Réseaux d'Automates Stochastiques : Analyse transitoire en temps continu et algèbre tensorielle pour une sémantique en temps discret

Leonardo Brenner

INSTITUT POLYTECHNIQUE DE GRENOBLE

Número attribué par la bibliothèque
|_/_/_/_/_/_/_/_/_/_/

THÈSE

pour obtenir le grade de

DOCTEUR DE L'Institut Polytechnique de Grenoble

Spécialité : ***"Informatique"***

préparée au LABORATOIRE D'INFORMATIQUE DE GRENOBLE
dans le cadre de l'École Doctorale ***"Mathématiques, Sciences et Technologies de l'Information, Informatique"***

présentée et soutenue publiquement par

Leonardo BRENNER

le 16 Septembre 2009

Réseaux d'Automates Stochastiques :
Analyse transitoire en temps continu et Algèbre tensorielle pour une sémantique en temps discret

---

Directeur de thèse :
Mme Brigitte Plateau

JURY

MR ANDRZEJ DUDA	Grenoble INP	Président
MME LEILA KLOUL	Université de Versailles St Quentin	Rapporteur
MR PATRICE MOREAUX	Université de Savoie	Rapporteur
MME BRIGITTE PLATEAU	Grenoble INP	Directeur de thèse
MR JEAN-MARC VINCENT	Université Joseph Fourier	Invité

À Marina, mon amour, ma vie.

Remerciements

Une thèse n'est pas une tache facile à réaliser. L'élaboration de cette thèse a été possible grâce au support, à l'amitié et au savoir de plusieurs personnes.

Tout d'abord, j'aimerais remercier ma directrice de thèse Mme. Brigitte Plateau. Je n'aurai pas fini cette thèse sans ses enseignements et son support. Sa rigueur scientifique et sa volonté inépuisable m'ont motivé et m'ont toujours mis sur le bon chemin. Je vais me rappeler (et essayer de suivre) ses enseignements et ses conseils pour toute ma vie. Merci beaucoup.

Aux membres du jury, Leïla Kloul, Patrice Moreaux et Andrzej Duda, je les remercie d'avoir accepté d'évaluer ces travaux. Leur remarques et suggestions m'ont été très utiles et m'ont permis d'améliorer la qualité de ce document. Je remercie spécialement Jean-Marc Vincent, non seulement pour ses conseils scientifiques, mais aussi pour toutes les discussions agréables que nous avons pu avoir à la cafétéria sur beaucoup de sujets intéressants.

J'aimerais remercier sincèrement Paulo Fernandes. Il y a déjà plus de dix ans que j'ai commencé à travailler avec toi et que j'ai découvert le goût pour la recherche. Aujourd'hui, au delà d'un collège, c'est un grand ami.

J'aimerais également remercier Jean-Michel Fourneau et Billy Stewart pour les enseignements qu'ils m'ont apporté tout au long de cette thèse.

J'aimerais aussi remercier tous mes amis qui ont rendu mon séjour à Grenoble formidable. Je tiens à citer mes amis Maxime Martinasso, Jérôme Vienne, Mauricio Pillon, Edicarsia Barbiero, Thaïs Webber, Edson Moreno, Ricardo Czekster, Osmar Marchi, Simone Ceolin, Alessandro Noriaki, Thiago Carmago, Michelle Leonhardt, Pedro Velho, Patricia Scheeren, Johnatan Pecerro, Karoline Aragão, Windson Carvalho, Lucas Schnorr, Fabiane Basso, Bringel Filho, Suely Ramos, Everton et Ana Paula Hermann, Rodrigo Ribeiro, Christiane Pousa, Daniel Cordei-ro, Kelly Braghetto, Marcio Castro, Luciano Ost, Beatriz Leite et César Gonçalves. Et tellement d'autres qui ont fait partie de ma vie dans cette période et que j'ai surement oublié de citer. Un grand merci à tous.

J'aimerais remercier les membres du laboratoire LIG qui m'ont accuilli et m'ont aidé pendant toutes ces année.

J'aimerais dire un mot spécial à Céline et Aline Lopes, mes chères colocataires, je les remercie beaucoup pour toutes les leçons de français et surtout pour m'avoir accueilli chez elles lorsque je venais d'arriver ici.

J'aimerais remercier Luiz Angelo et Manuele, Luciano et Andreia, et, Afonso et Carol, qui ont été les compagnons les plus proches à moi et à ma femme pendant cette période. Leur amitié compte beaucoup pour moi.

Je tiens à remercier plus spécialement mes collègues Afonso Sales (encore) et Ihab Sbeity. Travailler avec eux a été surtout un grand plaisir.

Je ne pourrais jamais oublier de remercier toute ma famille, ainsi que ma belle-famille au Brésil. Leur soutien et leurs mots chaleureux m'ont toujours réchauffé le coeur et m'ont donné la force et l'envie de continuer. J'aimerais surtout remercier ma mère Nelsi Brenner et mon père Neumar Brenner pour leur soutien inconditionnel. iIls sont un modèle de vie pour moi.

Et finalement, j'aimerais remercie la personne la plus importante de ma vie, Marina, ma femme. Son soutien, sa compréhension et son amour, m'ont toujours motivé et m'ont donné la force de finir cette thèse. Cette thèse est la sienne aussi.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Le développement des systèmes informatiques est devenu si complexe que leur conception ne peut être basée uniquement sur l'intuition et l'expérience du concepteur.

L'intuition et l'expérience du concepteur, comptent bien sûr dans la qualité de la conception du système, mais l'aide de *méthodes de modélisation* et *prédiction de performances* permet une planification et une gestion plus systématique et moins ambiguë du système.

Pour prédire la performance d'un système et anticiper les problèmes de performance avant la construction d'un premier prototype, il est donc fondamental de disposer de méthodes de modélisation et prédiction de performance qui permettent d'analyser et de comprendre le comportement du système. Les étapes de modélisation et prédiction de performances sont appelés par le terme générique d'*évaluation de performance*.

L'évaluation de performance doit être appliquée tout au long du cycle de vie du système, de la spécification à l'exploitation. Chaque phase du cycle de vie du système favorise l'utilisation de certaines méthodes d'évaluation de performance. Si d'un côté les formalismes de modélisation et les méthodes de simulation sont plus utilisés dans phase de spécification, de l'autre côté, l'observation est plus souvent employée après la construction d'un premier prototype.

L'évaluation de performance dans une seule des phases du cycle de vie du système peut engendrer des surcoûts importants, *i.e.*, des erreurs de conception découvertes durant la phase d'exploitation peuvent entraîner la réécriture complète du système.

Dans cette thèse nous nous sommes intéressés à l'évaluation de performances des premières phases du cycle de vie de la conception des systèmes informatiques parallèles et distribués. Par définition, de tels systèmes sont modulaires et naturellement grands. Ils sont trop complexes pour être traités par des méthodes traditionnelles d'évaluation de performances, comme par exemple une description directe sous la forme de Chaînes de Markov.

Nous nous intéressons donc à l'évaluation de performances de modèles avec très grands espaces d'états décrits par des formalismes de haut niveau.

Du fait de la structure modulaire de ces systèmes, ces modules ont souvent un comportement presque autonome, *i.e.*, interactions entre les modules sont plutôt rares. Pour décrire ce type de comportement, Plateau a proposé le formalisme de *Réseaux d'Automates Stochastiques (*SAN-*Stochastic Automata Networks)* [72, 73].

Le formalisme SAN nous permet de modéliser le comportement autonome (modélisés par des événements locaux) de chaque composant du système dans un automate et l'interaction entre les composants par des événements synchronisants.

L'objectif de cette thèse est d'améliorer le formalisme SAN *par l'inclusion de nouvelles méthodes numériques en temps continu et par la proposition d'une représentation tensorielle compacte en temps discret.*

Nous faisons, dans la section suivante, un bref rappel des méthodes d'évaluation de performances connues dans la littérature.

1.1 Méthodes d'évaluation de performances

L'évaluation de performances peut être décomposée en deux étapes distinctes : la modélisation du système et la résolution du modèle. L'étape de modélisation du système consiste à développer une description formelle du système, alors que la résolution du modèle est une étape numérique qui permet d'obtenir des indices de performances du système.

1.1.1 Formalisme de modélisation

Les travaux les plus anciens viennent de l'analyse directe des processus stochastiques [97] et plus généralement des Chaînes de Markov [92].

Les Chaînes de Markov sont un des formalismes de modélisation les plus anciens et aussi un des plus utilisés dans l'évaluation de performance. Que ce soit à temps continu ou à temps discret, la description d'un système consiste à définir successivement :

- les états du système (on se restreint à des systèmes avec un nombre d'états fini) ;
- les transitions entre les états ;
- la temporisation des transitions.

Ces définitions nous permettent d'analyser le comportement dynamique d'un système étant donné que celui s'exécute dans un environnement aléatoire. La simplicité et la facilité de modélisation des Chaînes de Markov et les théorèmes opérationnels de la théorie concernée, leur permettent d'être employées dans de nombreux domaines d'applications [32, 38, 45].

