallélisme important. Ainsi cet espace occupe une partie considérable qu'il est souvent impossible de stocker en mémoire. Plusieurs travaux se sont intéressés à la résolution de ce problème, connu sous le nom du phénomène «d'explosion combinatoire de l'espace d'états».

Parmi ces travaux, nous avons les techniques de réduction de l'espace d'états, telles que la vérification à la volée, qui vérifie les propriétés en même temps que l'exploration *[Jou 06]*, la vérification modulaire *[Pet 05]*, la représentation symbolique telle que les diagrammes de décision *[Paq 05]*. Parmi ces techniques nous citons aussi celles qui évitent de représenter tous les états du système en prenant en compte certaines propriétés du modèle telles que la symétrie *[Jor 99]* et les ensembles de Stubborn *[Kri 99]*.

Néanmoins, ces techniques malgré qu'elles permettent une réduction notable de l'espace des états, elles ne concernent que des systèmes particuliers (possédant la propriété de symétrie, construit en assemblant des modules ou constitués d'un ensemble de processus relativement indépendant qui synchronisent occasionnellement).

Durant cette dernière décennie, plusieurs travaux se sont orientés vers une nouvelle approche qui consiste à distribuer l'espace d'états sur un ensemble de machines connectées en réseau, ce qui permet l'augmentation de la taille des systèmes à vérifier.

La répartition de l'espace d'états sur l'ensemble des processus se fait à l'aide d'une fonction de répartition (fonction de hashage), qui attribue chaque état généré à un processus du réseau. Le choix de la fonction de hashage est trés

important et ceci afin d'assurer un équilibre de charge entre les différents processus.

Le problème qui se pose après la répartition de l'espace d'états, est comment peut-on nous vérifier les différentes propriétés, qualitatives et quantitatives, de tel espace? De ce fait, différents travaux se sont orientés vers la vérification distribuée, où la plus part parmi eux se focalisent sur le modèle checking et la logique temporelle *[Bar 01],[Ler 99]*.

Dans ce travail, nous nous intéressons à la vérification distribuée des propriétés telles que la vivacité et la bornitude qui sont des propriétés importantes et complexes. La vérification de la vivacité qui nécessite la génération de tout l'espace d'états, se base sur la construction des composantes fortement connexes du graphe. Par contre la vérification de la bornitude est une vérification à la volée, qui se base sur la construction de l'arbre de recouvrement d'une manière distribuée *[Bra 83]*.

Afin d'évaluer les performances des systèmes modélisés, nous proposons aussi dans ce travail, un algorithme distribué permettant la résolution numérique du système d'équation, généré à partir du processus stochastique, et cela après vérification de l'ergodicité du modèle. Cet algorithme est basé sur la méthode de Gauss pour la résolution des systèmes d'équations.

Le présent document est organisé comme suit : Le chapitre 1 est consacré à la présentation de quelques outils de modélisation tels que : les RdP simples et leur extensions, RdP temporels, RdP colorés et RdP stochastiques, en définissant tous les concepts et les propriétés de ces modèles. Nous présentons aussi dans ce chapitre l'algorithme séquentiel qui permet la génération de l'espace d'états. Dans le chapitre 2 nous abordons quelques approches traditionnelles permettant l'exploration de l'espace d'états en exploitant les caractéristiques du système représenté, tels que la symétrie, les ensembles des Stubborns et la modularité. Dans le chapitre 3 nous présentons les nouvelles approches d'exploration d'espace d'états, qui se basent sur la distribution de ce dernier sur plusieurs processus. Le chapitre 4 est dédié à notre conception, où nous proposons une vérification distribuée des propriétés du modèle. Dans le chapitre 5 nous donnons les résultats obtenus aprés exécution des algorithmes proposés sur un ensemble de modèles, tels que les bases de données distribuées (BDD) et l'exemple des philosophes. Nous terminons par une conclusion et un ensemble de perspectives qui peuvent être d'éveloppés

dans des futurs travaux.

Chapitre1:
Les Réseaux de Petri

I. Introduction:

Depuis les dernières décennies, les systèmes répartis sont de plus en plus répandus dans le monde informatique. Ils sont caractérisés par leur puissance et leur complexité. Dans ces systèmes coexistent plusieurs processus qui s'exécutent en parallèle et en concurrence sur différentes ressources (mémoires, périphériques d'entrés / sorties,...etc.), et qui communiquent entre eux par échange de messages.

Néanmoins, les avantages fournis par ces systèmes ont pour contrepartie la complexité de les concevoir et de s'assurer qu'ils possèdent les propriétés attendues.

Il est alors plus approprié et plus admis de valider le système et même d'effectuer une évaluation de ses performances avant sa construction.

L'analyse quantitative et qualitative des comportements des systèmes parallèles et distribués a suscité de nombreuses tentatives visant à représenter le système à l'aide des outils de modélisation. Ces outils de modélisation permettent, en effet, d'obtenir une représentation à partir de laquelle on peut étudier le comportement des processus à des fins de vérification.

II. Les Réseaux de Petri :

Le formalisme des réseaux de Petri (RdP) a été introduit dans les années soixante par C.A.Petri *[Pet 62]*.Les RdP constituent un outil graphique et mathématique important, utilisé pour la description et l'analyse des systèmes concurrents, distribués, et parallèles*[Bra 83]*.

Un RdP est un graphe biparti, constitué de deux types de noeuds, les places représentées par des cercles, et les transitions par des tirets. Les arcs relient les places aux transitions et inversement. Les places peuvent contenir des jetons, l'état d'un RdP est défini par le nombre de jetons contenus dans chaque place, représenté par un vecteur M appelé marquage, M(p) représente le nombre de marques dans la place p.

Les places possédant des arcs qui les relient aux transitions, i.e. ayant des arcs allant de ces places aux transitions, sont souvent appelées par simplification les places d'entrée de t ($\bullet\cdot$ t); les places, reliées à une transition par un arc allant de t à ces places, sont appelées places de sortie de t(t$\cdot\bullet$). De même, on parlera de transitions d'entrée et de sortie des places ($\bullet$p,p$\bullet$ respectivement).

II.1 Définition formelle : *[Bra 83]*

Un réseau de Petri est défini formellement par un quadruplet:
R=<P,T,Pré,Post> où :
P : ensemble fini de places,
T : ensemble fini de transitions,
Pré : P×T→ N est l'application d'incidence avant (Pré-condition),
Post : P×T→ N est l'application dâincidence arrière (Post-condition).

II.2 RdP marqué :

Un réseau de Petri marqué est un couple N =<R,M> où :
R : est un RdP,
M : est un vecteur de P vers N représentant le marquage de R tel que M(p) est le marquage de la place p, on dit aussi, le nombre de jetons (marques) contenus dans P.

L'état initial d'un RdP est défini par un marquage initial noté par M_0.

II.3 Représentation graphique:

À un réseau de Petri, on associe un graphe biparti où les noeuds (sommets) sont les places et les transitions .Un arc de valuation « n» relie une place p à une transition t si Pré(p,t)=n et une transition t à une place p si Post(p,t)=n.

Dans ce graphe, une place est représentée par un cercle ◯et une transition par un rectangle ou un tiret .▬

Un arc relie une place à une transition ou bien une transition à une place mais ne relie jamais deux sommets de même type.

Exemple:

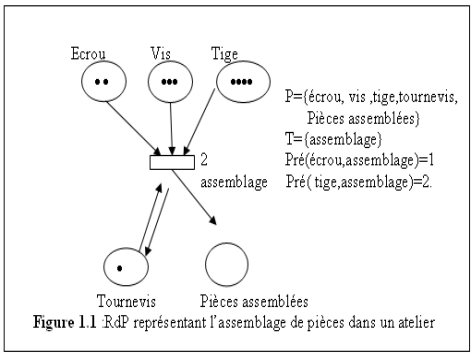


Figure 1.1 :RdP représentant l'assemblage de pièces dans un atelier

II.4 Représentation matricielle :

Étant donné un réseau R, nous pouvons également le décrire par une matrice, dite matrice d'incidence C qui représente l'application :

$P \times T \rightarrow Z$ définie par : $p \in P$, $t \in T$: $C(p,t)=Post(p,t)-Pré(p,t)$.

C(p,t) donne la modification pour la place p résultant du franchissement de la transition t. Post (resp.Pré) est dite matrice d'incidence arrière (resp.matrice d'incidence avant).

III. Evolution d'un RdP :

Soit R un réseau de Petri, M un marquage de R.

Une transition t est franchissable (tirable) pour un marquage M si et seulement si :

$\forall\ p \in P$, $M(p) \geq Pré(p,t)$. Si t est franchissable pour M, le franchissement(le tir) de t permet d'obtenir le nouveau marquage M', tel que :

$\forall$ p$\in$ P, M'(p)=M(p)-Pré(p,t)+Post(p,t).
on note : M[t> la franchissabilité de t à partir de M.

Remarque :

Le franchissement d'une transition t est une opération qui consiste pour chaque place de P (entrée de t), à enlever un nombre de marques égal au poids de l'arc la reliant à la transition et à rajouter à chaque place p (sortie de t) un nombre de marques égal au poids de l'arc reliant la transition à la place.

III.1 Séquence de franchissement :

Soit s$\in$ T*, s est une séquence franchissable pour un marquage M ssi :
s'$\in$ T* tel que s=s'.t et de plus M' tel que : M[s'>M'et M'[t>M", ce franchissement on le note M[s>M".

III.2 Equation fondamentale :

Soit S le vecteur dont les composantes S(t) sont le nombre d'occurrences d'une transition t dans une séquence de franchissements s, ce vecteur est appelé vecteur caractéristique de s.
Si M[s>M',on peut alors écrire : M'=M-Pré.S +Post.S .
M'=M+C.S.

III.3 Marquages accessibles :

L'ensemble des marquages atteints à partir de M_0 d'un RdP noté par $A(R,M_0)$ est défini par : $A(R,M_0)=\{M/\ s\in T \text{ tq } :M_0[s>M\}$.

Exemple:

L'ensemble des marquages accessibles du RdP de la figure 1.1 est:

$A(R, M_0) = \{(2,3,4,1,0),(1, 2,1,1,1), (0,1,0,1,2)\}$.

III.4 Graphe des marquages accessibles :

On appelle graphe des marquages accessibles, le graphe G défini par :

L'ensemble des noeuds: est l'ensemble des marquages accessibles $A(R, M_0)$

L'ensemble des arcs: Un arc étiqueté t joint M à M' si et seulement si : $M\ [t>M'$.

IV. Propriétés des RdP :

Pour analyser le fonctionnement d'un système représenté par un réseau de Petri, nous étudierons certaines propriétés :

1. Bornitude :

Une place p d'un réseau marqué $<R,M>$ est dite k-bornée ($k\in N$), si pour tout marquage accessible $M \in A(R,M_0)$, $M(p) \leq k$.

Un réseau marqué $<R,M>$ est dit borné si toutes ses places sont k-bornées, k est appelé borne de R. Si k=1,alors le RdP est dit sain (sauf).

L'intérêt de l'existence d'un réseau borné intervient dans le cas où les variables d'état ou les ressources du phénomène décrit ne doivent pas franchir un seuil fixé.

2. Quasi-vivacité :

Soit R un réseau marqué. Une transition t est dite quasi-vivante, s'il existe un marquage $M \in A(R,M_0)$ tel que $M[t>$.

Cette propriété désigne la possibilité de franchir au moins une fois cette transition depuis le marquage initial. Si une transition est non quasi-vivante, l'opération qu'elle représente est inutile au fonctionnement du système modélisé.

Si toutes les transitions sont quasi-vivantes, on dit que le réseau est quasi-vivant .

3. Vivacité :

Une transition t d'un RdP marqué $<R,M_0>$ est dite vivante, si pour tout marquage accessible $M \in A(R,M_0)$, t est quasi-vivante pour le réseau R. Un réseau marqué $<R,M_0>$ est vivant si toutes ses transitions sont vivantes. Un réseau R est vivant, s'il existe un marquage M tel que $<R,M>$ est vivant.

La vivacité est une propriété trés forte : elle permet de garantir une durée de vie totale infinie, c à d: s'assurer qu'un système donné garde toujours ses fonctionnalités à n'importe quel moment (aucune transition ne sera définitivement infranchissable).

Une conséquence immédiate de la vivacité est l'absence de blocage dans le système.

4. Réinitialisabilité :

Un réseau est dit réinitialisable (ou propre) si :

$\forall M \in A(R,M_0)$, il existe une séquence de tirs 's' tirable à partir de M et produisant le marquage M_0.

La réinitialisabilité assure que le système peut toujours revenir dans son état initial. C'est une propriété importante à vérifier, parce qu'il est souhaitable de revenir à l'état initial dans le cas d'une erreur.

5. Etat puits et réseau sans blocage :

Un réseau $<R,M_0>$ admet un état puits M, si aucune transition n'est franchissable depuis M. Un réseau $<R,M_0>$ est dit sans blocage, si tout marquage atteignable à partir de M_0 n'est pas un état puits.

6. Etat d'accueil :

Un réseau <R,M_0> admet un état d'accueil Ma, ssi Ma est un marquage toujours accessible, c'est à dire :

$\forall$ M$\in$A(R,M_0), Ma est toujours accessible à partir de M.

Remarques :

- Si le marquage initial est un état d'accueil alors le réseau est reinitialisable.
- Si un réseau admet un état d'accueil, alors il est aisé de démontrer qu'il est sans blocage. Il suffit de vérifier qu'il existe au moins une transition franchissable à partir de ce marquage.

V. Arbre de recouvrement :

L'arbre de recouvrement permet de déterminer pour un RdP ayant pour marquage initial M_0 et un autre marquage M, s'il existe un marquage M' accessible de telle sorte que M'$\geq$ M (i.e.M' recouvre M). L'arbre de recouvrement est obtenu par l'énumération récursive de tous les marquages possibles en partant du marquage initial. Cependant, cet arbre peut être infini. De ce fait, on utilise un symbole ω représenter le nombre de jetons qui peut augmenter sans limite. Quatre opérations arithmétiques s'appliquent sur ω. Pour a entier :

$a < \omega$

$\omega \leq \omega$

$\omega + a = \omega$

$\omega - a = \omega$

Ayant ajouté la notion ω, on peut utiliser l'algorithme suivant pour construire l'arbre de recouvrement .