Cependant, lorsque le nombre d'états est important (de l'ordre de millions d'états), il est quasiment impossible d'envisager une modélisation directe sous forme état-transition. La taille

de l'espace d'états généré est parfois si grande que non seulement on ne peut pas obtenir des indices de performances du modèle, mais souvent la matrice de transition de la chaîne de Markov ne peut même pas être stockée.

Pour cela, plusieurs techniques et formalismes de modélisation de haut niveau ont été développés au cours des dernières décennies. Ces formalismes de haut niveau sont structurés, ce qui leur permet de prendre en compte l'aspect compositionnel et hiérarchique des systèmes.

Vers la fin des années 50, avec les travaux de Jackson [57, 58], il est apparu une nouvelle approche basée sur les réseaux de files d'attente [62]. Les recherches dans ce domaine ont connu un formidable essor jusqu'à la fin des années 70, notamment avec les travaux de Little [63], Baskett, Chandy, Muntz et Palacios [5], et Reiser et Lavenberg [78].

Cependant, le domaine d'application des réseaux de files d'attente classiques est assez restreint. La nécessité de traitement de problèmes plus complexes, comme par exemple des systèmes avec synchronisation, a motivé dès les années 70 d'autres approchées en évaluation de performance.

Une partie de ces approches ont abouti à des extensions du formalisme de réseaux de files d'attente, comme c'est le cas des travaux sur :

- les méthodes d'approximation de Chandy et Sauer [18] et de Courtois [70] ;
- les réseaux à capacité limitée de Dallery (*fork-join*) [23, 24] ;
- les réseaux avec clients négatifs (réseaux généralisés) [44] ;
- les réseaux avec contrôle de décision [99, 60] ;
- les réseaux hiérarchiques [15].

Toutes ces extensions ont fourni des outils puissants pour la modélisation de systèmes d'attente, cependant, leur domaine d'application continu à être relativement restreinte à des systèmes d'attente.

D'autres approches ont adopté les réseaux de Petri [49, 79] comme formalisme de base en ajoutant des extensions qui vont de simples temporisations constantes [81], jusqu'à des mécanismes beaucoup plus sophistiqués comme par exemple :

- les réseaux de Petri stochastiques [37] ;
- les réseaux de Petri stochastiques généralisés [3, 2] ;
- les réseaux de Petri colorés [50, 51, 59] ;
- les réseaux de Petri stochastiques généralisés superposés [30].

Dans les travaux sur les réseaux de Petri stochastiques, on va s'intéresser spécialement au travaux sur réseaux de Petri stochastiques à temps discret [65, 19, 102, 82].

D'autres approches ont abandonné la notion d'*entités* et de *flot* (respectivement *clients* et *routage* dans les réseaux de files d'attente ou *jetons* et *arcs* dans les réseaux de Petri), pour proposer une vision compositionnelle de sous-systèmes qui interagissent entre eux (concept de modèles modulaires).

Ceci est le cas de :

– des algèbres des processus stochastiques [67, 54] ;
– des réseaux d'automates stochastiques [74, 75, 34].

À partir de ces formalismes, il est possible de générer la chaîne de Markov qui représente le système.

Dans le cadre de cette thèse, nous nous intéressons plus particulièrement au formalisme de **Réseaux d'Automates Stochastiques** *(*SAN- *Stochastic Automata Networks).*

Nous ne rentrerons pas dans une discussion détaillée sur une comparaison des différents formalismes. Nous donnerons uniquement quelques éléments de comparaison en introduisant les travaux précédents dans chaque partie de cette thèse.

1.1.2 Méthodes de résolution

Quelque soit le formalisme utilisé, nous cherchons à calculer des indices de performance sur le système étudié. Plusieurs méthodes de résolution sont proposées dans la littérature. Parmi ces méthodes de résolution nous pouvons faire la distinction entre :

– les méthodes analytiques [5, 62, 78] ;
– les méthodes numériques [86, 92] ;
– les simulations [96, 100, 12].

Dans le cadre de cette thèse, les méthodes de simulations ne serons pas considérées.

Les méthodes analytiques sont les méthodes qui donnent une solution sans passer par la résolution numérique du système linéaire $\pi Q = 0$ où Q est la matrice de transition de la chaîne de Markov. Ces méthodes ont l'avantage d'éviter la résolution du système linéaire, système généralement très grand.

Par exemple, certaines chaînes de Markov, de par leur structure, ont des solutions avec une forme produit. Les processus de naissance et mort sont un des cas le plus simple de ce type de solution. Il est possible de trouver des solutions à forme produit pour d'autres chaînes plus complexes que les processus de naissance et mort. Les cas les plus étudiés sont probablement les réseaux de files d'attente [28, 62]. D'autres solutions à forme-produit sont connues en partant d'autres formalismes de modélisation. Notamment quelques réseaux de Petri stochastiques [37] et réseaux de Petri stochastiques généralisés [3] ont cette propriété. Plus récemment, pour quelques modèles des réseaux d'automates stochastiques, une solution à forme produit a pu être trouvée [77, 42, 41, 40].

Les méthodes numériques peuvent être partagées en deux groupes :

– les méthodes numériques directes ;
– les méthodes numériques itératives.

La littérature décrivant les méthodes numériques de résolution de systèmes linéaires est très abondante [71, 86, 92].

Les méthodes numériques de résolution de systèmes linéaires applicables aux chaînes de Markov sont généralement des méthodes itératives. Les méthodes directes, comme la méthode de Gauss [71], ne sont pas utilisables pour des modèles de grande taille (nombre d'états) comme ceux qui nous préoccupent ici.

Les méthodes itératives peuvent aller des méthodes simples comme la méthode de la puissance, Jacobi, Gauss-Siedel et sur-relaxation successive, jusqu'à des méthodes plus complexes comme les méthodes de projection (Arnoldi, GMRES, Lanczos, etc). Le lecteur peut trouver des descriptions très précises de ces méthodes dans les ouvrages de Stewart [92] et de Saad [86].

Aux méthodes itératives, on peut ajouter aussi les méthodes d'analyse transitoire, comme par exemple, la méthode d'uniformisation. Les méthodes itératives pour l'analyse transitoire utilisent le même principe de la méthode de la puissance pour obtenir la distribution de probabilité à l'instant de temps t. Autrement dit, des multiplications successives d'un vecteur par une matrice.

Dans le cadre de cette thèse, nous nous intéressons aux méthodes numériques itératives, plus particulièrement aux méthodes de calcul de distribution transitoire du système.

1.2 Objectifs de cette thèse

Les objectifs de cette thèse s'articulent en fonction de deux axes de recherche : les méthodes numériques et les formalismes de modélisation. Plus précisément, dans l'axe "méthodes numériques" nous abordons le problème de détection du régime stationnaire dans le calcul de la disponibilité ponctuelle pour des modèles avec grand espace d'états. Dans l'axe "formalismes de modélisation", nous proposons une représentation tensorielle compacte pour le formalisme SAN à temps discret.

Les principaux résultats de cette thèse sont :

- l'implantation et comparaison des méthodes de détection du régime stationnaire dans le calcul de la disponibilité ponctuelle des modèles à grand espace d'états stocké sous un format tensoriel généralisé ;
- la définition d'une algèbre tensorielle complexe pour les SAN à temps discret ;
- la proposition d'un descripteur pour le formalisme SAN à temps discret.

1.2.1 Plan de la thèse

Cette thèse s'articule autour de trois parties. La première partie, intitulé **Méthodes numériques pour l'analyse transitoire**, introduit les travaux réalisés sur le calcul de la disponibilité ponctuelle et des méthodes de détection du régime stationnaire en SAN à temps continu. La deuxième partie, intitulé SAN **à temps discret**, est dédiée à la présentation et définition du formalisme SAN à temps discret et, enfin, la troisième partie, intitulé **Outil de calcul : le logiciel** PEPS, présente les contributions apportées au logiciel PEPS.

Méthodes numériques pour l'analyse transitoire

Le premier chapitre de cette partie de la thèse (Chapitre 2 traite du formalisme de réseaux d'automates stochastiques à temps continu. On présente le formalisme de façon informelle afin de permettre la compréhension des modèles développes dans cette première partie de la thèse.

Le chapitre 3 présente les méthodes de calcul de la disponibilité ponctuelle, de détection du régime stationnaire ainsi que l'adaptation de l'algorithme *shuffle* pour la multiplication à droite.

Le chapitre 4 donne des exemples de modélisation de systèmes sujets aux défaillances utilisant le formalisme SAN. Les exemples présentés dans ce chapitre ont pour but de fournir des cas pratiques suffisamment représentatifs pour l'obtention des mesures numériques des méthodes de détection du régime stationnaire.

Le chapitre 5 dresse une comparaison de l'efficacité de méthodes de détection du régime stationnaire. Cette comparaison est faite par rapport au nombre d'itérations et à la précision des résultats sur les mesures obtenues sur les exemples de systèmes sujets aux défaillances.

SAN **à temps discret**

Le premier chapitre de cette partie (Chapitre 6) présente le formalisme SAN à temps discret. La définition formelle donnée dans ce chapitre nous permet de définir un modèle SAN et d'obtenir l'automate global représenté par le modèle.

Le chapitre 7 définit une nouvelle algèbre tensorielle (appelé *Algèbre Tensorielle Complexe*). Les propriétés de bases des opérateurs de cette algèbre sont énoncées. Dans ce chapitre on propose une algèbre tensorielle qui sera utilisée dans la définition d'un descripteur compact pour un modèle SAN à temps discret.

Enfin, le dernier chapitre de cette partie (Chapitre 8) propose un format tensoriel compact (*Descripteur Discret*) égal à la matrice de transition de l'automate global d'un modèle SAN et donne des preuves de cette égalité.

Outil de calcul : le logiciel PEPS

Le chapitre 9 présente un bref historique du logiciel PEPS et présente aussi la nouvelle structure du logiciel. Les motivations et la méthode de découpage sont également présentées.

Le chapitre 10 présente le nouveau langage de description du logiciel PEPS dans sa version 2007. Nous abordons dans ce chapitre les problèmes des anciens langages face à la description de systèmes avec des composants répliqués. Pour faire face à ces problèmes, nous proposons l'inclusion des nouvelles fonctionnalités et nous présentons, pour plusieurs cas-type, une façon compacte de les décrire.

Conclusion

Dans la conclusion nous dressons un bilan de travaux développés dans cette thèse, ainsi que les travaux en cours. Pour terminer, nous parlons de travaux futurs.

Annexes

Deux annexes sont incluses dans ce document. La première annexe est composée des tableaux et graphes des résultats numériques (nombre d'itérations et précision de résultats) de l'ensemble d'exemples utilisés pour tester les méthodes de détection du régime stationnaire. Bien que les principales conclusions soient extraites dans le chapitre 5, nous mettons en annexe ces chiffres pour permettre au lecteur de vérifier les conclusions avancées. La seconde annexe présente une description de la syntaxe du nouveau langage de description du logiciel PEPS de façon à ce que au lecteur puisse mieux comprendre les exemples présentés dans la thèse.

Première partie

Méthodes numériques pour l’analyse transitoire

Chapitre 2

Formalisme des Réseaux d'Automates Stochastiques (SAN) à temps continu

À cause de la nécessité de manipuler de grands volumes de données nécessaires pour modéliser des systèmes complexes, les recherches actuelles visent à trouver des solutions de plus en plus rapides et performantes. Dans ce contexte, le formalisme des Réseaux d'Automates Stochastiques (SAN - *Stochastic Automata Networks*) [72, 74, 34, 9, 89] propose des techniques qui permettent de traiter la modélisation de systèmes complexes.

Le formalisme SAN est basé sur le formalisme des Chaînes de Markov [92]. Néanmoins, la modélisation de systèmes complexes en utilisant le formalisme des Chaînes de Markov peut parfois rendre impraticable la réalisation du modèle à cause de la complexité de sa description, *e.g.*, un système avec des centaines de millions d'états.

Ainsi, le formalisme des Réseaux d'Automates Stochastiques, centre d'intérêt de ce chapitre[1], fournit une façon compacte et performante pour la description de systèmes complexes à grand espace d'états, et des méthodes pour optimiser des solutions stationnaires et transitoires. Ainsi, le formalisme SAN associe des techniques qui visent à éliminer (ou à réduire au maximum) la difficulté de modélisation en améliorant la vitesse d'obtention d'indices de performance, et en facilitant la modélisation des systèmes à grand espace d'états.

Un réseau d'automates stochastiques représente une chaîne de Markov. En conséquence, les propriétés des Chaînes de Markov sont aussi applicables à l'objet décrit par des Réseaux d'Automates Stochastiques. Le formalisme SAN permet la modélisation d'un système complexe par des petits sous-systèmes presque indépendants de façon à ce que soit possible le parallélisme (quand les automates n'interagissent pas) et le synchronisme (quand les automates interagissent).

[1] La présentation du formalisme SAN à temps continu donnée dans ce chapitre est commune dans cette thèse et dans [87].

2.1 Description informelle des SAN à temps continu

Le formalisme des Réseaux d'Automates Stochastiques a été proposé par Plateau [72]. L'idée principale du formalisme SAN est de modéliser un système en plusieurs sous-systèmes, *i.e.*, un système composé de modules. Cette modularité définie par le formalisme permet le stockage et la solution de *systèmes complexes* en faisant face au problème de l'*explosion combinatoire de l'espace d'états*. Le formalisme SAN est utilisé pour la modélisation de systèmes à *grand espace d'états*.

Chaque sous-système est représenté par un *automate stochastique*. Les transitions entre les états de chaque automate représentent les transitions d'un processus stochastique à échelle de temps continue ou discrète, grâce à des distributions exponentielles ou géométriques respectivement.

Il est intéressant de remarquer que tout modèle SAN peut être représenté par un seul automate stochastique qui a tous les états possibles du système modélisé. Cet automate correspond à la chaîne de Markov équivalente au modèle SAN (Section 2.1.6, page 19).

Les modèles SAN présentés dans ce chapitre peuvent être interprétés dans une échelle de temps discrète ou continue. Cependant, l'explication et les exemples présentés dans ce chapitre font référence à l'échelle de temps continue (*taux d'occurrence*), à la différence des modèles à échelle de temps discrète (*probabilité d'occurrence*). Un modèle SAN à échelle de temps continue gère une chaîne de Markov à échelle de temps continue (CTMC - *Continuous Time Markov Chain*), tandis qu'un modèle SAN à échelle de temps discrète gère une chaîne de Markov à échelle de temps discrète (DTMC - *Discrete Time Markov Chain*). Une description formelle du formalisme SAN à échelle de temps discrète, qui est le formalisme d'intérêt de la deuxième partie de cette thèse, sera présenté dans le chapitre 6. Dans cette thèse, on va adopter les notations suivantes pour la définition de modèles SAN. Pour le temps continu, nous allons présenter ce qui se trouve dans [34, 9, 89].

✍ **Soit**

$\mathcal{A}^{(i)}$	i-ème automate d'un modèle SAN, où $\mathcal{A}^{(1)}$ est le premier automate ;
$x^{(i)}$	x-ème état de l'automate $\mathcal{A}^{(i)}$, où $0^{(i)}$ est le premier état de l'automate $\mathcal{A}^{(i)}$;
e	l'identificateur d'un événement (local ou synchronisant)[2] ;
$e(\pi)$	l'identificateur d'un événement e avec une probabilité de routage π ;

Les probabilités de routage (π) sont utilisées quand un événement a plusieurs alternatives de transition. La probabilité peut être omise dans le cas où elle est égale à 1. De plus, la somme des probabilités pour les transitions du même événement partant du même état local doit être égale à 1.

[2]Dans le contexte de cette thèse, on utilise le caractère e comme identificateur d'un événement. Cependant n'importe quel caractère peut être utilisé pour définir l'identificateur d'un événement.

Dans ce qui suit, on va introduire les notions d'automates et d'événements, qui sont les notions de base du formalisme SAN. Ensuite, on présente les concepts d'éléments fonctionnels, ainsi que le concept de fonction d'atteignabilité et de fonction indice de performance. Enfin, on montre comment faire la génération d'une chaîne de Markov à partir d'un modèle SAN.

2.1.1 Automates stochastiques

Un automate stochastique est un modèle mathématique d'un système qui possède des entrées et sorties discrètes. Le système peut se trouver dans un quelconque état ou configuration interne. L'état dans lequel le système se trouve résume les informations sur les entrées précédentes et indique ce qui est nécessaire pour déterminer le comportement du système pour les entrées suivantes [56].

Selon cette définition, on peut décrire un automate stochastique comme un *ensemble d'états* et un *ensemble de transitions* entre eux [75]. La désignation stochastique attribuée aux automates signifie que le traitement du temps se fait par des variables aléatoires, qui suivent une distribution exponentielle sur une échelle de temps continue ou une distribution géométrique sur une échelle de temps discrète.