Grâce à l'arbre de recouvrement, on peut tirer les résultats suivants :

- Un arbre de recouvrement sans ω est un arbre d'accessibilité.

Algorithm 1 Algorithme de l'arbre de recouvrement *[Bra 83]* :

1. Marquer la racine représentée par M_0 **new**.
2. Tant qu'il existe un marquage marqué **new**;
(a) sélectionner un marquage M marqué **new**
(b) s'il n'y a aucune transition tirable par M, marquer M par **deadend**.
(c) Pour toutes les transitions franchissables par le marquage M:
i.Trouver M' accessible immédiatement de M;
ii.S'il existe un marquage M" sur le chemin entre M_0 et M' de
sorte que M'(p_i)≥M"(p_i),∀i= 1,...,n,et M'≠M",alors :
remplacer M'(p_i) par Ïωoù M'(p_i) >M"(p_i);
ajouter M' dans l'arbre, en le marquant **new**
marquer l'arc avec la transition tirée; marquer M' new.
(d)si M est identique à un autre marquage sur le chemin entre M_0 et M ,marquer M **old** et choisir un autre marquage **new**.

- Le RdP avec le marquage initial M_0 est k-borné si le symbole ω n'apparaît pas dans l'arbre de recouvrement. La borne supérieure k est donnée par le nombre de jetons le plus grand de l'arbre.
- L'arbre de recouvrement permet de détecter les transitions qui ne sont pas tirées, ou qui ne sont plus tirées à partir d'un certain niveau.
- S'il y a des marquages deadend dans l'arbre de recouvrement, alors il y a des blocages dans le système modélisé.

Exemple :

Pour la figure ci-dessous, nous constatons que le RdP n'est pas borné car les ω existent dans l'arbre de recouvrement .C'est la place p_1 qui n'est pas bornée . Le RdP n'est pas réinitialisable car on ne peut jamais revenir au marquage initial. Si on tire la transition t_2, on se trouve dans un état de blocage . Cela aussi signifie que la transition t_2 n'est pas vivante et ainsi le RdP non plus.

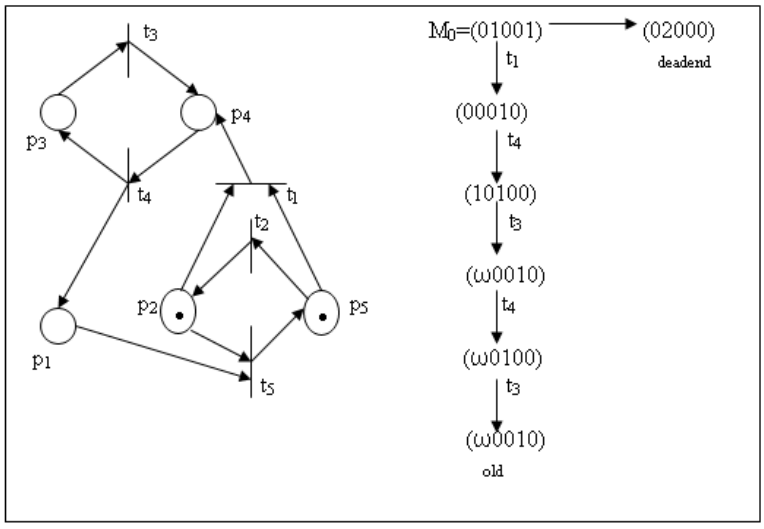


Figure1.2 :Un RdP avec son arbre de recouvrement

VI. Extentions des RdP :

Les RdPs, introduits en 1962 afin de modéliser la composition et la communication entre automates, ont été améliorés afin de représenter les différents aspects et concepts liés à la modélisation des comportements parallèles et distribués, et en particulier des aspects temporels et stochastiques.

Les différentes extensions des RdP ont été proposées selon les deux axes importants suivants:

Le premier axe a conduit à modéliser de façon plus simple et plus compacte les comportements qualitatifs. (RdP colorés).

Le deuxième axe, complémentaire du premier et indispensable lors de la conception des systèmes, concerne les extensions quantitatives, c'est-à-dire les aspects comportementaux liés à la fois aux caractéristiques temporelles et stochastiques des systèmes.

Ainsi, la figure suivante représente une partie des principales familles.

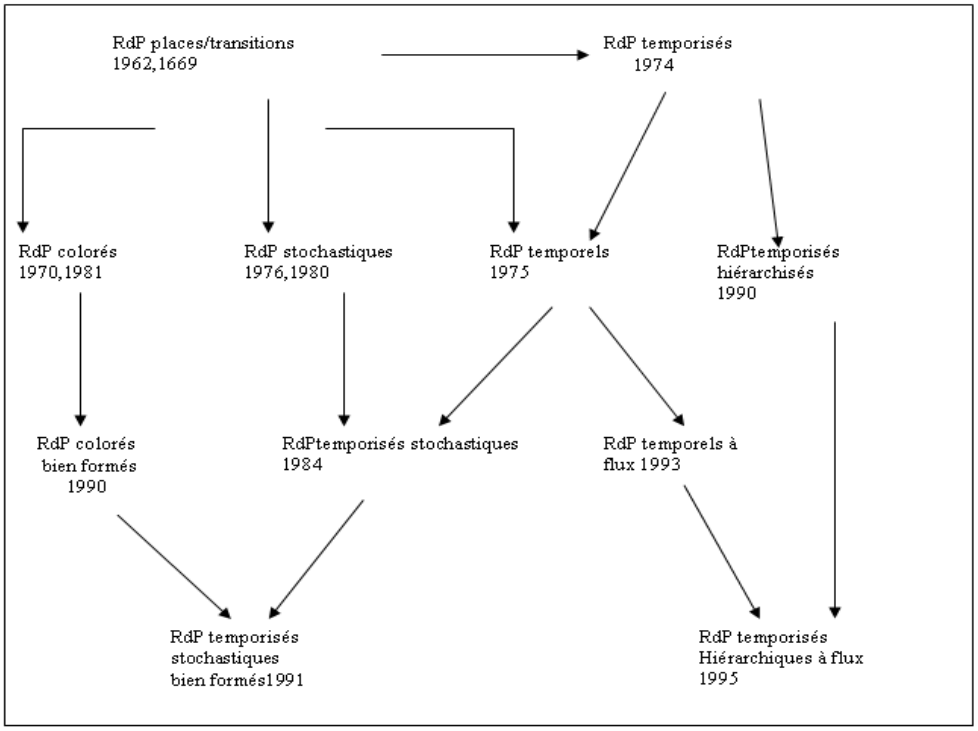


Figure1.3 : Famille des RdP

VI .1 Les réseaux de Petri colorés (RdPC):

Les réseaux de Petri colorés (RdPC) ont été développés comme un outil de modélisation idéal des gros systèmes qui permet de condenser les informations à représenter.

VI.1.1 Définition: *[Bra 83]*

Soit Co=$\{c_1,c_2,...,c_I\}$ un ensemble fini de couleurs. Un marquage M des places dâun RdPC se définit comme une application de lâensemble des places P dans lâensemble des fonctions de Co dans les entiers : M : P$\rightarrow$[$C_0\rightarrow$N].

Ainsi M(p)(c_{0i}) désigne le nombre d'éléments de couleurs c_{0i} au sein du marquage M(p) de la place p.

Le principe d'abréviation induit par le type de marques s'énonce alors ainsi :

Une place p et son marquage M(p) constituent une abréviation de $p_1,...,p_L$ places distinctes telles que chaque place p_i posséde un marquage de valeur M(p)(c_{0i}).

On spécifie l'utilisation des marques colorées par des expressions étiquetant les arcs du réseau. Ces expressions sont de deux types :

<k> où k désigne un sous-ensemble quelconque de couleurs,

<x> où x est une variable libre prenant ses valeurs dans un sous-ensemble de couleurs spécifié par l'énoncé « domaine(x)=...» ; un tel énoncé faisant partie de la description d'un réseau coloré.

Le premier type d'expressions permet de lier le franchissement d'une transition à la présence dâun sous-ensemble k de couleurs au sein du marquage d'une place d'entrée de cette transition. Quant au second type d'expressions, il induit une condition de franchissement soumise à la présence d'une même couleur au sein du marquage de toutes les places d'entrée d'une transition, liées à cette dernière par des arcs étiquetés par la même variable.

VI.1.2 Evolution d'un RdPC :

Le franchissement d'une transition t depuis un marquage M est régi par les règles suivantes : t est franchissable depuis un marquage M si et seulement si :

pour tout sous-ensemble de couleurs k, pour toute place p liée à t par un arc(p,t) portant l'étiquette <k>, pour toute couleur s$\in$k, M(p)(s) $\geq$1.

Pour toute variable x, trouver s$\in$ domaine(x), tel que pour toute place p' liée à t par un arc(p' t) d'étiquette <x>, M(p')(s) $\geq$1.

Dans ce dernier cas, nous dirons que s est une couleur effective de x au marquage M pour t .

VI.1.3 Le marquage des RdPC : *[Bra 83]*

Notation : Ef (x,M,t) désigne l'ensemble des couleurs effectives de x pour t au marquage M, s' il existe une place p et un arc (p,t) d'étiquette <x>; sinon Ef(x,M,t)=domaine(x).

Le marquage M' résultant du franchissement d'une transition t se définit alors comme suit :

M[t>M' si et seulement si M' satisfait les trois conditions suivantes :

- pour toute place p qui n'est pas connectée à t, M(p)=M(p).
- pour tout sous-ensemble k⊆Co pour toute place p liée à t par un arc v étiqueté par <k>

- si v=(p,t) alors pour toute couleur s∈Co

$$M'(p)(s) = \begin{cases} M(p)(s)-1 & si\ s\in K \\ M(p)(s) & sinon \end{cases}$$

- si v=(t,p) alors pour toute couleur s∈k

$$M'(p)(s) = \begin{cases} M(p)(s)+1 & si\ s\in K \\ \\ M(p)(s) & sinon \end{cases}$$

pour toute variable x, il existe s∈EF(x,M,t) tel que pour toute place p liée à t par un arc v d'étiquette <x> :

-si v = (p,t) alors pour toute couleur s'∈Co

$$M'(p)(s') = \begin{cases} M(p)(s)-1 & si\ s'=s \\ \\ M(p)(s) & sinon \end{cases}$$

-si v =(t,p) alors pour toute couleur s'∈Co

$$M'(p) = \begin{cases} M(p)(s)+1\ si\ s'=s \\ \\ M(p)(s) & sinon \end{cases}$$

VI.2 Les réseaux de Petri temporels:

Parmi les techniques proposées pour spécifier et vérifier des systèmes dans lesquels le temps apparaît comme paramètre, deux sont largement utilisées : Les Automates Temporisés et les Réseaux de Petri Temporels (ou Time Petri Nets), introduits dans *[Mer 74]*.

Les réseaux temporels sont obtenus à partir des réseaux de Petri en associant deux dates min et maxà chaque transition. Supposons que t devienne franchissable pour la dernière fois à la date θ, alors t ne peut être tirée avant la date $\theta+$ min et doit l'être au plus tard à la date $\theta+$ max , sauf si le tir d'une autre transition a désensibilisé t avant que celle-ci ne soit tirée. Le tir des transitions est de durée nulle. Les réseaux temporels expriment nativement des spécifications «en délais». En explicitant les débuts et les fins d'actions, ils peuvent aussi exprimer des spécifications «en durées». Leur domaine d'application est large.

VI.2.1 Définition : *[Ber 05]*

Soit I^+ l'ensemble des intervalles réels non vides à bornes rationnelles non négatives.

Pour i$\in I^+$, $\downarrow$i désinge sa borne inférieure et $\uparrow$i sa borne supérieure (si i est borné) ou ∞(sinon). Pour tout $\theta \in \mathbb{R}^+$, i $\div\theta$ désignera l'intervalle $\{x\text{-}\theta$ / x $\in$i$\wedge$ x $\geq\theta$ $\}$.

Un réseau temporel (ou Time Petri net, ou TPN en abrégé) est un tuple <P,T,Pre,Post,M_0,Is> , dans lequel M_0 est un réseau de Petri et Is est une fonction appelée Intervalle Statique.

L'application Is associe un intervalle temporel Is(t) à chaque transition du réseau.

Exemple :

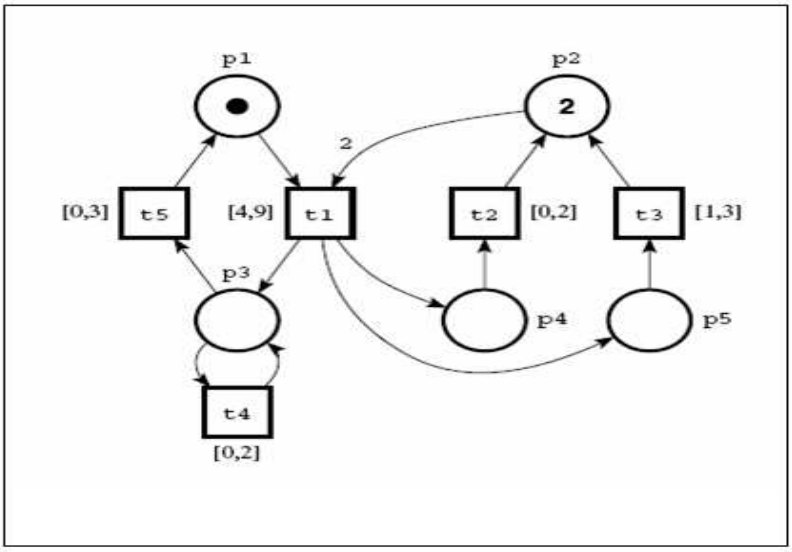


Figure1.4 : Un réseau temporel

IV.2.2 Evolution d'un TPN:

Un état d'un réseau temporel est un couple e=(M,I) dans lequel M est un marquage et I :⟶ T I^+ une fonction qui associe un intervalle temporel à chaque transition franchissable par le marquage M .