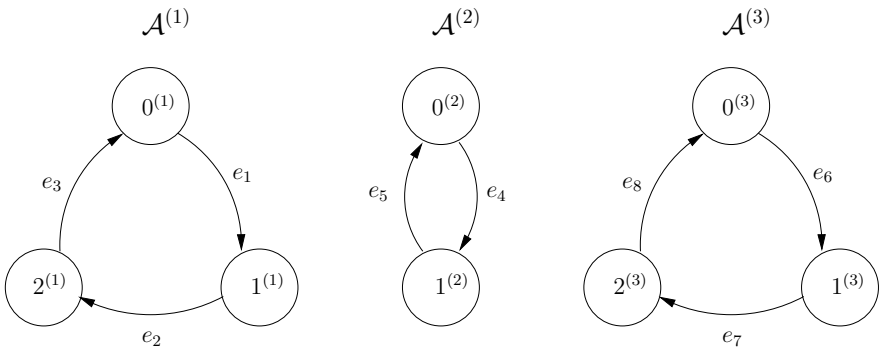
Graphiquement, un réseau d'automates stochastiques peut être représenté par un ensemble de graphes orientés, où chaque graphe est associé à un automate. Les *noeuds* d'un graphe représentent les *états* d'un automate et les *arcs* entre les états représentent la *transition* d'un état à l'autre.

L'*état local* du système modélisé par un SAN est l'état individuel de chaque automate du modèle. L'*état global* d'un modèle est composé par l'ensemble des états locaux de chaque automate du système modélisé. Le changement de l'état global du système est le résultat du changement de l'état local d'un (au moins) automate du modèle.

Le changement d'un certain état local à l'autre est fait par l'occurrence des événements qui provoquent des transitions. Les transitions sont des primitives qui indiquent la possibilité de changement d'un état à l'autre. Chaque transition est associée à un ou plusieurs *événements*. Dans FIG. 2.1, on présente un modèle SAN avec trois automates complètement indépendants.

Dans ce premier exemple, les automates $\mathcal{A}^{(1)}$ et $\mathcal{A}^{(3)}$ du modèle possèdent trois états[3] chacun : les états $0^{(1)}$, $1^{(1)}$ et $2^{(1)}$ correspondent aux états du premier automate, les états $0^{(3)}$, $1^{(3)}$ et $2^{(3)}$ correspondent aux états du troisième automate et l'automate $\mathcal{A}^{(2)}$ possède seulement deux états $0^{(2)}$ et $1^{(2)}$. Huit événements sont modélisés dans cet exemple : trois événements (e_1, e_2 et e_3) se produisent dans l'automate $\mathcal{A}^{(1)}$; deux événements (e_4 et e_5) se produisent dans l'automate $\mathcal{A}^{(2)}$; et les autres trois événements (e_6, e_7 et e_8) se produisent dans l'automate $\mathcal{A}^{(3)}$. Vu qu'il n'y a pas d'interaction entre les automates, la solution de ce type de modèle peut être

[3]Indépendamment du choix des étiquettes des états locaux d'une automate, chaque état possède une représentation numérique récurrente de l'ordre de la modélisation, *i.e.*, un état possède une représentation numérique entre les valeurs $0, 1, 2, \ldots, n^{(i)} - 1$, où $n^{(i)}$ est le nombre d'états de l'automate $\mathcal{A}^{(i)}$.



Type	Even.	Taux	Type	Even.	Taux	Type	Even.	Taux
loc	e_1	α	loc	e_4	δ	loc	e_6	μ
loc	e_2	β	loc	e_5	λ	loc	e_7	σ
loc	e_3	γ						

FIG. 2.1 – Modèle SAN avec 3 automates indépendants

obtenue de manière totalement indépendante pour chacun des automates, *i.e.*, chaque automate peut être évalué individuellement et pour le système global on a une forme produit.

2.1.2 Événements

L'événement est l'entité du modèle responsable de l'occurrence d'une transition qui modifie l'*état global* du modèle. Un ou plusieurs événements peuvent être associés à une transition. Une transition est provoquée par l'occurrence d'un des ses événements.

Chaque événement doit avoir un taux de franchissement et une probabilité de routage. Le taux de franchissement et la probabilité de routage peuvent avoir des valeurs constantes ou fonctionnelles (Section 2.1.3, page 16). Le non-déterminisme entre le tirage des différents événements associés à une même transition est traité par un comportement Markovien, *i.e.*, tous les événements habilités peuvent se produire et leurs taux de franchissement respectifs définissent la fréquence avec laquelle ils vont se produire.

Deux types d'événements peuvent être modélisés par le formalisme SAN : événements *locaux* ou *synchronisants*.

Événements locaux

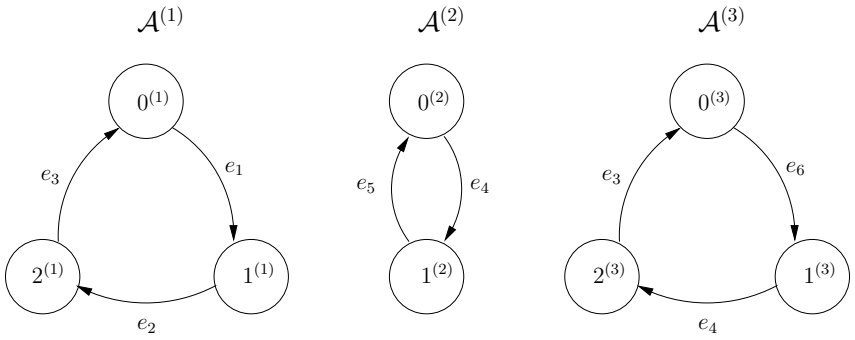
Les événements locaux sont utilisés dans les modèles SAN pour modifier l'état interne (local) d'un *seul automate*, sans que cette modification cause un changement d'état dans autre automate du modèle. Ce type d'événement est particulièrement intéressant, vu qu'il permet que plusieurs automates aient un comportement *parallèle*, travaillant de façon *indépendante* entre eux.

Notamment, on peut observer des exemples d'événements locaux dans FIG. 2.1, qui contient exclusivement par ce type d'événement.

Événements synchronisants

Si les événements locaux sont très simples à définir, les événements synchronisants, au contraire, sont plus complexes. Les événements synchronisants changent l'état local de *deux ou plus automates* simultanément, *i.e.*, l'occurrence d'un événement synchronisant dans un automate *oblige* l'occurrence de ce même événement dans les autres automates concernés. Il est possible de modéliser l'interaction entre des automates en utilisant des événement synchronisants. Cette interaction se fait sous la forme de synchronisme dans le tirage des transitions.

Un événement est classé comme *local* ou *synchronisant* selon l'occurrence de l'identificateur de l'événement e dans l'ensemble des événements d'un automate. Un événement est dit *local* si son identificateur est associé aux transitions d'un *seul automate*. Si l'identificateur d'un événement est associé aux transitions de *plusieurs automates*, cet événement est classé comme *synchronisant*.

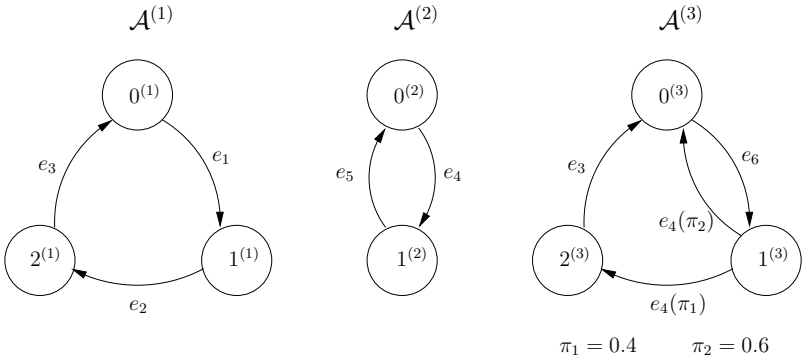


Type	Even.	Taux	Type	Even.	Taux	Type	Even.	Taux
loc	e_1	α	syn	e_4	δ	loc	e_6	μ
loc	e_2	β	loc	e_5	λ			
syn	e_3	γ						

FIG. 2.2 – Modèle SAN avec événements synchronisants

FIG. 2.2 est un modèle SAN modifié par rapport au modèle présenté dans FIG. 2.1. Dans ce nouveau modèle, les événements e_3 et e_4 ne sont plus locaux mais sont des événements synchronisants, car l'identificateur de ces deux événements apparaît dans deux automates. L'événement e_3 synchronise les automates $\mathcal{A}^{(1)}$ et $\mathcal{A}^{(3)}$ de telle façon à changer les états locaux de chaque automate simultanément, *i.e.*, l'occurrence de l'événement e_3 change l'état de l'automate $\mathcal{A}^{(1)}$ de $2^{(1)}$ vers $0^{(1)}$ en même temps qu'elle change aussi l'état de l'automate $\mathcal{A}^{(3)}$ de $2^{(3)}$ vers $0^{(3)}$. De façon analogue, l'occurrence de l'événement e_4 change l'état de l'automate $\mathcal{A}^{(2)}$ de $0^{(2)}$ vers $1^{(2)}$ en même temps qu'elle change l'état de l'automate $\mathcal{A}^{(3)}$ de $1^{(3)}$ vers $2^{(3)}$.

Des événements synchronisants, ainsi que les événements locaux, peuvent avoir des probabilités de routage pour un même événement. Les probabilités de routage sont utilisées pour



Type	Even.	Taux	Type	Even.	Taux	Type	Even.	Taux
loc	e_1	α	syn	e_4	δ	loc	e_6	μ
loc	e_2	β	loc	e_5	λ			
syn	e_3	γ						

FIG. 2.3 – Modèle SAN avec probabilités de routage

déterminer dans quelles proportions les transitions alternatives possibles se produisent. Dans FIG. 2.3, l'événement synchronisant e_4 peut déclencher deux transitions différentes. Ainsi, l'occurrence de l'événement dans l'automate $\mathcal{A}^{(3)}$ déclenche avec probabilité π_1 la transition de l'état $1^{(3)}$ vers l'état $2^{(3)}$, et avec probabilité π_2 la transition de l'état $1^{(3)}$ vers l'état $0^{(3)}$.

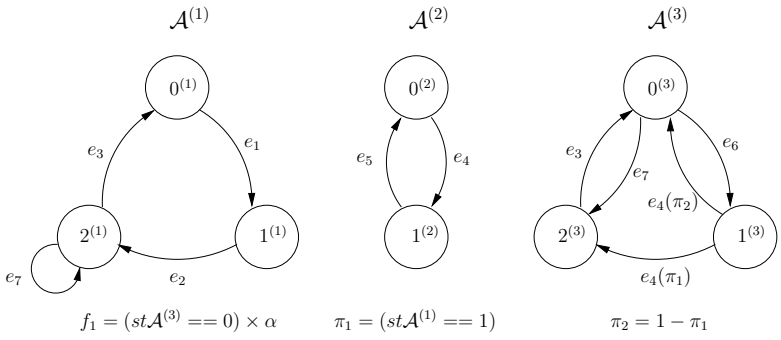
2.1.3 Taux et probabilités fonctionnels

Une possibilité d'exprimer une interaction entre des automates est l'utilisation d'événements synchronisants. Une autre façon est l'utilisation de fonctions pour définir des taux et/ou probabilités qui permet d'associer à un même événement différentes valeurs déterminant son occurrence en fonction de l'état local des autres automates du modèle. L'utilisation de taux et probabilités fonctionnels peut être employée autant que avec des événements locaux ou synchronisants.

Les taux et probabilités fonctionnels sont exprimés par des fonctions qui ont comme paramètres des états courants des automates du modèle. FIG. 2.4 présente une variation du modèle présenté dans FIG. 2.3, où l'événement e_1 possède un taux d'occurrence *fonctionnel* au lieu d'un taux d'occurrence *constant*, *i.e.*, l'événement e_1 se produit avec un taux exprimé par la fonction[4] f_1. Ici, $(st\ \mathcal{A}^{(3)} == 0)$ est la fonction caractéristique de l'ensemble d'états où l'automate $\mathcal{A}^{(3)}$ est dans l'état 0.

Ainsi, la fonction f_1 détermine un taux d'occurrence variable pour l'événement e_1. Selon la fonction f_1, l'occurrence de l'événement e_1 se produira avec le taux suivant :

[4]Dans cette thèse, les fonctions sont définies utilisant le notation adoptée par le logiciel PEPS (Annexe B).



Type	Even.	Taux	Type	Even.	Taux	Type	Even.	Taux
loc	e_1	f_1	syn	e_4	δ	loc	e_6	μ
loc	e_2	β	loc	e_5	λ	syn	e_7	σ
syn	e_3	γ						

FIG. 2.4 – Modèle SAN avec taux fonctionnel

$$f_1 = \begin{cases} \alpha & \text{si l'automate } \mathcal{A}^{(3)} \text{ est dans l'état } 0^{(3)} \\ 0 & \text{si l'automate } \mathcal{A}^{(3)} \text{ est dans l'état } 1^{(3)} \\ 0 & \text{si l'automate } \mathcal{A}^{(3)} \text{ est dans l'état } 2^{(3)} \end{cases}$$

Dans cette thèse et pour définir les fonctions, on utilisera les notations employées par le logiciel PEPS (Annexe B). L'interprétation d'une fonction est l'évaluation d'une expression de langage de programmation non-typé, *e.g.*, langage C. Chaque comparaison d'une fonction est évaluée à 1 (*vrai*) ou 0 (*faux*).

L'évaluation de f_1 donne le résultat en *faux* quand l'automate $\mathcal{A}^{(3)}$ est dans les états $1^{(3)}$ ou $2^{(3)}$, *i.e.*, la valeur résultant pour cette comparaison est *zéro*, qui annule la taux d'occurrence de l'événement e_1. Donc, le taux fonctionnel associé au tirage de l'événement e_1 est tel que, le taux est α si l'état de l'automate $\mathcal{A}^{(3)}$ est égal à $0^{(3)}$ et nul, ce qui empêchera que l'événement e_1 puisse être tiré, pour tous les autres cas.

Les probabilités de routage d'un événement peuvent aussi être exprimées par des fonctions. La définition de fonctions utilisées pour exprimer les probabilités fonctionnelles sont les mêmes que les fonctions utilisées pour exprimer les taux d'occurrence d'un événement.

Dans FIG. 2.4, il faut observer que les probabilités de routage π_1 et π_2 de l'événement e_4 sont aussi exprimées par une fonction. Comme on l'a dit précédemment, la somme des probabilités des transitions d'un événement à partir d'un même état doit être toujours égale à 1 (100%). Donc, de même si les probabilités de routage d'un événement sont exprimées par une fonction, cette caractéristique doit être respectée.

$$\pi_1 = \begin{cases} 1 & \text{si l'automate } \mathcal{A}^{(1)} \text{ est dans l'état } 1^{(1)} \\ 0 & \text{si l'automate } \mathcal{A}^{(1)} \text{ est dans l'état } 0^{(1)} \text{ ou } 2^{(1)} \end{cases}$$

$$\pi_2 = \begin{cases} 0 & \text{si l'automate } \mathcal{A}^{(1)} \text{ est dans l'état } 1^{(1)} \\ 1 & \text{si l'automate } \mathcal{A}^{(1)} \text{ est dans l'état } 0^{(1)} \text{ ou } 2^{(1)} \end{cases}$$

Dans cet exemple, les probabilités π_1 et π_2 s'excluent, *i.e.*, si le résultat de l'évaluation de la probabilité π_1 est égal à 1, le résultat de la probabilité π_2 sera égal à 0, et vice-versa. Cependant, des probabilités exprimées par des fonctions n'ont pas toujours l'obligation de l'exclusion.

2.1.4 Fonction d'atteignabilité

Il y a un autre utilisation des fonctions pour la modélisation dans le formalisme SAN : la *fonction d'atteignabilité*. Les expressions qui définissent la fonction d'atteignabilité sont décrites de la même façon que les fonctions pour les taux et probabilités fonctionnels. Cependant, ce type de fonction a un rôle différent dans le formalisme.

Comme la représentation d'un modèle SAN est faite de façon modulaire et l'automate global (équivalent à chaîne de Markov) est composé de la combinaison de tous les états des automates du modèle (Section 2.1.6, page 19), il faut déterminer une fonction sur le modèle global qui définit les *états atteignables* du modèle SAN.

La définition de quels états peuvent être *atteignables* ou *accessibles* dans un modèle SAN est donnée par la *fonction d'atteignabilité*. Jusqu'à présent, la fonction d'atteignabilité est définie par l'utilisateur. Il faut aussi remarquer que la fonction d'atteignabilité pourrait être calculée à partir d'un état initial et de transitions définies par le modèle.

Cette fonction utilise les règles adoptées pour la définition des taux et probabilités fonctionnels.

La notion de fonction d'atteignabilité est plus claire si on imagine, par exemple, un modèle de partage de ressources avec N clients distincts qui partagent R ressources communes identiques entre elles. Ce système peut être modélisé par le formalisme SAN en utilisant un automate avec deux états pour chaque client. L'état $0^{(i)}$ indique que la ressource n'est pas utilisée par le client i, tant que l'état $1^{(i)}$ indique que la ressource est utilisée par le client i. Il est facile d'imaginer que s'il y a plus de clients que de ressources ($N > R$), l'état global qui représente tous les clients utilisant une ressource ne pourra pas se produire. Les états qui possèdent telle caractéristique sont nommés de *états non-atteignables* et doivent être éliminés du modèle par la *fonction d'atteignabilité*. La probabilité du modèle se trouver dans quelconque de ces états est égale à *zéro*. La fonction d'atteignabilité correcte pour le modèle de partage de ressources décrit ci-dessus est[5] :

$$reachability = nb\,[\mathcal{A}^{(1)}..\mathcal{A}^{(N)}]\ 1 \leq R$$

Ainsi, la fonction d'atteignabilité représente un concept important dans la description de

[5]Notation utilisée par le logiciel PEPS (Annexe B).

modèles SAN. Néanmoins, la complexité de certains modèles peut rendre la description de la fonction d'atteignabilité difficile à réaliser.