L'état initial est $e_0=(M_0,I_0)$ où I_0 est la restriction de Is aux transitions sensibilisées par le marquage initial M_0. Toute transition sensibilisée doit être tirée dans l'intervalle de temps qui lui est associé. Cet intervalle est relatif à la date de franchissement de la transition.

Franchir t à la date θ depuis e=(M,I) , est donc permis si et seulement si *[Ber 05]* :

$M \geq Pre(t) \wedge \theta \in I(t) \wedge (\forall k \neq t)\ (M \geq Pre(k) \Rightarrow \theta \leq \uparrow I(k))$

L'état e'=(M',I') atteint depuis e par le tir de t à la date θ est alors déterminé par :

1.M'=M-Pre(t)+Post (t) (comme dans tous les RDP) ;

2.Pour toute transition k franchissable à partir de M' :

$$I'(k) = \begin{cases} I(k) \div \theta & \text{si } k \neq t \text{ et M-Pre(t)} \geq \text{Pre (k)} \\ Is(k) & \text{sinon} \end{cases}$$

Notons que, dans les réseaux temporels, l'écoulement du temps ne peut qu'augmenter l'ensemble des transitions tirables, mais en aucun cas le réduire.

VI.3 Les réseaux de Petri stochastiques :

La théorie des réseaux de Petri a été trés utilisée pour la modélisation et l'évaluation des performances des systèmes parallèles. Seulement, avec un RdP simple, seule l'évaluation qualitative est possible, puisque aucune considération temporelle nâest prise en compte lors de la modélisation. Ainsi, un nouveau modèle de RdP a été introduit :les réseaux de Petri stochastiques (RdPS), qui associent à chaque transition, une variable aléatoire exponentielle représentant le délai de franchissement .

VI.3.1 Définition :

Un RdPS est un couple S=<N,W> où :
N=<R ,M_0> est un RdP marqué avec M_0 son marquage initial.
W=$\{\lambda_i \in \mathbb{R}$ / λ_i est le taux de franchissement de la transition $t_i\}$.
L'évolution d'un RdPS est décrite par la suite (M_0,M_1,M_2 , ...,M_n,...) des marquages successifs aux instants de franchissement (instants d'observation) que nous supposons être d_0=0,d_1,...,d_n,... tel que $d_{i-1}<d_i$ $\forall$ i $\in$ N.
Les états possibles du réseau S à chaque instant d_n sont les éléments d'un ensemble E avec E=$\{M_0,M_1,...,M_n,...\}$.

VI.3.2 Analyse des RdPS :

L'analyse d'un RdPS a pour but de révéler les deux aspects du système : l'aspect qualitatif et l'aspect quantitatif.

L'aspect qualitatif qui définit les propriétés qualitatives du système à analyser : vivacité, bornitude ... etc.
L'aspect quantitatif qui a pour but, le calcul des paramètres de performance du système modélisé.

En effet un théorème a été introduit par Zenie *[Zen 87]* pour définir l'isomorphisme entre les RdPS et les chaînes de Markov, qui permet d'effectuer l'évaluation des performances d'un RdPS. Cependant, appliquer les méthodes d'analyse des chaînes de Markov sur les RdPS nécessite la vérification d'une forte propriété, dite condition d'ergodicité, qui assure l'existence d'un régime stationnaire permettant le calcul d'une solution stationnaire de la chaîne de Markov.

Proposition 1: ***[Zen 87]***

Si un RdPS est borné, vivant et admet l'état initial comme état d'accueil alors il est ergodique.

Résultat 1: ***[Zen 87]***

Un RdPS borné, quasi vivant et tel que son graphe de marquage est fortement connexe est ergodique.

Résultat 2:

Une condition suffisante de l'ergodicité d'un RdPS est que son graphe des marquages accessibles soit borné et admette l'état initial comme état d'accueil.

Calcul de la solution stationnaire et évaluation des paramètres de performances:

En vérifiant la propriété d'ergodicité pour un RdPS, il est possible de calculer sa distribution stationnaire (ou permanente). Cette distribution est obtenue en utilisant la matrice génératrice Q construite à partir des taux de transitions entre les états (marquages).

Pour évaluer les performances du RdPS à analyser, on procède alors au calcul du vecteur des probabilités stationnaires en résolvant un système d'équations, puis au calcul des paramètres quantitatifs en utilisant le vecteur obtenu.

Calcul de la matrice génératrice:

Les éléments de la matrice Q sont calculés à l'aide du graphe des marquages accessibles. Les éléments sont égaux aux taux de transitions entre états ou marquages.

La matrice est carrée, d'ordre égal au nombre de marquages dans le graphe d'accessibilité. Les taux de transitions sont définis comme suit:

- Ayant deux marquages accessibles distincts M_i et M_j tel que: M_i [s > M_j et s=$t_{j1}t_{j2}...t_{jk}$

Alors: Q [i,j] = $\sum_{i=1}^{K}\lambda_{ij}$, et Q[i,i] = -$\sum_{i\neq j}^{K}$Q[i,j]. Où λ_{ij} désigne le taux de la transition t_{ij}.

Calcul du vecteur des probabilités stationnaires :

Le vecteur des probabilités stationnaires, de dimension égale au nombre de marquages du système modélisé est calculé, en résolvant le système d'équations suivant:

$$\begin{cases} \pi Q = 0 \\ \|\pi\| = \sum_{i=1}^{n} \pi_i = 1 \text{ ou } \|\pi\| \text{ est la norme de } \pi. \end{cases}$$

Calcul des paramètres de performance :

En utilisant le vecteur des probabilités stationnaires, il est possible d'obtenir plusieurs paramètres de performances ; tels que :

- Fréquence moyenne de franchissement des transitions.
- Nombre moyen de marques dans chaque place.
- Temps moyen de séjour des marquages dans chaque place.

- Probabilité d'un évènement A défini à travers une condition.

A chacun de ces paramètres, une formule de calcul est donnée comme suit :

1.Nombre moyen de marquages dans une place p:

$$M^*(p) = \sum_{Mi \in GMA} \pi_i \, . \, M_i(p)$$

2. Fréquence moyenne de franchissement d'une transition t_i:

$$\theta^*(t_i) = \sum_{Mj \in GMA} \pi_j \, \lambda_i(M_j)$$

Où: λ_i (M_j) est le taux de franchissement de la transition t_i dans M_j.

3. Temps moyen de séjour d'une marque dans une place p:

$T^*(p) = M^*(p) / (Post\ (p, .).\ \theta\ ^*)$

Où:

- θ *: est le vecteur des fréquences moyennes de franchissement.
- Post (p, .): est la ligne de la matrice d'incidence arrière post correspondant à la place «p».

- M*: est le vecteur des nombres moyens de marques.

4. Probabilité d'un évènement A défini à travers une condition :

Prob(A) = $\Sigma\pi_i$, la somme est effectuée sur les indices des marquages où la condition est satisfaite.

Exemple d'application *[Flo 85]* :

Soit le RdPS suivant :

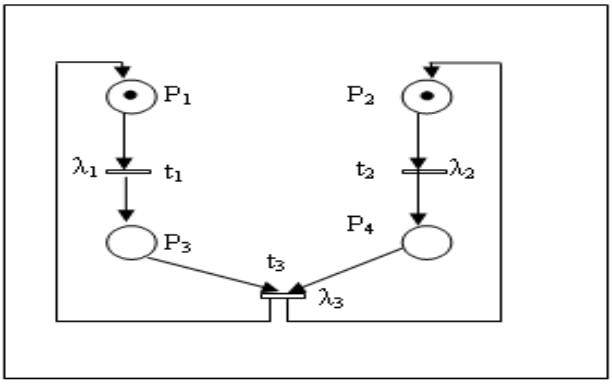


Figure 1.5 : RdPS à analyser

Les transitions t_1, t_2, t_3 ont respectivement comme taux λ_1, λ_2, λ_3.
Le graphe des marquages accessibles résultant est le suivant:

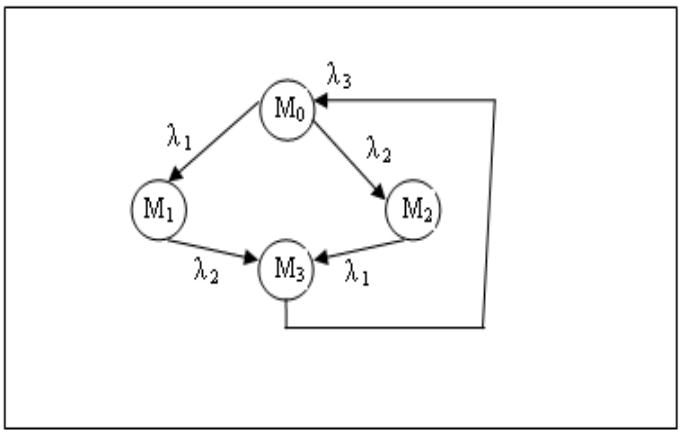


Figure 1.6 : Graphe d'accessibilité du RdPS

Le graphe d'accessibilité est borné, quasi vivant (toutes les transitions sont tirables) et fortement connexe. Donc, d'aprés les résultats précédents, le réseau est ergodique.

La matrice génératrice Q est alors la suivante:

$$Q = \begin{pmatrix} -(\lambda_1+\lambda_2) & \lambda_1 & \lambda_2 & 0 \\ 0 & -\lambda_2 & 0 & \lambda_2 \\ 0 & 0 & -\lambda_1 & \lambda_1 \\ \lambda_3 & 0 & 0 & -\lambda_3 \end{pmatrix}$$

On trouve le vecteur suivant des probabilités stationnaires:

$$\pi = \begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \end{pmatrix} \quad \text{Où} \quad \begin{cases} \pi_1=(\lambda_1 \lambda_2 \lambda_3)/A \\ \pi_2=(\lambda_1^2 \lambda_3)/A \\ \pi_3=(\lambda_2^2 \lambda_3)/A \\ \pi_4=(\lambda_1 \lambda_2 (\lambda_1+\lambda_2))/A \\ A=\lambda_1 \lambda_1 \lambda_3+\lambda_1^2 \lambda_3+\lambda_2^2 \lambda_3+\lambda_1 \lambda_2(\lambda_1+\lambda_2) \end{cases}$$

En appliquant les formules de calcul des paramètres de performances, on aura:

1-Vecteur des fréquences moyennes de franchissement des transitions:

$$\theta^* = \begin{pmatrix} \lambda_1 (\pi_1+\pi_3) \\ \lambda_2 (\pi_1+\pi_2) \\ \lambda_3 \pi_4 \end{pmatrix}$$

2-Vecteur des marques moyennes de chaque place:

$$M^* = \begin{pmatrix} \pi_1+\pi_3 \\ \pi_1+\pi_2 \\ \pi_2+\pi_4 \\ \pi_3+\pi_4 \end{pmatrix}$$

3-Vecteur des temps moyens de séjour dans les places:

$$T^* = \begin{pmatrix} 1/\pi_1 \\ 1/\pi_2 \\ 1/\pi_1+\pi_3/(\pi_2(\pi_1+\pi_2)) \\ 1/\pi_2+\pi_3/(\pi_1(\pi_1+\pi_2)) \end{pmatrix}$$

VII. La génération de l'espace d'états:

VII.1 Définition: *[Kri 00]*

Un espace d'états est l'ensemble des états que peut prendre un système. Cet ensemble est généralement représenté par un graphe orienté avec un noeud pour chaque état et un arc pour chaque changement d'état possible.

Si l'espace d'états est fini, il peut être utilisé pour vérifier plusieurs propriétés telles que la vivacité, le blocage, la bornitude ... etc.

Un espace d'états pour un RdP est traditionnellement appelé graphe des marquages accessibles (GMA). Notons que l'espace d'états n'est pas utilisé par les RdP seulement mais par plusieurs autres formalismes de modélisation.

VII.2 L'exploration d'espace d'états:

Pour générer l'espace d'états d'un RdP, un algorithme a été proposé ***[Hav 99]***:

Dans cette approche un seul processus est responsable du calcul de l'espace d'états en suivant l'algorithme donné ci-dessous.

Algorithm 2 Algorithme séquentiel pour la génération de l'espace d'états

```
Declare S, U: set of state;
Declare i,j : state ;
Declare Q: matrix of real;
S ←{s init };
U← { s_init};
Q← 0;
while ∃ i ∈ U do
  for each j ∈N(i) do
    if j ∉ U ∪ S then
                U ←U ∪ {j};
                Q_i,j← Q_i,j +Rate(i,j);
    end if;
  end for;
  U←U \ {i};
  S ←S ∪{i};
end while;
return S;
```

Où:

S_{init}: L'état initial à partir duquel le calcul commence.

S: Ensemble de tous les états calculés, initialisé à s_{init}.

U: Ensemble des états non encore explorés, initialisé à s_{init}.

N(i): Fonction qui permet le calcul des états atteignables à partir d'un certain état i, appelée fonction d'état suivant.

Rate (i,j): Taux de franchissement permettant le passage de l'état i vers l'état j.

Q: Matrice génératrice obtenue(dans le cas stochastique).

L'avantage de cette méthode réside dans sa simplicité et la facilité de son implémentation, mais elle présente plusieurs inconvénients tels que le temps de réponse qui est généralement trés long, car l'exécution des instructions dépend de la cardinalité des ensembles S et N.

En plus, aprés avoir exploré et stocké S, il peut être utilisé pour vérifier certaines conditions sur les états ou sur l'existence des chemins dans le graphe d'accessibilité. Pour le premier cas, les vérifications demandent le contrôle de chaque état qui se réalise dans O(|S|) unités de temps. Par contre pour le deuxième cas, la complexité dépend de la question demandée où les tendances actuelles utilisent la logique temporelle pour spécifier les propriétés à vérifier.

VIII.Conclusion :

Dans ce chapitre, nous avons présenté les réseaux de Petri, qui ont été introduits comme un outil de modélisation des systèmes parallèles et asynchrones. Cependant, cet outil n'est suffisamment expressif pour évaluer le comportement des systèmes réels qui dépendent du temps. De ce fait, nous avons abordé aussi, les réseaux de Petri stochastiques et les réseaux de Petri temporels, qui constituent un outil puissant pour la formalisation des propriétés quantitatives des systèmes réels.

Nous avons présenté aussi un autre type de réseau de Petri qui sert à réduire la taille des modèles des systèmes complexes en utilisant la notion de marques colorées, il s'agit des RdP colorés (RdPC).

Comme ces différents outils peuvent modéliser des grands systémes complexes, ils génèrent par conséquent un espace d'états trés important en utilisant l'algorithme de génération séquentiel, ce qui cause le problème d'explosion combinatoire. Pour cela des approches de réduction pour l'exploration d'espace d'états ont été proposées, qui se basent sur quelques caractéristiques du système analysé tel que la symétrie, et qui seront exposées au chapitre suivant.

Chapitre2:
Méthodes de réduction

I. Introduction:

La génération d'espace d'états ou le graphe d'accessibilité des modèles à états discrets tel que les RdP est une étape nécessaire dans l'étude et l'analyse des propriétés quantitatives et qualitatives. Mais l'inconvénient majeur de cette génération est le problème d'explosion combinatoire, lorsque la taille de l'espace d'états peut augmenter exponentiellement par rapport à la taille du système, où même pour des systèmes relativement petits, il peut y avoir un nombre énorme ou même infini d'états accessibles, et cela pose un grand problème pour l'analyse des systèmes à temps réels. Ainsi plusieurs techniques de réduction, basées sur la nature du système modélisé, ont été proposées pour la construction et le stockage de l'espace d'états. Dans ce chapitre, nous en présentons quelques unes.

II. La symétrie: *[Jor 99]*

Plusieurs systèmes concurrents possédent un certain degré de symétrie, par exemple plusieurs systèmes sont construits à l'aide d'un ensemble de composants similaires. Cette structure symétrique influe évidemment sur l'espace d'états de ces systèmes. L'idée de base est de mettre en facteur cette symétrie et d'obtenir un espace d'états condensé qui est beaucoup plus réduit que l'espace d'états complet où toutes les propriétés peuvent être vérifiées.

Un exemple de système symétrique est donné dans la figure 2.1. Il représente un RdPC modélisant un protocole d'élection (commit protocol). Ce type de protocole peut être utilisé pour implémenter l'atomicité des transactions *[Cou 98]*.

La partie gauche du RdPC modélise la boucle exécutée par *le coordinateur* qui détermine si la transaction peut être lancée ou non, de l'autre côté la partie droite modélise la boucle exécutée par *les travailleurs*. Le protocole doit assurer qu'une transition est activée si et seulement si tous les travailleurs sont prêts à exécuter leur partie de la transaction. Les deux premières parties communiquent à l'aide d'un réseau de connexion.

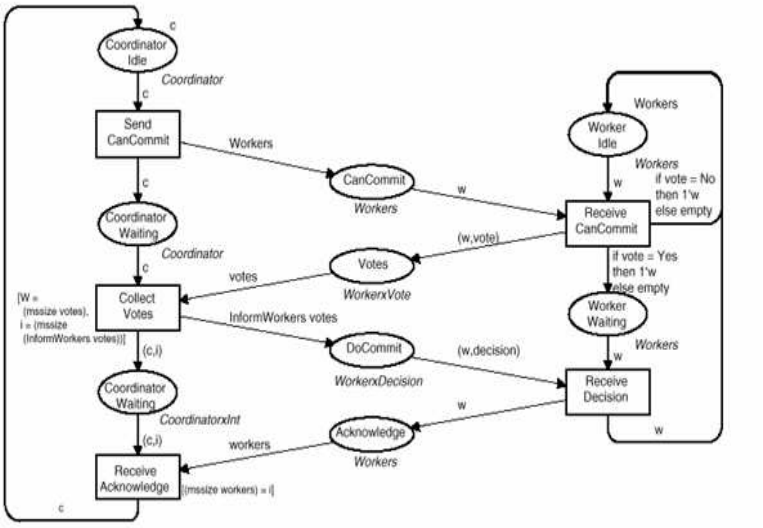


Figure 2.1 : Exemple de système symétrique.

Intuitivement ce protocole est symétrique dans le sens où il traite tous les travailleurs de la même manière, c-à -d tous les travailleurs se comportent de la même façon, ils sont alors symétriques, ce qui influe sur l'espace d'états de ce modèle (figure 2.3).

Considérons par exemple les deux marquages 3 et 5 qui correspondent aux états dans lesquels exactement un travailleur a reçu le message *CanCommit* et il a répondu par un message *No* au *coordinateur*. Ces deux marquages sont symétriques dans le sens où le marquage 3 peut être obtenu à partir du marquage 5 en échangeant l'identité du travailleur (même chose pour le quatrième et le sixième marquage). De plus, on peut remarquer que deux marquages symétriques (tels que le trois et le cinq) ont des ensembles de transitions d'entrée et de sortie symétriques.

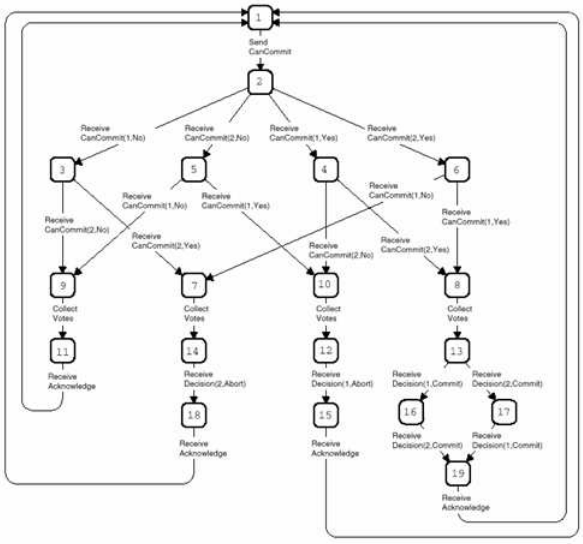


Figure 2.2 :L'espace d'états complet du protocole.

L'idée de base pour la construction de l'espace d'états condensé est de regrouper les marquages symétriques en classes d'équivalence.

La figure suivante présente l'espace d'états condensé pour l'exemple du protocole d'élection en prenant en considération que les deux travailleurs sont symétriques, c-à -d qu'on peut permuter leurs identités. Les noeuds et les arcs représentent maintenant les classes d'équivalence et les liens entre elles respectivement (par exemple le noeud 3 représente le troisième et le cinquième marquage de la figure précédente). L'espace d'états condensé contient 13 noeuds et 16 arcs, ce qui montre que la réduction obtenue en exploitant la symétrie est très importante *[Jor 99]*.

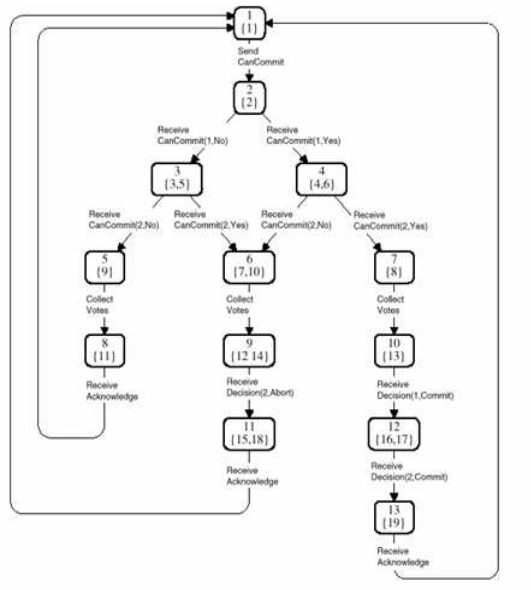


Figure 2.3 :L'espace d'états condensé du protocole

Formellement *[Jor 99]*, la symétrie est définie comme étant une application qui attribue à chaque couleur du modèle, un ensemble de permutation dans l'ensemble global des couleurs, tel qu'en remplaçant cette couleur par son ensemble de permutation, le comportement du modèle ne changera pas et il conservera ses différentes propriétés.

La symétrie a été appliquée pour analyser plusieurs systèmes qui n'ont pas pu être modélisés par l'approche traditionnelle (exemple des philosophes) Elle a permis une grande réduction dans l'espace d'états. Cependant son inconvénient majeur est qu'elle ne peut pas être appliquée sur tous les systèmes, car elle exige qu'ils soient symétriques, ce qui n'est pas toujours le cas.

III. Ensembles des Stubborn (résistants) : *[Kri 99]*

Plusieurs systèmes concurrents et distribués sont asynchrones et constitués d'un nombre de processus relativement indépendants qui se synchronisent ou communiquent occasionnellement seulement, d'où l'idée des ensembles de Stubborn, qui se base sur la génération d'un espace d'états réduit en évitant la représentation de toutes les actions possibles. La construction de l'espace d'états dans ce cas, suit la même procédure que la construction de l'espace d'états complet avec une exception : au lieu de travailler avec tout l'ensemble des transitions du système, le traitement s'effectue seulement sur un sous ensemble de transitions, appelé ensemble de Stubborn. Ce qui permet la réduction du nombre de marquages développés et par conséquent la taille de l'espace d'états.

Pour avoir une analyse correcte, l'ensemble des Stubborn doit être choisi tel que l'espace d'états obtenu ait les mêmes propriétés que l'espace d'états complet, en plus le choix de l'ensemble des Stubborn dépend des propriétés à analyser ou à vérifier.

Définition *[Kri 99]* : Soit (P,T,Pré,Post) un RdP, l'ensemble Stub$\subseteq$T est un ensemble de stubborn dans le marquage M si pour t$\in$ Stub, t vérifie :

1. Si $\exists$ t1$\in$ T : M[t1 $>$, alors $\exists$t2 $\in$Stub tel que M[t2 $>$.
2. Si$\neg$ M[t$>$ alors $\exists$ p $\in$ $\bullet$t tel que M(p) $<$Pré (p,t) $\wedge\bullet$p$\subseteq$ Stub.
3. Si M[t$>$ alors ($\bullet$ t)$\bullet$ $\subseteq$ Stub.

Les expériences ont montré une grande efficacité de cette technique, mais elle présente un inconvénient majeur qui est le calcul des ensembles de Stubborn qui peut être très difficile pour les RdP de grande taille.

Pour pallier à ce problème, des travaux ont proposé la combinaison de cette méthode avec d'autres méthodes de réduction telle que la symétrie *[Val 91]*.

IV. La modularité :

Une manière d'alléger le problème de l'explosion combinatoire dans les espaces d'états, est d'utiliser l'analyse modulaire qui prend l'avantage de la structuration en modules du système représenté. Les résultats donnés dans

[Pet 05], ont montré que l'analyse modulaire peut produire une réduction significative dans la taille de l'espace d'états.

Dans cette partie nous présenterons comme exemple, un algorithme donné dans *[Pet 05]*, permettant l'exploration de l'espace d'états pour un système temporel modulaire.

IV.1 Définitions :

Un RDP temporel modulaire est une paire (S, TF) où :

-1 S est un ensemble fini de modules tel que :

- Chaque module s∈S est un réseau de Petri temporel.
- les ensembles des noeuds correspondants aux modules sont deux à deux disjoints:$\forall s_1, s_2 \in \mathrm{S} : [s_1 \neq s_2\ (\mathrm{P}s_1 \bigcup \mathrm{T}s_1) \bigcap (\mathrm{P}s_2 \bigcup \mathrm{T}s_2) = \oslash]$
- P $= \bigcup_{s \in S} \mathrm{P}_S$ et T$= \bigcup_{s \in S} \mathrm{T}_s$ sont les ensembles de toutes les places et de toutes les transitions.

-2 TF $\subseteq 2^T \setminus \{\oslash\}$ est un ensemble fini des ensembles de transition de fusion. Les transitions de fusion permettent la synchronisation entre les différents modules.

IV.2 Exploration de l'espace d'états modulaire :

Dans l'analyse modulaire, on explore l'espace d'états de chaque module, puis on capture les points de synchronisation entre les différents modules dans un graphe de synchronisation. L'algorithme présenté ci-dessous permet l'exploration du graphe de synchronisation *[Pet 05]*.

Notons que le marquage M est enlevé de l'ensemble Waiting à chaque fois qu'il est exploré. La fonction *Explore* retourne un ensemble de triplets, dont le premier élément est un noeud de synchronisation, le deuxième est un marquage local atteignable à partir du premier élément et le dernier est une transition de fusion qui est localement franchissable à partir du deuxiéme élément.

Ainsi, chaque appel de la fonction *Explore* retourne un tel ensemble dans la variable *Trysynch*. A partir du résultat retourné pour tous les modules, on

Algorithm 3 Algorithme de génération graphe de synchronisation

```
Integer: System_ limit ← Value;
Integer: System_ time←0;
Set : Waiting ← ⊘ ; Node.Add(M_0) ;
While System_ time ≤System_ limit do
   Repeat
      ∀i: Trysynch_i ←Explore(S_i, Waiting, System_ time) ;
        Waiting ← ⊘;
        For all tf ∈TF do
              For all (M, M') such as (M, M_i', tf) ∈ Trysynch_I
              Do if M' [tf > M" at System_ time then
                Node.Add(M") ;
                Arc.Add(M,(M',tf, System_ time),M") ;
               End if
            End for
      End for
   Until stable
System_ timeï← System_ time+1 ;
End While
```

construit des paires de marquages globaux (M, M'), dont le premier élément est un noeud du graphe de synchronisation et le deuxième élément correspond à un marquage localement atteignable à partir du premier élément. Si le marquage local permet le franchissement d'une transition de fusion à l'instant courant, alors on ajoute le nouveau noeud et l'arc correspondant au graphe.