2.1.5 Fonction à intégrer

Une autre utilisation des fonctions dans la modélisation avec le formalisme SAN est l'utilisation de *fonctions à intégrer*. Une fonction à intégrer est utilisée pour l'obtention d'indice de performance ou de fiabilité moyens sur le modèle. Ces fonctions évaluent, par exemple, la probabilité du modèle SAN de se trouver dans un état donné. Calculer un indice de performance moyen revient à intégrer cette fonction sur le *vecteur de probabilité* stationnaire ou transitoire du modèle.

Par exemple, avec le modèle de partage de ressources (décrit dans la section 2.1.4), on peut définir la fonction u pour déterminer la probabilité de l'automate $\mathcal{A}^{(1)}$ de ne pas utiliser une ressource (quand l'automate $\mathcal{A}^{(1)}$ se trouve dans l'état $0^{(1)}$).

$$u = \big(st(\mathcal{A}^{(1)}) == 0\big)$$

Toutes les fonctions utilisées dans les modèles SAN sont décrites de la même façon, ce qui les différencie est l'utilisation de chaque fonction dans le modèle.

2.1.6 Construction de la chaîne de Markov équivalente

Bien qu'un modèle SAN soit représenté par un ensemble d'automates, il peut aussi être représenté par un *seul* automate qui contient tous les états globaux possibles du modèle. Cet automate est une représentation de la chaîne de Markov du système modélisé [35]. Une autre représentation est sa matrice de transition, ou bien le descripteur qui sera calculé ultérieurement (Section 2.1.7, page 21).

Dans cet automate global, il n'y a plus d'événements synchronisants, mais seulement des événements locaux. Les arcs de cet automate global sont étiquetés par les taux d'occurrence des événements et les probabilités de routage. Les durées de résidence dans chaque état sont des variables aléatoires de distribution exponentielle. Donc, à un instant de temps, le changement vers un état suivant ne dépend que de l'état courant et pas du temps écoulé dans cet état.

La représentation graphique de la chaîne de Markov équivalente est un graphe à *états-transitions* qui représente l'automate global du système. Les états de la chaîne de Markov sont formés à partir du produit cartésien des états locaux de chaque automate du modèle SAN. Dans FIG. 2.5, on présente la chaîne de Markov (CTMC) équivalente au modèle présenté dans FIG. 2.4, en supposant que l'état global initial est égal à 000.

Pour représenter clairement les états de la CTMC équivalente, on omet les indices de chaque automate, en supposant que le chiffre le plus à gauche représente l'état du premier automate et

le chiffre plus à droite représente l'état du dernier automate, *i.e.*, l'état 201 est équivalent à l'état $2^{(1)}$ dans l'automate $\mathcal{A}^{(1)}$, $0^{(2)}$ dans l'automate $\mathcal{A}^{(2)}$ et $1^{(3)}$ dans l'automate $\mathcal{A}^{(3)}$.

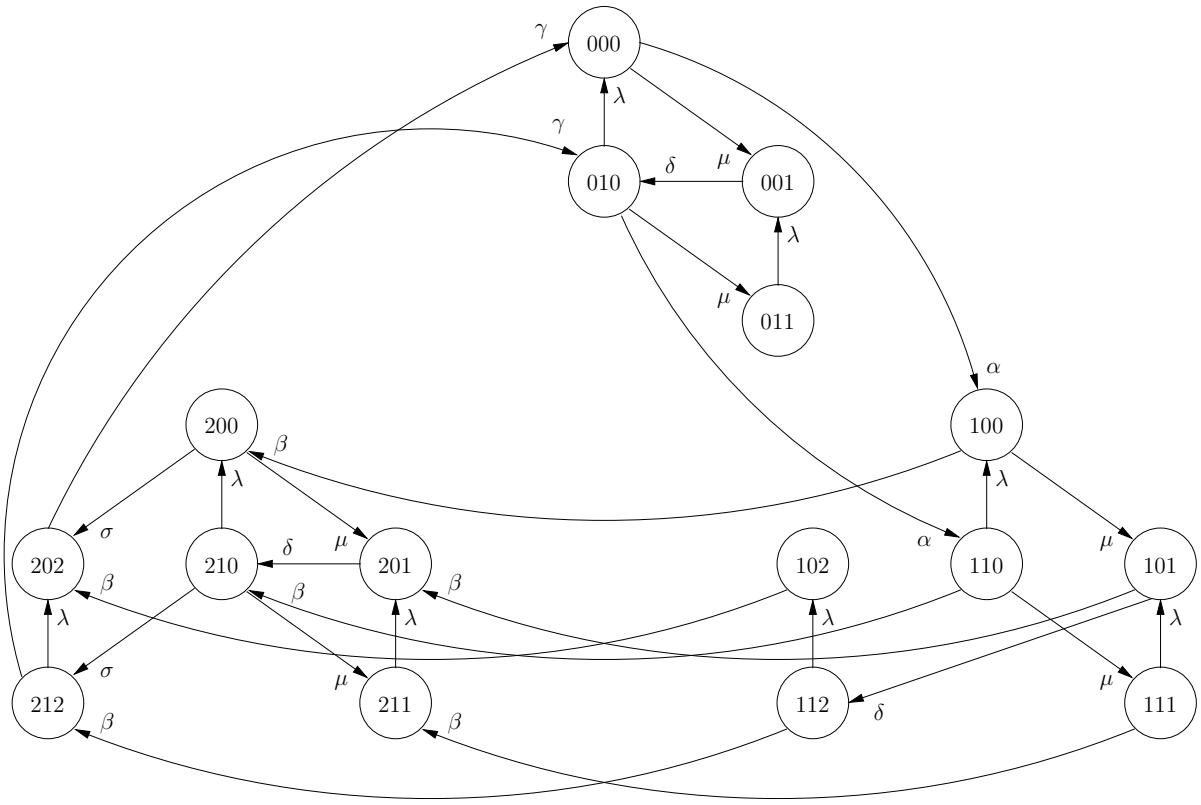


FIG. 2.5 – CTMC équivalente au modèle SAN dans FIG. 2.4

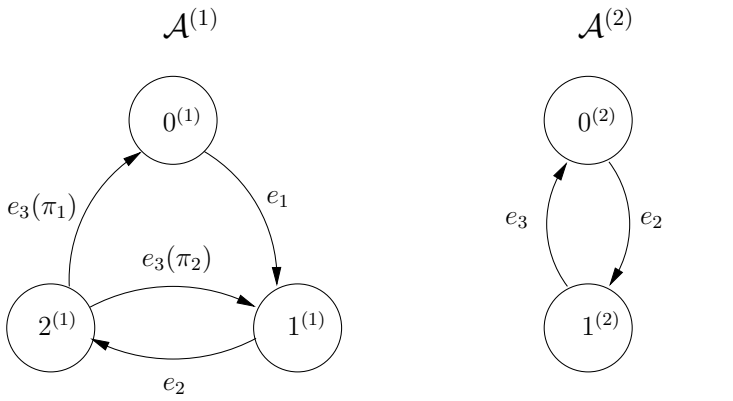
Le formalisme SAN propose une vision modulaire du système, alors que la CTMC équivalente (ou l'automate global) représente une vision "centralisée" (mais potentiellement très grande) du même système. Le générateur infinitésimal de cette chaîne de Markov est représenté directement par une matrice, dont la taille est exprimée par le nombre d'états atteignables du modèle. Comme on peut l'observer dans FIG. 2.4, il y a seulement 16 (sur 18) états atteignables dans le modèle. Les états 002 et 012 ne sont pas représentés, vu qu'ils ne sont jamais *atteignables* dans le modèle (en faisant des tirages successifs des transitions possibles du modèle, on n'atteint jamais ces deux états).

Vu que le formalisme SAN propose une représentation modulaire du système, la représentation du générateur infinitésimal de la chaîne de Markov équivalente au modèle SAN peut aussi être exprimée de façon modulaire. Conséquemment, au lieu de l'obtention d'une seule matrice de transition, on a un ensemble de *petites* matrices pour chaque automate de taille $(n^{(i)})^2$, où $n^{(i)}$ est le nombre d'états de l'automate $\mathcal{A}^{(i)}$. L'obtention du générateur infinitésimal à partir de cet ensemble de petites matrices est exprimée par un format tensoriel, qui est aussi nommé *descripteur Markovien*. Dans ce qui suit, on va démontrer comment on peut obtenir le descripteur Markovien à partir d'un modèle SAN.

2.1.7 Descripteur Markovien

En plus de faciliter la description des modèles, un des avantages du formalisme SAN est de générer automatiquement et d'une façon compacte le générateur infinitésimal Q de la chaîne de Markov équivalente du modèle. Cette façon compacte de représenter le générateur infinitésimal est une formule mathématique nommée *descripteur* Markovien [35, 73].