La fonction Node.Add(M) ajoute le noeud étiqueté M au graphe et l'état M à l'ensemble Waiting, s'il n'existe pas déjà . D'une manière similaire la fonction Arc.Add (M,t,M') ajoute un arc au graphe.

Exemple : Soient les deux modules A et B suivants :

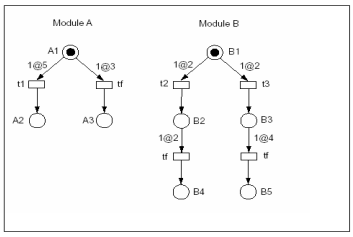


Figure 2.4 : Exemple de deux modules temporels

L'espace d'états obtenu est donné par :

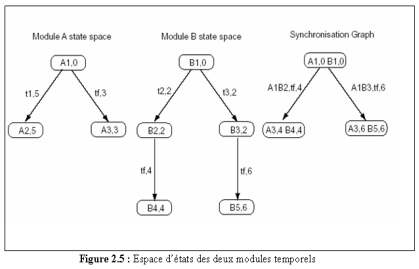


Figure 2.5 : Espace d'états des deux modules temporels

V. Compression et représentation symbolique :

La compression utilise des structures de données afin de représenter de façon concise des ensembles d'états. Les opérations se font alors sur des ensembles d'états plutôt que sur des états explicites. Cette technique est souvent qualifiée de symbolique. Une des structures qui utilisent cette technique, est la représentation par les diagrammes de décision binaire (BDD, de l'anglais Binary Decision Diagrams). L'utilisation des BDD est aujourd'hui tellement répandue que BDD est souvent utilisé comme synonyme de méthode symbolique.

Les BDD sont des diagrammes binaires qui servent à représenter efficacement une formule booléenne. En vérification, ils sont utilisés pour représenter un ensemble d'états. Un BDD est un graphe orienté acyclique, il se compose d'un ensemble de noeuds de décision intermédiaires, et deux noeds terminaux appelés 0-terminal et 1-terminal. Les chemins qui se terminent par le noeud 1-terminal sont les seuls pris en considération parmi tous les chemins présentés par le BDD *[Cia 02]*. Afin de pouvoir représenter un espace d'états S à l'aide des BDD, le modèle correspondant à cet espace, doit être décomposable en k sous modèles.

Par conséquent, l'espace d'états S sera obtenu par le produit de ces sous modèles : $S = S_k * ... * S_1$.

Exemple : (Pour k = 4)

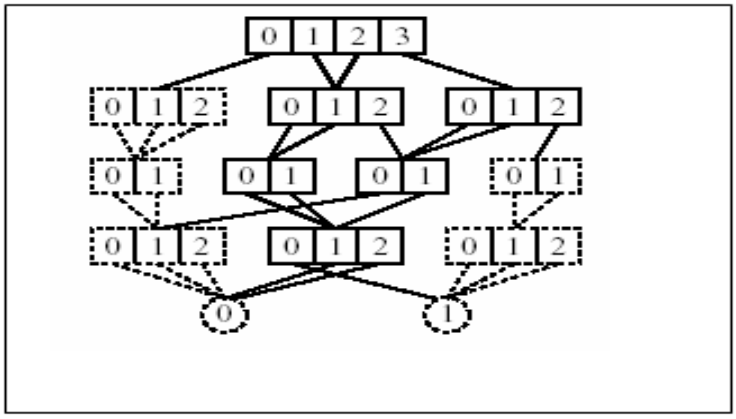


Figure 2.6 Un exemple de BDD

Le BDD donné dans la figure 2.6 est composé de quatre niveaux où chacun représente un sous modèle : S4 ={0,1,2,3}, S3 = {0,1,2}, S2 = {0,1} et S1 ={0,1,2} , l'espace d'états S obtenu dans ce cas, en suivant tous les chemins du niveau le plus haut jusqu'au noeud terminal 1, est donné par : S={1000,1010,1100,1110,1210,2000,2010,2100,2110,2210,3010,3110,3200,3201,3202, 3210,3211,3212}. L'utilisation des BDD a grandement augment la taille des systèmes pour lesquels il est possible de faire la vérification, passant de 2^{17} états plus de 2^{100} états.

VI. Conclusion :

Les techniques de réduction qui ont été utilisées pour la génération de l'espace d'états, sont des techniques qui exigent que le système modélisé ait une structure particulière, et par conséquent impose des limitations sur les types des systèmes qui peuvent être analysés.

Pour pallier ce problème d'autres méthode ont été proposées, parmi elles, celles qui utilisent la distribution de l'espace d'états sur plusieurs machines, ce qui fait l'objet du prochain chapitre.

Chapitre 3:
La distribution de l'espace d'états

I.Introduction:

L'exploration de l'espace d'états est une technique pratique pour l'analyse des systèmes concurrents et distribués. Son inconvénient principal est le problème de l'explosion combinatoire. Pour pallier à ce problème des techniques ont été utilisées, qui exploitent les avantages de calculs parallèles et distribués.

Dans cette approche, plusieurs machines coopèrent pour générer l'espace d'états et réaliser la vérification. Par conséquent l'espace d'états et les charges de calcul sont répartis entre les différentes machines.

II. La distribution de l'espace d'états: *[Cia 02]*

Les méthodes distribuées utilisent un ensemble de processus (processeurs) interconnectés à lâaide d'un réseau, afin d'augmenter la capacité de stockage et d'accélérer le temps de résolution.

II.1 La génération de l'espace d'états:

En supposant que l'on a N processus et un espace d'états S, une manière naturelle d'effectuer cette exploration est de répartir l'espace d'états en appliquant une fonction de hashage définie comme suit : hash: S $\rightarrow$ {0,...,N-1},

Où hash (i) est le processus propriétaire (owner) de l'état i c-à-d le processus responsable de l'exploration et du stockage de l'état i, ce qui permet de définir une partition de l'ensemble S en N sous ensembles: $S_{[n]}$= {i: hash(i) = n} pour $0 \leq n < N$, plus précisément l'ensemble S[n] correspond au sous espace d'états exploré par le processus n.

L'algorithme réparti permettant cette génération distribuée est défini comme suit *[Cia 02]*:

Où chacun des N processus effectue des tâches analogues à celles de la génération séquentielle, avec les différences suivantes:

Algorithm 4 DistributedExplicitExplorer (n :process) : Set of state

Declare S[n], U[n]: set of state;
Declare A[n]: set of arcs;
Declare i, j: state;
Declare m: process;
1. if hash($s^{initial}$) = n then $U_{[n]} \leftarrow$ { $s^{initial}$ }; else $U_{[n]} \leftarrow \emptyset$;
   end if;
2. $S_{[n]} \leftarrow \emptyset$;
3. while " not received terminate message " do
4. while $\exists$ i $\in U_{[n]}$ do
5. for each j $\in$ succ(i) do
6. m $\leftarrow$ hash (j);
7. if m $\neq$ n then
8. SendState (m,j); SendArc (m,j$\rightarrow$ i);
9. else if j$\notin U_{[n]} \cup S_{[n]}$then
10. $U_{[n]} \leftarrow U_{[n]} \cup$ {j}; $A_{[n]}$ $A_{[n]} \cup$ {j$\rightarrow$ i };
11. end if;
12. end for;
13. $U_{[n]} \leftarrow U_{[n]} \setminus$ {i};
14. $S_{[n]} \leftarrow S_{[n]} \cup$ {i};
15. end while;
16. $U_{[n]} \leftarrow_{[n]} \cup$ ReceiveState $\setminus S_{[n]}$; $A_{[n]} \leftarrow A_{[n]} \cup$ ReceiveArcs;

Le processus qui doit initier le calcul de l'espace d'états, est le processus propriétaire de l'état initial s initial, ayant comme indice hash($s^{initial}$). *[Instruction 1]*

Un état j n'est traité localement par un processus, que s'il est son propriétaire, sinon il sera envoyé au processus propriétaire. *[Instructions de 5 à 8]*

Les états envoyés à un processus à partir des N-1 autres processus, sont insérés dans l'ensemble des états non encore explorés $U_{[n]}$.*[Instruction 16].*

Lorsque n envoie l'état j à m (m$\neq$ n); il lui envoie aussi l'arc reliant l'état j à son prédécesseur i.

La terminaison de cette génération peut être détectée lorsque tous les

processus sont inactifs (l'ensemble $U_{[n]}$est vide) et qu'aucun message n'est en transit.

II.2 Complexité et efficacité de l'algorithme:

Les performances des algorithmes distribués dépendent de la charge de calcul à effectuer dans chaque noeud du réseau *[Cia 02]* et ils dépendent fortement du coût de communication, que l'on peut calculer à partir du nombre de messages circulant entre les différents noeuds du réseau. Dans ce qui suit, nous citons quelques critères correspondant aux performances de l'exploration distribuée:

- l'équilibre spatial: La mémoire demandée pour stocker S[n] doit être approximativement la même pour tout processus n, cela est fondamental, puisque la mémoire est la ressource principale qui cause le problème de goulet d'étranglement dans la génération de l'espace d'états. Ceci est appelé l'équilibre spatial, et qui est défini comme suit *[Cia 02]*:

Max $0 \leq m,n < N$ ($| S_{[n]} |$ / $| S_{[m]} |$).

Une bonne répartition doit avoir un équilibre spatial proche de 1.

- L'équilibre temporel: Les processus aussi doivent être actifs le plus longtemps possible, en garantissant une charge équilibrée entre les processus ce qui est appelé l'équilibre temporel.

- La localité: Le nombre d'arcs reliant deux états associés à deux propriétaires différents, ce qu'on appelle les arcs traversants (cross arcs), doit être aussi petit que possible. Ce critère est mesuré à l'aide de l'expression suivante:

$$0 < \frac{\sum_{i \in S} |\{ j \in \mathbb{N}(i) \ : hash\,(i) \neq hash(j)\}|}{\sum_{i \in S} |N(i)|} \leq 1$$

Pour mieux illustrer la différence entre l'équilibre spatial et l'équilibre temporel, on considère le cas extrême de la figure suivante:

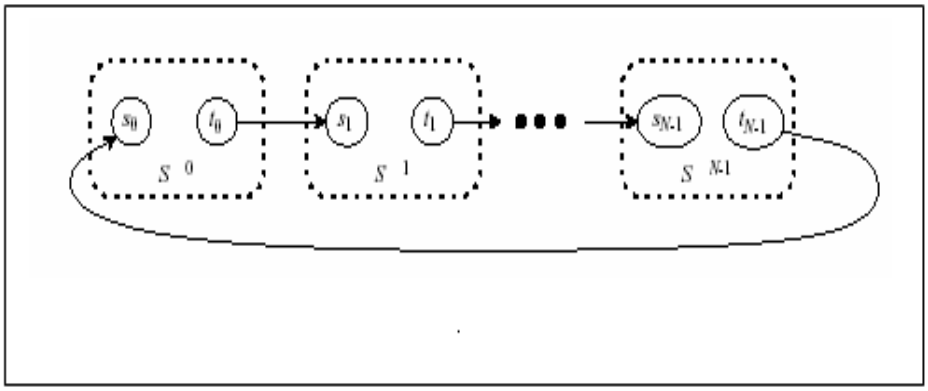


Figure 3.1 : Un cas extrême pour l'algorithme distribué.

Dans ce cas seulement un arc $t_i \rightarrow s_{(i+1)} \bmod N$, connecte l'ensemble des états du processus i (S^i) avec celui du processus i+1 (S^{i+1}), avec $s^{initial} \in S^0$ et t_i est le dernier état exploré dans S^i. Dans ce cas, l'algorithme distribué va être exécuté d'une manière séquentielle c-à-d avec un équilibre temporel faible, l'ensemble des états est réparti d'une façon régulière sur l'ensemble des processus c-à-d avec un équilibre temporel élevé et un nombre d'arcs traversants trés réduit.

II.3 Choix de la fonction de répartition (hashage):

Selon les critères de performances de l'algorithme de distribution on a deux manières de définir la fonction de hashage : statique ou dynamique:

a. Définition d'une fonction statique:

Dans ce cas la fonction de hashage (hash) est généralement définie comme suit:

hash (i)=(i[o]+ p i[1]+... p^{r-1}i[r-1]) mod N

Où p est un nombre premier et i[0]...i[r-1] sont le nombre de jetons dans r places du RdP.

Ces paramètres sont fixés par l'utilisateur selon les propriétés à vérifier. La sélection des sous ensembles de places sur lesquels la définition de hash est basée, peut influer sur la qualité de la répartition résultante. Des expériences ont été réalisées en utilisant 6 processus *[Cia 02]*, ont atteint 1.33 d'équilibre spatial et un équilibre temporel de 0.82.

L'avantage de laisser l'utilisateur définir la fonction de répartition est d'avoir la possibilité de spécifier ses buts, par exemple en prenant p=1 et en sélectionnant r places où les jetons peuvent seulement augmenter ou diminuer de 1 à chaque franchissement, la communication entre processus se réalise seulement entre les processus d'indices contigus, c-à-d de n à (n-1) mod N ou à (n+1) mod N. Cette propriété est trés importante lorsque l'interconnexion entre les processus se fait à travers un réseau en anneau où seulement des paires de processus adjacents peuvent communiquer.

En plus un autre point important de cette technique est que le franchissement de n'importe quelle transition à partir d'un marquage i, non reliée à aucune des r places, donne un marquage j appartenant au même processus propriétaire que i, c-à-d hash(i)= hash(j). Cette propriété peut être exploitée pour minimiser le nombre d'envois de messages entre les différents processus en prenant lâensemble approprié des r places.

Par contre un mauvais choix peut présenter quelques inconvénients: par exemple choisir r places constituant un invariant de la forme: i[o]+i[1]+... i[r-1] = C, donne comme processus propriétaire de tous les marquages (états) le processus C mod N et par conséquent tous les autres processus restent inactifs.

b. Définition d'une fonction dynamique *[Cia 02]* :

La définition statique demande l'assistance de l'utilisateur qui fournit des informations additionnelles, mais comme on a déjà vu, cela peut conduire à un déséquilibre spatial, où la majorité des états sont associés à un sous ensemble de processus ou bien à un déséquilibre temporel où seulement quelques processus sont actifs en même temps. Par conséquent une deuxième approche a été proposée dans laquelle la définition de hash peut être ajustée d'une manière dynamique. Ainsi une deuxième fonction intermédiaire est utilisée, appelée Classe.