La formule du descripteur Markovien est basée sur l'algèbre tensorielle. Quand le modèle utilise des taux et/ou probabilités fonctionnels, il est impératif d'utiliser l'algèbre tensorielle généralisée pour la représentation du descripteur Markovien. Cependant, l'utilisation de l'algèbre tensorielle généralisée ne change rien par rapport à la structure de la formule du descripteur. D'autres formalismes, tels que les Réseaux de Petri Stochastiques Généralisés [31] et l'Algèbre de Processus [55] par exemple, utilisent des techniques similaires pour représenter le générateur infinitésimal d'une chaîne de Markov d'une façon compacte et performante.



Type	Even.	Taux
loc	e_1	f_1
syn	e_2	β
syn	e_3	μ

$$f_1 = (st\mathcal{A}^{(2)} == 0) \times \alpha$$

$$\pi_1 = 0.4$$

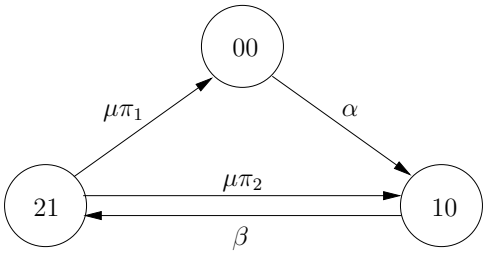
$$\pi_2 = 0.6$$

FIG. 2.6 – Modèle SAN

Pour simplifier la démonstration de la génération du descripteur Markovien d'un modèle SAN, on va utiliser l'exemple présenté dans FIG. 2.6, où l'état global initial est égal à 00. Ce petit exemple a six états potentiels, mais seulement trois des états sont atteignables : 00, 10 et 21.

La chaîne de Markov équivalente au modèle montrée dans FIG. 2.6 est présentée dans FIG. 2.7. Il y a seulement trois états atteignables, et le changement de l'état global 00 vers l'état 10 est exprimé par le taux fonctionnel f_1.

À partir de ce petit exemple, on montre comme s'obtient le descripteur Markovien d'un modèle SAN. La formule mathématique qui représente le descripteur est décomposée en deux parties : *locale* et *synchronisante*. La partie locale représente les transitions des événements



	00	01	10	11	20	21
00	-α	0	α	0	0	0
01	0	0	0	0	0	0
10	0	0	-β	0	0	β
11	0	0	0	0	0	0
20	0	0	0	0	0	0
21	$\mu\pi_1$	0	$\mu\pi_2$	0	0	-μ

FIG. 2.7 – CTMC équivalente au modèle FIG. 2.6

locaux des automates du modèle, tandis que la partie synchronisante représente les transitions de chaque événement synchronisant.

Par rapport à partie locale, à chaque automate est associé une matrice nommée $Q_l^{(i)}$, qui regroupe toutes les transitions des événements locaux de l'automate $\mathcal{A}^{(i)}$. La partie locale du descripteur est représentée par la matrice Q_l qui est définie comme la somme tensorielle des matrices locales de chaque automate. Pour l'exemple présenté dans FIG. 2.6, la partie locale du modèle est représenté par :

$$Q_l = Q_l^{(1)} \underset{g}{\oplus} Q_l^{(2)} = \begin{pmatrix} -f_1 & f_1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \bigoplus_g \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Chaque matrice $Q_l^{(i)}$ a la taille n_i, où n_i est le nombre d'états de l'automate $\mathcal{A}^{(i)}$. Il est intéressant de remarquer que la matrice $Q_l^{(2)}$ est une matrice nulle, *i.e.*, tous les éléments sont égaux à zéro, car l'automate $\mathcal{A}^{(2)}$ (FIG. 2.6) ne possède pas d'événements locaux.

Les matrices locales sont formées à partir des transitions locales de chaque automate. Ainsi, l'élément sur la ligne i et la colonne j de la matrice représente la transition de l'état i vers l'état j de l'automate. Cet élément possède le taux (constant ou fonctionnel) de l'événement local qui correspond à la transition entre les états de l'automate. L'élément diagonal ii (ligne i, colonne i) représente l'ajustement de telle manière que la somme de tous les éléments de la ligne soit égale à *zéro*. L'état global 01 n'est pas un état atteignable (FIG. 2.7). Donc, la transition de l'état 01

vers 11 ne se produit pas. Ainsi, la fonction f_1 est évaluée à 0 pour l'occurrence de l'événement dans l'état 01. En utilisant la somme tensorielle généralisée [35] pour l'obtention de la matrice Q_l, on a la matrice suivante :

$$Q_l = \left(\begin{array}{cc|cc|cc} \text{-}\alpha & 0 & \alpha & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

À la différence de la partie locale du descripteur Markovien qui est obtenue d'une façon directe (une matrice pour chaque automate du modèle), la partie synchronisante est composée, pour chaque événement, de deux ensembles de matrices : *positives* et *négatives*.

Les matrices positives représentent l'occurrence des événements synchronisants, tandis que les matrices négatives représentent l'ajustement diagonal. Donc, chaque événement synchronisant est associé à une paire de matrice (positive et négative) pour représenter son occurrence pour chaque automate concerné. Les matrices positives et négatives synchronisantes associées à l'automate i possèdent aussi la taille n_i (en fonction du nombre d'états de chaque automate $\mathcal{A}^{(i)}$).

La matrice positive $Q^{(i)}_{e_k^+}$ représente l'occurrence de l'événement e_k dans l'automate $\mathcal{A}^{(i)}$. Pour l'exemple dans FIG. 2.6, le modèle SAN a deux événements synchronisants : e_2 et e_3. Donc, les matrices positives pour les événements synchronisants de ce modèle sont :

$$Q_{e_2^+} = Q^{(1)}_{e_2^+} \underset{g}{\otimes} Q^{(2)}_{e_2^+} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & \beta \\ 0 & 0 & 0 \end{pmatrix} \bigotimes_g \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = \left(\begin{array}{cc|cc|cc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & \beta \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

$$Q_{e_3^+} = Q^{(1)}_{e_3^+} \underset{g}{\otimes} Q^{(2)}_{e_3^+} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \mu\pi_1 & \mu\pi_2 & 0 \end{pmatrix} \bigotimes_g \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} = \left(\begin{array}{cc|cc|cc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ \mu\pi_1 & 0 & \mu\pi_2 & 0 & 0 & 0 \end{array}\right)$$

Les matrices positives sont remplies de la même façon que les matrices locales, où l'élément ij représente la transition de l'état i vers l'état j de l'automate. Cependant, un événement synchronisant concerne deux ou plusieurs automates. Donc, pour chaque événement synchronisant, il faut associer deux (ou plus) matrices pour représenter l'occurrence de cet événement.

On peut prendre la convention que la matrice positive correspondante à l'automate de plus petit indice de l'événement reçoit le taux d'occurrence de l'événement. Les autres matrices des automates concernés de cet événement synchronisant reçoivent la valeur 1 pour chaque transition entre les états locaux. Par exemple, pour l'événement synchronisant e_2, la matrice positive $Q^{(1)}_{e_2^+}$ reçoit le taux d'occurrence β dans la position 12 (ligne 1, colonne 2) équivalent à la transition de l'état 1 vers l'état 2 de l'automate $\mathcal{A}^{(1)}$; et la matrice positive $Q^{(2)}_{e_2^+}$ reçoit la valeur 1 dans la position 01 pour représenter la possibilité de la transition de l'état 0 vers l'état 1 de l'automate $\mathcal{A}^{(2)}$ (cf. FIG. 2.6).

De façon analogue, on met en place les transitions globales pour l'événement synchronisant e_3. Néanmoins, cet événement possède des probabilités de routage, qui doivent apparaître dans les matrices. Ainsi, les probabilités π_1 et π_2 apparaissent dans les positions des matrices équivalentes aux transitions qui causent le changement d'état de l'automate.

L'exemple présenté dans FIG. 2.6 est un petit modèle SAN, où les deux événements synchronisants e_2 et e_3 apparaissent dans tous les automates du modèle. Pour des exemples plus complexes, où un événement synchronisant n'apparaît pas dans tous les automates, les matrices positives et négatives de cet événement sont égales à matrices *identités*, *i.e.*, des matrices qui conservent l'état de l'automate pour les automates non concernés par l'occurrence de l'événement.