Cette fonction est définie comme suit :
Classe: S $\rightarrow$ {0, ... C-1}
Où C est un entier assez grand par rapport à N (disons C=100N).
Dans ce cas la fonction hash est définie comme suit:
hash : {0,...C-1} $\rightarrow$ {0,...N-1}
Lorsque C est entre mille et dix milles, l'application hash peut être stockée sous forme d'un tableau de taille C dont les entrées ont des valeurs de 0 à N-1, il reste donc la définition de la fonction Classe qui n'est pas triviale.

Pour cela on utilise un ordre lexicographique entre les processus. En premier lieu chaque processus construit un même ensemble de C états différents, appelé ensemble de contrôle, en utilisant une recherche en profondeur d'abord initialisée avec le même chemin d'exploration d'espace d'états, où les transitions sont choisies avec les mêmes probabilités.

Les processus classent les états explorés dans un tableau (Search) et ils allouent un autre tableau (Class2proc) de taille C+1 initialisé comme suit: Class 2 proc [i] = i mod N, ensuite chacun d'eux exécute l'algorithme de l'exploration distribuée (donné dans la figure 2.5) où:

- Chaque processus n initialise son ensemble U[n] avec les états qui se trouvent à la position i définie par: [C/N] * n $\leq$ i $<$ [C/N] * (n+1), puis il commence les itérations usuelles.

- Lorsqu'un processus a besoin de déterminer le processus propriétaire d'un état j, il fait la recherche de ce dernier dans le tableau Search.

- Si j est trouvé, alors cet état est déjà connu son propriétaire,

- Sinon on prend l'indice x $\in$ {0,...,C-1} où la recherche a échoué pour identifier le processus m propriétaire de j, qui est donné par : m = Class2Proc [x] , si m=n alors le processus n vérifie si j est déjà stocké dans $U_{[n]} \cup S_{[n]}$, sinon il envoie j au processus m.

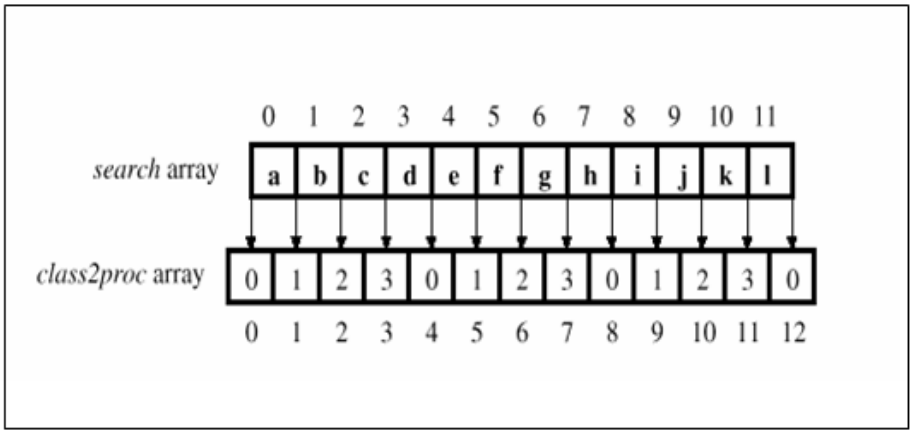


Figure 3.2 : Exemple des applications hash et class2proc.

Par exemple en prenant N= 4, C=3N et l'ensemble de contrôle contenant les états {a,...,l}, les ensembles U[0], U[1], U[2] et U[3] vont être initialisés avec {a,b,c}, {d,e,f}, {g,h,i}et {j,k,l} respectivement. Si un processus veut identifier le processus j propriétaire de l'état m qui ne figure pas dans le tableau search, alors il posera j = class2proc[12]=0.

L'aspect dynamique de cette approche consiste à pouvoir arrêter le processus de génération d'espace d'états périodiquement pour vérifier l'équilibre spatial ou l'équilibre temporel en comparant la taille de $U_{[n]} \cup S_{[n]}$ ou $U_{[n]}$ seulement de tous les processus et si un déséquilibre est détecté, il peut être corrigé avec un réarrangement dynamique de l'allocation des classes aux processus en redéfinissant l'application hash.

L'avantage de cette technique est de libérer l'utilisateur de la définition de la fonction de distribution, de plus les réarrangements effectués pendant l'exploration garantissent un meilleur équilibre. Mais d'un autre côté, ces réarrangements peuvent ralentir l'exécution, ce qui influe négativement sur

le temps de réponse.

II.4. La génération du processus stochastique: *[Cia 02]*

Lorsqu'on veut étudier le comportement quantitatif d'un certain modèle, on doit construire son processus stochastique à partir de l'espace d'états déjà construit.

Si le système correspond à une chaîne de Markov à temps continu, on construit alors la matrice génératrice Q où $Q^{s,s'}$ est le taux de transition de l'état s vers l'état s', pour s≠s'(s, s' ∈ S). Les entrées de la diagonale de Q sont données par la négation de la somme des éléments de la ligne correspondante.

$Q^{s,s}$=-$\sum^{s\in S, s\neq s'} Q^{s,s'}$

L'algorithme de cette génération pour un espace d'états distribué est comme suit *[Cia 02]*:

Où Rate (i, j) retourne le taux de franchissement de la transition permettant le passage de l'état i vers l'état j.

À la fin de l'algorithme, tout processus n contient les états $S_{[n]}$ = {s∈S et hash(s)=n} et les entrées de Q correspondant aux arcs qui atteignent ces états notées $Q_{.,S_{[n]}}(Q_n)$.

Un processus coordinateur fait l'union des différentes matrices $Q_{.,S_{[n]}}(Q_n)$ générées afin de construire la matrice génératrice finale Q.

Notons qu'à ce niveau, on peut aussi transmettre les différents taux de franchissement directement au coordinateur afin qu'il construise la matrice génératrice Q au fur et à mesure.

Au niveau implémentation, la communication entre les processus se fait à travers le passage des messages. Le flot de contrôle est donné par un ensemble de messages d'accusation, un processus émetteur ne peut envoyer un message que lorsqu'il reçoit l'accusé de ses A derniers messages envoyés (où A est un paramètre dont la valeur dépend de la capacité du buffer), ce qui permet d'éviter le blocage qui peut se produire lorsque le buffer est complet.

Algorithm 5 La génération distribuée de la matrice Q

```
Declare S[n], U[n]: set of state;
Declare Qn array of real;
Declare i, j: state;
Declare m: process;
   if hash(s^initial) = n then U_[n] ← { s^initial }; else U_[n]←∅;
   end if;
 S_[n]← ∅;
 Q_n← 0;
 while " not received terminate message "do
    while ∃ i ∈U_[n] do
      for each j ∈ N(i) do
        m ←hash (j);
        if m ≠  n then
          SendState (m,j); Send (m,Rate(i, j));
        else if j∉U_[n]∪ S_[n] then
           U_[n] ←U_[n]∪ {j}; Q_n(i,j) ← Q_n(i,j) +Rate(i,j);
              end if;
        end if;
      end for;
    U_[n] ← U_[n] \ {i};
    S_[n] ← S_[n] ∪ {i};
   end while;
U_[n] ← U_[n] ∪ ReceiveState \ S_[n]; Q_n ← Q_n +ReceiveRates;
```

A cause de la grande potentialité du nombre d'états envoyés entre processus, chaque paire (état, arc) sera stockée dans le buffer de l'émetteur, pour l'assembler et l'envoyer dans un même message que dâautres paires.

La taille du buffer est un paramètre de compilation, à chaque fois que la taille augmente, plus d'états et d'arcs seront assemblés dans un seul message et par conséquent moins de messages seront envoyés, ce qui permet de réduire la charge dans le réseau..

III. Conclusion:

La génération distribuée de l'espace d'états peut se faire en utilisant un

réseau de machines. Ce qui permet de résoudre le problème de l'explosion combinatoire de l'espace d'états pour les systèmes de grande taille. Selon le nombre de machines et leurs capacités, des systèmes de tailles importantes peuvent être maintenant analysés. Les performances de l'algorithme distribué sont fortement liées à la fonction d'attribution des états aux machines.

Dans le prochain chapitre, nous présentons quelques algorithmes permettant d'analyser certaines propriétés dans un espace d'états distribués.

Chapitre 4:
Analyse distribuées des propriétés

I. Introduction :

Le problème qui se pose après distribution de l'espace d'états, est comment peut-on analyser cet espace et vérifier ses propriétés, qualitatives et quantitatives, sur l'ensemble des machines. La majorité des travaux qui se sont intéressés à cette vérification, utilisent les méthodes formelles telle que la logique temporelle et le modèle Checking *[Bar 01],[Ler 99]*.

De ce fait, le but de cette partie est d'exploiter les ressources de stockage et de calcul d'un ensemble de machines, interconnectées en un réseau local et communiquant par envoi de messages, pour effectuer l'analyse de performances de systèmes de taille importante, et en plus précisément la vérification de la vivacité et la bornitude.

En effet la vérification de ces deux propriétés, qui sont des propriétés désirables par plusieurs systèmes, constitue un problème intéressant de leur importance d'une part et leur complexité d'autre part. Tel qu'elle nécessite la génération de l'espace d'états complet.

La vrification de la vivacité et de la bornitude garantit l'ergodicité du modèle, ce qui permet de passer à la deuxième étape qui consiste en la reconstitution de la matrice génératrice pour le calcul des paramètres de performance.

II. L'analyse distribuée des propriétés :

II.1 Vérification de la vivacité *[Bou 07]*:

Une transition t est dite vivante si en tout état du système, on peut trouver une séquence d'états telle que t soit toujours franchissable.

Vérifier qu'une transition est vivante, revient à vérifier que cette transition est franchissable dans toute composante fortement connexe (CFC) terminale du graphe. Ce qui demande de déterminer les différentes CFC du graphe, puis désigner celles qui sont des CFC terminales.

Une CFC est dite terminale si aucun sommet de celle-ci ne possède de successeurs dans une autre CFC.

L'algorithme qui a été utilisé pour déterminer les différentes CFC est celui de Tarjan *[Tar 72]* qui effectue itérativement les actions suivantes :

1. Une numérotation des sommets en profondeur d'abord ;

2. Calcul des points d'attache ;

3. Détermination d'une CFC.

La première étape qui consiste en la numérotation des sommets distribuée peut se faire de la manière suivante:

- Le processus possédant le sommet correspondant à l'état initial commence la numérotation suite à la réception d'un message du processus coordonnateur (le processus qui initie la génération des états).

- Lorsqu'un processus a numéroté un sommet, il passe à l'un de ses successeurs (profondeur d'abord), si le successeur s_i d'un sommet appartient à un autre processus, il lui envoie le message avec le couple *(s_i, Num)* pour que ce dernier reprenne la numérotation à partir du sommet s_i.

- Lorsqu'un processus reçoit un sommet à numéroter, s'il n'est pas déjà numéroté, il lui affecte le numéro reçu *Num*, sinon il renvoie un message de type *backtrack* au processus initial.

- Lorsque la numérotation d'un sous arbre dont l'origine est reçue d'un processus distant est terminée, un message de backtrack est transmis à celui-ci.

Le code exécuté par le processus pi pour commencer la numérotation, à partir du sommet s reçu du processus pj est:

Algorithm 6 Numérotation des sommets.

```
Numerotation()
begin
NotEnd = True ;
while (¬ Empty(P)) and (NotEnd) do
   if (Top(P) ≠ backtrack)  then
        s ← Depiler(P) ;
        if (hash(s) =i) then
             s.Num = Num ; Num++ ;
             for all s' ∈ succ(s) do Empiler(s') ;
        else SEND(hash(s), s, Num) ;
       endif
   else SEND(pj, backtrack) ;
        NotEnd ←False ;
   endif
enddo
end.
```

Ce code est exécuté lorsque le processus *pi* reçoit un nouveau sommet *s* à numéroter ou bien lorsqu'il reçoit un *backtrack*. Ã la réception d'un message, en l'occurrence un état, le processus est interrompu et une procédure *MessageHandle*r est déclenchée afin d'effectuer le traitement adéquat du message reçu :

Algorithm 7 La procédure MessadeHandler

```
Handler(message)
begin
⋮
if Message.Type = StateNum then
     s = message.s ;
    if (s.Num ≠ Null) then SEND(pj, Backtrack) ;
    else Num ← Message.Num ;
        s = Message.s ;
        Empiler(pj, Backtrack) ;
        Empiler(s) ;
        Numerotation() ;
     endif
endif
if message.Type = Backtrack then
  Numerotation() ;
endif
.
end
```

Le point d'attache at(x) d'un sommet x est égal au minimum parmi :

- *Num(x)*
- *Le Num(y)* si y est un successeur de x et que *Num(y)* $<$ *Num(x)*
- *at(y)* si y est un successeur de x et que *Num(y)* $>$ *Num(x)*

Pour les deux premiers cas, le calcul des points d'attache se fait au fur et à mesure de la numérotation. Par contre pour le dernier point, le calcul des points d'attache doit se faire en parcours arrière, une pile spécifique est utilisée pour contenir les couples de sommets *(x, y)* tels que *Num(x)* $<$ *Num(y)* en prévision de mettre à jour le point d'attache de x.

Lorsque la numérotation est terminée, si le couple de sommets *(x, y)* se trouve au sommet de la pile, alors nécessairement on connaît déjà le point d'attache de y, on peut alors mettre à jour le point d'attache de x.

Dans le cas distribué, deux cas se présentent :

1. x et y sont tous les deux au niveau d'un même processus, dans ce cas la mise à jour du point d'attache se fait localement.

2. L'un des deux sommets x ou y est distant, dans ce cas les deux processus propriétaires de x et y doivent avoir le couple *(x, y)* dans leur pile respective. Ainsi, le processus propriétaire de y envoie le point d'attache de ce sommet au processus propriétaire de x pour que celui-ci mette -à -jour le point d'attache du sommet x.

L'étape suivante consiste à déterminer un sommet x candidat pour engendrer une composante fortement connexe, ce sommet est tel que:

$$Num(x) = at(x) = Max\{Num(y)/Num(y) = at(y)\}.$$

Ce sommet est recherché par tous les processus en parallèle, puis envoyé au coordonnateur qui en déterminera celui qui correspond au max. Le sous arbre dont la racine est x engendre alors une CFC, une nouvelle exploration parallèle des sommets du sous arbre permet alors de délimiter cette CFC.

II.2. Vérification de la bornitude:

On s'int'resse dans cette partie à la vérification distribuée de la bornitude dans un espace d'états distribué.

Comme on l'a déjà défini, un réseau est dit borné si toutes ses places sont bornées, c-à -d elles ont un nombre limité de marques dans tout état atteint.

Dans ce travail, on s'intéresse à distribuer une des méthodes la plus utilisée pour la vérification de la bornitude qui est la construction de l'arbre de

recouvrement. Sa construction va être distribuée sur l'ensemble des processus responsables de la génération de l'espace d'états.

Algorithme distribué de la génération de l'arbre de recouvrement :

L'algorithme de construction de l'arbre de recouvrement exécuté par tout processus pi est le suivant :

Algorithm 8 La génération distribuée de l'arbre de recouvrement.

```
GenerationArbredeCouverture()
Begin
Terminate ←false;
Borne ←true;
P←{s_init};
While ( (¬ Empty(P)) and (¬ Terminate) do
  if (Top(P) = Backtrack) then
            pj←sender _of(Backtrack) ;
          send (pj, Backtrack) ;
  else s←Depiler(P) ;
    if (s.compare=True and s.type=New) then
       if (Empty(succ(s)) then s.type←deadend
        else for all s'∈succ(s) do
            if (s=s') then s.type←old
              else  if (s'>s) then
                Borne←false ;Teminate ← true ;send (All, Terminate) ;
                    else     s'.type←New;
                             s'.Compare=false ;
                             pre←pred(s) ;
                             while(pre≠Null) and(hash(pre)←i) and (s'<pre)do
                              pre←pred(pre) enddo
                             if (s'>pre) then Borne←false ; Teminate ← true ; send
(All, Terminate) ;
                                else if (s'=pre) then pre.type←old ;
                                    else if (pre=Null) then s.Compare← true ;
                                                        if (hash(s') = i) then
                                                          empiler (s' );
                                                        else send(s',hash(s'),State)
                                                        endif
                                         else send(s',pred, hash(pred),Compare)
                                         endif
                                    endif
                               endif
                          endif
                    endif
                enddo
end
```

La procédure MessageHandler qui traite les différents messages reçus, comportera dans ce cas les types des messages suivants :

Algorithm 9 Mise à jour de la procédure MessageHandler

```
MessageHandler(message)
Begin
.
If (Message.type= Compare)
   while(pre≠ Null) and( hash(pre)=i) and (s'<pre) do
        pre←pred(pre) enddo
        if (s'>pre) then Borne←false ; Teminate ← true ;send (All, Terminate) ;
        else if (s=pre) then pre.type←old ;
             else if (pre=Null) then
                    s'.Compare← true ;
                    if (hash(s') = i) then empiler (s') ;
                    else send(s',hash(s'),State)
                   endif
                 else
                 send(s',pred, hash(pred),Compare)
               endif
           endif
if(message.type= State)
then empiler (s')
     send ( sender, Backtrack)
endif
if(message.type= Backtrack)
then empiler (Backtrack)
endif
.
end
```

A chaque fois qu'un processus explore un état s, il ne l'envoie à son propriétaire que lorsqu'il l'aura comparé à tous ses antécédents.

La comparaison se fait par des envois de message et elle se termine par lâun des cas suivants :

- Arriver à un marquage qui recouvre s (supérieur à s) ce qui permet de détecter la non bornitude du modèle, et par conséquent la fin de l'algorithme de la bornitude est détectée.

(Notons qu'un marquage s est dit supèrieur à un marquage s' ssi toutes ses composantes sont supérieures à celles de ce dernier.)

- Arriver à un marquage identique au marquage comparé, par conséquent on empile l'état exploré chez le processus propriétaire et on le marque old,

- Arriver au marquage initial, ce qui nécessite l'envoi de l'état à son propriétaire (si celui-ci est différent au processus explorant le marquage) et terminer l'éxploration.

Dans le cas où le marquage s n'as pas de successeur, son type est immédiatement marqué deadend, ce qui signifie que c'est un état de blocage.

Détection de la terminaison :

Afin de détecter la terminaison de l'algorithme, un test au niveau des processus est effectué.

A chaque fois qu'un processus i n'a plus d'états à explorer, il envoie un message (jeton) de type **End** au processus i+1, ce dernier si lui même n'as plus d'états à explorer il envoie de la même façon le message **End** à son voisin, sinon il le détruit et ainsi de suite.

Dans le cas où le message de détection est retourné au processus initiateur i, la terminaison de l'algorithme est alors détectée.

III. Evaluation des performances :

Dans le cas des réseaux de Petri stochastiques, on s'intéresse à l'analyse du comportement temporel du système modélisé.

Après vérification de l'ergodicité du modèle, on procède à la construction de la matrice génératrice Q où $Q_{s,s'}$ est le taux de transition à partir de l'état s vers l'état s', pour $s \neq s'$ *(s, s' ∈ S)*. Les entrées de la diagonale de Q sont données par la négation de la somme des éléments de la ligne correspondante.

A chaque fois qu'un processus i reçoit un état s ou à chaque fois qu'il termine la comparaison d'un état dont il est propriétaire, il attribue une identification à cet état s'il n'a jamais été exploré ou bien l'ancienne identification sinon, et il envoie par la suite un triplet *(s, s',τ)* au processus coordonnateur où s' est le successseur de s et τest le taux de franchissement de la transition t tel que : s[t >s'.

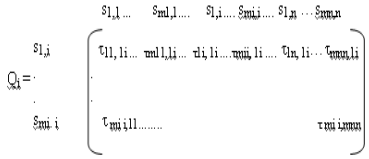
Lorsque la terminaison est détectée, le processus coordinateur aura toutes les entrées de la matrice Q, sur laquelle il applique la méthode Gauss-Seidal pour résoudre le système suivant :

$$\begin{cases} \pi Q = 0 \\ \sum_{i=1,n} \pi_i = 1 \end{cases}$$

Résolution distribuée:

La méthode de résolution présentée ci-dessus, ne permet pas d'exploiter l'aspect de distribution de calcul tel que utilisé dans la vérification de l'ergodicité, par conséquent nous proposons une nouvelle technique où la matrice génératrice reste distribuée sur l'ensemble des processus.

Dans ce cas chaque processus pi garde une sous partie de la matrice Q, notée Q_i tel que $Q_{i,s,s'}$ correspond au taux de transition de l'état s vers l'état s', pour $s \neq s'$ tel que $s \in S_i$, $s' \in S$, où S_i est l'ensemble des états appartenant au processus pi.



Où $\tau_{i,j}$ est le taux de transition de l'état s_i vers l'état s_j.

Lors de l'application de la méthode Gauss-Seidal pour résoudre le système des probabilités, le processus pi diffuse un message de calcul du pivot à tous les autres processus pour appliquer l'opération du pivotage sur le reste des lignes.

Algorithme de résolution distribuée :

Le procédé de pivotage distribué effectué par tout processus pi est le suivant :

Algorithm 10 Résolution numérique distribuée

```
ResolutionDist ()
Limit= height of G^i
index=0;
Begin
Receive (MessType) ;
If (MessType =Pivot) then
  for all line l of G^i: l ← G^i_l - G^i_{l,l}*(LinePivot) ;
  Send (sender,Ack) ;
Endif ;
If (MessType =Ack) then
  If (all send Ack) then index =index+1 ;
    If (index ≤ Limit )
    then LinePivot← G^i_{index} / G^i_{index,index}
       for all line l of G^i
        if(l> index)then G^i_l←G^i_l - G^i_{l,l}*(LinePivot);endif ;
        for all j >i Send (pj,Pivot, index, LinePivot)
    else if (index >Limit) then send(pi+1,Pass) ;endif ;
    endif
if(MessType=Pass) then
   If (index ≤ Limit )
    then LinePivot← G^i_{index} / G^i_{index,index}
       for all line l of G^i
       if(l> index)then G^i_l←G^i_l - G^i_{l,l}*(LinePivot);endif ;
       for all j >i Send (pj,Pivot, index, LinePivot)
      wait(Ack) ;
endif
end.
```

La fonction de pivotage est appliquée à toutes les lignes de la matrice G^iqui représente une sous partie du système d'équation à résoudre, écrit sous forme matricielle.

Aprés que le processus *pi* termine le pivotage de ses lignes, il passe la main au processus *pi+1*, et ainsi de suite jusqu'à l'obtention d'une matrice triangulaire supèrieure, répartie sur l'ensemble des processus.

Le processus *p1* reçoit son premier message de la part du processus coordonnateur, qui initie la fonction de pivotage sur la première ligne de la matrice qui représente la condition: $||\pi|| = \sum_{i=1,n} \pi_i = 1$.

IV. Conclusion :

Les algorithmes distribués proposés dans ce chapitre permettent l'analyse de la vivacité et la bornitude de l'espace d'états distribué.

Leur vérification garantit l'ergodicité du système modélisé, ce qui permet de passer à l'évaluation des performances de ce dernier.

Cette évaluation est basée sur la résolution du système stochastique, dépendant de la matrice génératrice qui est dans le cas distribué, répartie sur l'ensemble des processus. De ce fait un algorithme de résolution distribué est aussi proposée.

Afin d'analyser les performances de ces algorithmes, nous avons procédé à leur implémentation, ce qui fait l'objectif du chapitr prochain.

Chapitre 5:
Implémetation et expérimentations

I. Introduction:

Afin d'analyser les performances et la complexité des algorithmes proposés dans le chapitre précédant, nous avons procédé à l'implémentation ces derniers à l'aide de langage C++. Ce dernier offre un mécanisme simple et efficace, permettant la communication et les envois des messages entre les processus, qui est le principe des *sockets.*

Dans cette partie nous donnons les résultats obtenus pour deux exemples classiques qui sont: problème des gestionnaires de base de données distribuées et problème des philosophes. Et cela en utilisant un réseau de cinq machines (Pentium IV avec une mémoire de 512 Mbytes).

II. Génération de l'arbre de recouvrement:

II.1 Problème de gestionnaire de base de données (DDB):

Une base de donnée distribuée consiste en n stations différentes *[Bou 07]*, chacune d'elle contient une copie de toutes les données. Cette copie est manipulée par un gestionnaire local qui peut mettre à jour sa propre copie.

Un gestionnaire de base de données peut être dans un des trois états suivants : *Inactive, Waiting ou Performing.*

Lorsqu'un gestionnaire décide de mettre à jour sa copie, il envoie un message à tous les autres gestionnaires et il passe de l'état *Inactive* à l'état *Waiting.*

Tandis que les autres gestionnaires passent de l'état Inactive à l'état *Performing.* Lorsque ces derniers acquittent la réception de la mise à jour, ils reviennent à leur état initial : *Inactive.*

A la réception de tous les acquittements, le premier gestionnaire lui même passera à l'état *Inactive.*

L'exploration de l'arbre de recouvrement distribué des gestionnaires de base données distribuées, en utilisant un réseau de trois machines (Pentium IV avec une mémoire de 512 Mbytes), a donné les résultats ci-dessous:

Nb de gestionnaires	Nb de noeuds	Nb d'états	eq. spatial	Temps(s)	Temps(s) 1 proc
3	58	28	1.05	<0.01	<0.01
4	1 085	109	2	0.07	0.52
5	36 826	406	1.99	34.73	265.98
6	119 390	1 459	1	307.90	2005.23

Table 1: Exploration de l'arbre de recouvrement pour le problème des DDB

Une fonction de hashage a été bien choisie, afin d'équilibrer la distribution des états sur l'ensemble des processus. L'équilibre spatial maximal obtenu est égal à 2 , ce qui signifie qu'au maximum un processus peut avoir deux fois le nombre de noeuds d'un autre processu.

On remarque qu'à partir de niveau 5 on peut avoir un temps de réponse sept fois moins réduit que l'approche séquentielle;

II.2 Problème des philosophes :

Le problème des dîners des philosophes est un problème classique en informatique, il concerne l'ordonnancement des processus et l'allocation des ressources à ces derniers *[Wik 07]*.

Un philosophe n'a que trois éats possibles :

1. Penser pendant un temps indéterminé ;
2. Être affamé (pendant un temps déterminé et fini sinon il y a famine) ;
3. Manger pendant un temps déterminé et fini.

Les résultats obtenus, en analysant les réseaux de Petri modélisant le modèle des philosophes en fonction de leur nombre sont donnés comme suite:

Nb de philosophes	Nb de noeuds	Nb d'états	eq. Spatial	Temps(s)	Temps(s) 1 proc
5	96	11	1.31	0.36	0.36
6	8 335	18	1.84	0.82	0.85
7	13 373	29	1.99	32.01	143.21
8	199 556	47	3.09	111.82	528.90

Table 2: Exploration de l'arbre de recouvrement pour le problème des philosophes

On constate dans cet exemple qu'un processus peut avoir jusqu'à trois fois le nombre de marquages d'un autre processus.

La compléxité de l'algorithme de l'exploration en terme de temps d'exécution, de nombre de message échangés et d'espace consommé, est etroitement liée au choix de la fonction de hashage. De ce fait une étude sur le choix de cette dernière selon le modèle à étudier, s'avère nécessaire afin de pouvoir analyser des systèmes plus larges et plus complexes.

dans ce cas le taux de réduction est de cinq, à partir de niveau sept.

III. Génération de l'espace d'états:

L'algorithme de génération de l'espace d'états distribué pour les deux exemples a abouti aux résultats suivants:

1 . Problème de gestionnaire de base de données (DDB):

Nb de gestionnaires	Nb d'états	Nb arcs traversant	eq. spatial	eq. spatial	Temps (s)	Temps (s) 1 proc
3	28	4	1.11	1.05	<0.01	<0.01
5	406	6	2.01	2	0.03	0.08
8	17497	10	1.5	1.99	68.77	334.73
10	196837	12	1.33	1	8467.24	45269.75

Table 3: Génération de l'espace d'états pour le problème des DDB

La génération distribuée a permis une accélération de cinq par rapport à la génération séquentielle.

2. Problème des philosophes:

Nb de philosophes	Nb d'états	Nb arcs traversant	eq. Spatial	temps(s)	Temps (s) 1 proc
5	11	12	1.33	<0.01	<0.01
7	29	38	1.37	<0.01	0.01
10	123	168	1.5	<0.01	0.15
15	1364	1884	1.14	0.55	2.12
20	15127	20902	1.85	175.69	598.32

Table 4: Génération de l'espace d'états pour le problème des philosophes

On constate que cet algorithme est beaucoup moins complexe que celui de l'arbre de recouvrement, où un état n'est sauvegardé qu'une seule fois et en plus on n'a pas de message de comparaison. Ce qui a permis d'appliquer l'algorithme sur des modèles beaucoup plus larges.

IV. Résolution numérique:

L'évaluation des performances dans le cas stochastique est basée sur le calcul des probabilités stationnares en résolvant le sytème:

$$\begin{cases} \pi Q = 0 \\ \sum_{i=1,n} \pi_i = 1 \end{cases}$$

Aprés vérification de l'ergodicité du système, la résolution numérique est initiée par le processus coordinateur qui envoie un message de pivotage à tous les autres processus.

Le nombre de messages échangés entre les processus dans ce cas est donné comme suit:

Nb de gestionnaires	Nb de messages	Nb de messages 1 proc
3	58	207
4	164	361
5	976	1789
6	2920	4878

Nb de philosophes	Nb de messages	Nbde messages 1 proc
5	24	30
6	34	82
7	54	309
8	86	523

Table 5: Nombre de messages pour la résolution numérique.

On remarque que le nombre de message est réduit d'un taux de deux pour l'exemple des gestionnaires et de six pour le cas des philosophes.

V. Conclusion :

Les résultats donnés ci-dessus, sont obtenus en implémentant les algorithmes présentés précédemment pour les deux exemples : Gestionnaire de bases de données distribuées et Problème des philosophes.

Pour chaque exemple une fonction de hashage a été choisie pour mieux équilibrer la répartition des états sur l'ensembles des processus et pour réduire le nombre des arcs traversants ce qui revient à réduire le nombre de messages échangés.

Conclusion générale:

La modélisation formelle est devenue actuellement une contrainte incontournable pour couvrir les exigences des systèmes complexes dans de nombreux domaines.

De ce fait, plusieurs outils de mdèlisation sont apparus tels que les chaînes de Markov, les files d'attente, les réseaux de Petri (RdP)...etc

Ces derniers se sont imposés d'eux même comme un choix inévitable pour répondre aux besoins des systèmes complexes vue les deux concepts qu'ils présentent : un concept graphique permettant de mieux voir et exprimer le système, et un seconde mathématique pour l'évaluation qualitative et quantitative des performances du système.

Néanmoins, les besoins de la modélisation ont mené à introduire de nouvelles extentions des RdP : les réseaux de Petri colorés (RdPC), les réseaux de Petri temporels (RdPT) et les réseaux de Petri stochastiques (RdPS).

L'analyse d'un système à partir de son modèle, nécessite la génération de son espace d'états qui consiste à l'énumération de tous les états possibles à partir d'un certain état initial.

Cette génération est souvent confrontée au problème de l'explosion combinatoire.

Pour pallier ce problème, plusieurs techniques de réduction ont été définies afin de réduire la taille de cet espace tels que la symétrie, la modularité ...etc

Ces techniques ne peuvent être appliquées que si le modèle analysé ait certaines caractéristiques, ce qui n'est pas toujours possible.

Par conséquent une nouvelle approche a été introduite qui consiste à répartir l'ensemble des états générés sur un ensemble de processus interconnectés et communiquant en changeant des messages. Ce qui permet d'obtenir un espace d'états distribué.

Dans le cadre de notre travail, nous avons commencé par la présentation des différentes notions et caractéristiques qui se situent dans l'éventail des réseaux de Petri en mettant l'accent sur le problème de l'explosion combinatoire ainsi que l'intérêt de la distribution.

Néanmoins, Une distribution aléatoire peut poser un d'éséquilibre temporel et spatial, par conséquent la distribution se basera sur une fonction dite de hachage permettant d'attribuer à chaque état généré un processus propriétaire.

Nous avons également présenté un ensemble d'algorithmes pour l'étude et l'analyse de l'espace d'état distribué.

Cette analyse consiste à vérifier certaines propriétés qualitatives, qui sont la vivacité et la bornitude à évaluer les performances du système par

une étude quantitative.

Ces algorithmes ont étét implémentés et quelques expérimentations sur deux exemples standards, Problème des philosophes et gestionnaires de bases de données distribuées, ont été exposés vers la fin de ce document.

Perspectives :

Beaucoup de travaux restent comme perspectives dans ce domaine tels que:

- Améliorer l'algorithme de la bornitude: en utilisant un deuxième niveau de hashage, c-à-d au lieu de chercher un état s'il existe dans tout l'ensemble des états déjà explorés, on peut limiter la rechèrche dans un sous ensemble.
- L'utilisation des algorithmes distribués basés sur d'autres structures de données telles que les diagrammes de décision pour une représentation plus compacte de l'espace d'états.
- Etendre l'idée de répartition sur d'autres extensions de RdP, tel que les réseaux de Petri Colorés.
- La définition des algorithmes de répartition (fonction de hashage) à partir d'une analyse structurelle du modèle.

Annexe: Les chaînes de Markov

Les chaînes de Markov :

La théorie des processus stochastiques en particulier les processus de Markov permet de calculer les probabilités d'états stationnaires pour décrire l'évolution temporelle d'un système dynamique. Les processus Markoviens forment une classe particulière de processus aléatoires appelés les processus sans méoire pour lesquels tout l'historique du processus est résumé par l'état courant, tel que la connaissance de cet état suffit prévoir l'évolution future du système.

Notation :

Considérons un système qui comprend M états possibles : E_1à E_M, où M peut être fini ou non mais les états étant toujours dénombrables. La probabilité que le système soit dans l'état E_i à l'étape m est notée:Pi(m).

Dans le cas où la probabilité du passage d'un état E_ià un autre état E_j ne dépend que des deux états $E_i$et E_j, est notée P_{ij}.

Le passage d'un état à un autre peut être représenté par un graphe, dont les noeuds représentent les états et les arcs modélisent les différentes transistions entre ces états.

1. Définition de la chaîne de Markov:

La séquence de variables aléatoires X1, X2,.. Xk forme une chaîne de Markov à temps discret si pour tout entier n et pour toutes les valeurs possibles des variables aléatoires, on a : P[Xn=j/ X1=i1, X2=i2,...Xn-1=in-1] = P[Xn=j/ Xn-1=in-1]

2. Propriétés de la chaîne de Markov:

1)Homogénéité:

Soit $Pij(n)$la probabilité de transition qui est égale par définition à P[Xn=j/Xn-1=i], si cette quantité ne dépend pas de n, alors la chaîne de Markov est dite homogène, on peut écrire alors P_{ij}= P[Xn=j/Xn-1=i]. On associe au graphe précédent une matrice P appelée matrice de transition dont les éléments correspondent aux différentes probabilités P, appelées les probabilités de transition.

2)Irréductibilité :

Une chaîne de Markov est dite irréductible si tout état de la chaîne peut être atteint à partir de n'importe quel autre état, cela se traduit par : $\forall i, j \exists$

n tel que$Pij(n)>0$ où $Pij(n)$ est l'élément de la matrice $P^{(n)}$ = (P*P...*P) figurant à l'intersection de la ième ligne et de la jème colonne.

L'irr'ductibilité est traduite par l'existence d'une seule composante fortement connexe dans le graphe des états.

3)Récurrence:

Un état Ei est récurrent si la probabilité d'aller à cet état est égale à 1, on dit qu'un état est récurrent non nul si le temps moyen de retour à cet état est fini et récurrent nul sinon.

Si la probabilité de retour est infèrieure à un alors l'état est dit transitoire.

4)Apèriodicité :

Une chaîne de Markov est apériodique si le PGCD de la longueur de tous les cycles de probabilité non nulle est égal à 1.

5)Ergodicité:

Un état est ergodique s'il est apèriodique et récurrent non nul.

3. La stationnarité:

Soit Ω l'espace d'états de la chaîne de Markov, une distribution {zi , i$\in \Omega$}, définie sur les états de la chaîne est dite stationnaire ; si $\forall i \in \Omega$ zi est la probabilité d'être à l'état i. Une fois la distribution stationnaire est atteinte elle demeure la distribution du processus pour toutes les étapes suivantes.

Dans le cas d'une chaîne irréductible et apèriodique, les probabilités à l'état stationnaire sont calculées en résolvant le système suivant:

$$\begin{cases} \pi Q = 0 \\ \| \pi \| = \sum_{i=1}^{n} \pi_i = 1 \text{ ou } \| \pi \| \text{ est la norme de } \pi. \end{cases}$$

où πisont les probabilités à l'état stationnaire i.

Notons que l'ergodicité garantit l'existence d'une distribution stationnaire sur tous les états d'une chaîne de Markov.

Les chaînes de Markov permettent de réaliser une évaluation quantitative, mais pas qualitative, et par conséquent ce modèle n'est pas adapté à la modélisation de certaines caractéristiques des systèmes parallèles telles que: la synchronisation, l'interblocage, l'allocation des ressources etc. Par conséquent d'autres outils ont été développés tel que les réseaux de Petri.

Table des matières :

Bibliographie:

References

[Bar 01] :J.Barnat, L. Brim, J. Stribrnà .«Distributed LTL model checking in SPN». In Proc. 8th International SPIN workshop on model checking software, Spinger-Verlag, 2001.

[Ber 05] :B. Berthomieu, D. Lime, O. Roux, F. Vernadaf. « Modélisation des systèmes Réactifs », Journal européen des Systèmes Automatisés, 2005.

[Bra 83] :G.W.Brams. :«Réseaux de Petri : théorie et pratique». Edition Masson, tomes 1et 2, 1983.

[Bou 07] :M.C. Boukala, L. Petrucci. «Towards Distributed Verification of Petri Nests Properties», VECoS07, 1st International Workshop on Verification and Evaluation of Computer and Communication Systms, May, 2007.

[Cia 02] :G. Ciardo. «Distributed and structured Analysis Approaches to Study Large and Complex Systems ». Departement of Computer Science College of William and Mary, Williamsburg, USA, 2002.

[Cou 98] :G. Coulouris, J. Dollimore, T. Kindberg. «Distributed Systems â Concepts and Design Â». Addison-Wesley, 2nd edition, 1998.:N.Djoudi, A.Smail. «Application des RDPS généralisés non bornés pour l'évaluation des parformances des systèmes dâattente avec rappels ». Thèse de fin d'études, USTHB, 2003.

[Flo 85] :Florin,: G. S.Natkin.«Les réseaux de Petri stochastiques, théorie et technique de calcul».Thèse de doctorat d'état, P&M Curie, Paris, 1985.

[Hav 99] :B. Haverkot, Henrik Bohnenkamp, Alexander Bell . « On the Efficient Sequential and Distributed Evaluation of Very Large Stochastic Petri Nets». Department of Computer Science, Laboratory for Distributed Systems, Germany, 1999.

[Jor 99] :J.B. Jorgenesen, L.M. Krinsten. «Computer aided verification of Lamportâs fast mutual extension algorithme using Colored Petri Nets and occurrence graphs with symmetries». IEEE transaction on parallel and distributed systems; 1999.

[Jou 06] :C. Joubert, R. Mateescu. « Distributed On-The-Fly model Checking and test Case generationÂ», Rapport de recherché N"° 5880, Institut national de recherche en informatique et en automatique, Avril 2006.

[Kri 99] :L.M. Kristensen, A Valmari. «Finding Stubborn Sets of Coloured Petri Net Without Unfolding»,1999.

[Kri 00] :L.M. Kristensen. «State Space Methods for Coloured Petri Nets», Department of Computer Science, University of Aarhus, Denmark, 2000.

[Ler 99] :F. Lerda, R. Sisto. «Distributed-memory model checking with SPIN» 5th and 6th International SPIN workshop on model checking software, Spinger-Verlag, 1999.

[Mer 74] :P. Merlin. «Study of the recoverability of computing System», these de doctorat, California, 1974.

[Paq 05] :S. Paquette. «Evaluation symbolique des systèmes probabilistes à espace d'états continu », Département d'Informatique et de Génie Logiciel, faculté des Sciences et de Génie, Université Laval, Québec, 2005.

[Pet 62] :C. A. Petri «Kommunikation mit automaten», Ph. D. Thesis Institut fur instrumentelle mathematik, Bone, 1962.

[Pet 05] :L. Petrucci , C. Lakos. « Distributed and modular state space exploration for timed Petri nets», LIPN, CNRS UMR 7030, Université Paris VIII, 2005.

[Tar 72] :R. E. Tarjan.« Depth- first search and linear graph algorithms », SIAM Journal on computing, 1972.

[Val 91] :A. Valmari. «Stubborn Sets of Coloured Petri Nets ». In G. Rozenberg, editor, Proceedings of ICATPNâ91, pages 102-121, 1991.

[Wik 07] :« http://fr.wikipedia.org/wiki/Déner_des_philosophes »

[Zen 87] :A.Zenie. «Les RDPS colors: application à l'analyse des systèmes répartis en temps réel »Thèse de doctorat d'état, P&M Curie, Paris, 1987.