La somme de toutes les matrices positives des événements synchronisants représente la matrice positive de la partie synchronisante du descripteur Markovien, *i.e.*, pour cet exemple, la partie positive du descripteur est représentée par la matrice Q_{e^+} qui est la somme des matrices positives $Q_{e_2^+}$ et $Q_{e_3^+}$ des événements synchronisants e_2 et e_3 respectivement.

$$Q_{e^+} = Q_{e_2^+} + Q_{e_3^+} = \left(\begin{array}{cc|cc|cc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & \beta \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ \mu\pi_1 & 0 & \mu\pi_2 & 0 & 0 & 0 \end{array}\right)$$

Les matrices négatives des événements sont obtenues de façon à ce que la somme des matrices positives et négatives soit des matrices dont la somme des éléments de chaque ligne soit égale à 0. Les matrices négatives possèdent des éléments non nuls sur les positions diagonales. Les matrices négatives pour les événements synchronisants e_2 et e_3 sont :

$$Q_{e_2^-} = Q^{(1)}_{e_2^-} \underset{g}{\otimes} Q^{(2)}_{e_2^-} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -\beta & 0 \\ 0 & 0 & 0 \end{pmatrix} \bigotimes_g \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = \left(\begin{array}{cc|cc|cc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & -\beta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

$$Q_{e_3^-} = Q^{(1)}_{e_3^-} \underset{g}{\otimes} Q^{(2)}_{e_3^-} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \text{-}\mu \end{pmatrix} \underset{g}{\bigotimes} \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = \left(\begin{array}{cc|cc|cc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \text{-}\mu \end{array}\right)$$

La partie négative est aussi obtenue par la somme de toutes les matrices négatives des événements synchronisants. Donc, la partie négative du descripteur est représenté par la matrice Q_{e^-} :

$$Q_{e^-} = Q_{e_2^-} + Q_{e_3^-} = \left(\begin{array}{cc|cc|cc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & \text{-}\beta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \text{-}\mu \end{array}\right)$$

Le descripteur Markovien est représenté par deux parties : locale et synchronisante. La partie locale Q_l représente les transitions locales et la partie synchronisante, divisée en la partie positive Q_{e^+} et la partie négative Q_{e^-}, représente les transitions des événements synchronisants. Donc, le descripteur Q qui représente un modèle SAN est obtenu par la somme de ses parties :

$$Q = Q_l + Q_{e^+} + Q_{e^-} = \left(\begin{array}{cc|cc|cc} \text{-}\alpha & 0 & \alpha & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & \text{-}\beta & 0 & 0 & \beta \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ \mu\pi_1 & 0 & \mu\pi_2 & 0 & 0 & \text{-}\mu \end{array}\right)$$

Comme le descripteur Markovien sera représenté dans un format tensoriel, il est important de remarquer que seulement les matrices locales $Q_l^{(i)}$ et les matrices positives $Q_{e^+}^{(i)}$ et négatives $Q_{e^-}^{(i)}$ des événements synchronisants sont stockées. Ainsi, seulement les petites matrices de taille n_i sont manipulées en permettant d'avoir une représentation compacte et structurée du modèle. Généralement, le descripteur Markovien Q n'est pas représenté de façon entière, sachant qu'il y a des méthodes numériques performantes pour utiliser le descripteur dans un format tensoriel [35, 8].

2.2 Conclusion

Dans ce chapitre, on a présenté le formalisme SAN à temps continu et ses principales caractéristiques. Le formalisme a comme principal atout une démarche modulaire pour décrire un

système. De plus, l'utilisation d'éléments fonctionnels (tels que des taux et probabilités fonctionnels) permet une plus grande flexibilité au formalisme pour l'expression des interactions entre les composants du système.

On a aussi expliqué comment un SAN représente une chaîne de Markov, et comment un modèle SAN peut être représenté de une façon compacte dans ce contexte. Cette façon compacte de représenter un modèle est possible grâce au format tensoriel du générateur de la chaîne de Markov équivalente au modèle SAN.

Chapitre 3

Disponibilité ponctuelle

3.1 Introduction

Ces dernières années, s'est développé un intérêt croissant pour les méthodes d'analyse transitoire dans le domaine des systèmes sujets aux défaillances. Avec la croissance rapide de la taille mais aussi de la complexité de ces systèmes, l'obtention de mesures de performance transitoires, comme fiabilité et disponibilité, sont souvent difficiles à obtenir.

Les méthodes pour l'obtention de mesures de performance transitoires sont de deux types : celles qui utilisent des modèles qui ne sont pas basées sur une représentation par états et celles qui le sont.

Les méthodes qui ne sont pas basées sur une représentation par états, *e.g.*, *Diagrammes de Blocs* (*Block Diagrams*) ou *Arbre de Fautes* (*Fault Trees*) [11], font l'hypothèse de l'indépendance entre les composants du modèle, ce qui, malheureusement, empêche leur application à la plus grande partie des systèmes où, non seulement les erreurs et les défaillances sont dépendantes entre elles, mais les réparations sont aussi dépendantes.

Les méthodes du deuxième type décrivent les modèles par un ensemble d'états, typiquement des modèles utilisant le formalisme de Chaînes de Markov. Ce formalisme est couramment employé pour la modélisation et l'obtention de mesures de performance de systèmes complexes avec un haut niveau d'interdépendance.

Toutefois, cette approche comporte deux désavantages importants :

– la modélisation des systèmes complexes à partir des descriptions et des spécifications du système sont assez difficile ;
– l'obtention de solutions analytiques pour les distributions transitoires d'un modèle avec grand espace d'états n'est connu que pour un très petit nombre de systèmes.

En effet, l'espace mémoire requis et le temps nécessaire sont fréquemment trop impor-

tants pour résoudre des modèles réels, comme peuvent en attester la myriade de techniques numériques qui essaient de traiter ce problème [92].

Toutefois, l'emploi de formalismes de haut niveau comme les Réseaux de Petri Stochastiques (*Stochastic Petri Nets - SPN*) [31], les Réseaux d'Automates Stochastiques [73] et les Algèbres de Processus (*Process Algebra*) [55] permettent une modélisation du système de façon intuitive et exacte.

À partir de ces formalismes, avec des hypothèses de lois sans mémoire, l'emploi de méthodes comme l'Algèbre Tensorielle (classique ou généralisée) et les MDD (*Multivalued Decision Diagram*) permet le stockage du générateur infinitésimal de la chaîne de Markov sous-jacente de façon compacte. Plusieurs méthodes ont été développées pour l'obtention de mesures de performance de façon à profiter de la représentation tensorielle [94, 35] et des MDD [21].

On s'intéresse dans ce chapitre à l'obtention des solutions analytiques pour des distributions transitoires, telles que des mesures de disponibilité et fiabilité. On définit les mesures de disponibilité (AV) comme la somme des éléments de récompense de la fonction $r(i)$ sur une distribution π :

$$AV = \sum_i r(i)\pi[i] \tag{3.1}$$

La fonction $r(i)$ a la valeur 1 lorsque l'état i appartient à l'ensemble UP et la valeur 0 lorsqu'il appartient à l'ensemble $DOWN$. L'ensemble d'états UP est défini comme l'ensemble d'états où le système est en état de marche et l'ensemble d'états $DOWN$ est défini comme l'ensemble d'états où le système est en panne. $\pi[i]$ est le i-ème élément du vecteur de probabilité π.

Comme on peut le voir sur l'équation (3.1), une fois la fonction de récompense définie et le vecteur de probabilité π obtenu, le calcul de la disponibilité est assez simple et pas très coûteux. En effet, la partie la plus coûteuse de ce calcul est principalement l'obtention du vecteur de probabilité de la distribution (stationnaire ou transitoire) du modèle.

Le plus souvent, les mesures de performance, parmi lesquelles, les mesures de disponibilité et fiabilité [95], sont basées sur la distribution stationnaire de la chaîne de Markov, car il existe des techniques de résolution très efficaces pour l'obtention de la distribution stationnaire de la chaîne.

Par contre, l'obtention des mesures de disponibilité ponctuelle (PAV) sont beaucoup plus difficiles à avoir, car le nombre d'itérations nécessaires pour calculer la distribution transitoire liée aux mesures de disponibilité ponctuelle peut être très important.

Plusieurs méthodes ont été développées pour l'obtention de mesures de disponibilité ponctuelle. Souza e Silva et Gail ont été parmi les premiers à proposer l'utilisation de la méthode d'uniformisation de la chaîne de Markov à temps continu associée à des méthodes de récompense pour calculer la disponibilité ponctuelle avec un contrôle d'erreur efficace [25, 26].

Une autre approche, proposée par Goyal et Tantawi [47], calcule la distribution de la disponibilité à l'instant t récursivement par la discrétisassions de l'intervalle de temps $(0, t)$. Ces intervalles Δt sont suffisamment petits et la probabilité que deux évènements (ou plus) aient lieu pendant Δt est négligeable. Cette approche a été améliorée par Rubino et Sericola en [84], mais les bornes d'erreur ne sont toujours pas connues.

En [83], Rubino et Sericola ont proposé une nouvelle méthode pour calculer la distribution de disponibilité sur un intervalle. Cet algorithme a été amélioré par les méthodes proposées en [85], mais l'analyse est encore faite sur une chaîne de Markov de taille modeste.

Récemment, l'utilisation de techniques d'analyse transitoire, telles que des mesures de disponibilité et fiabilité, en CSL (*Continuous Stochastic Logic*) ont permis l'émergence de bons algorithmes pour calcul de la disponibilité. Malheureusement, ces algorithmes profitent des caractéristiques particulières du modèle et ne sont applicable qu'à un très petit nombre des modèles [101].

Cependant, l'une des meilleures méthodes pour réduire les coûts de calcul de mesures transitoires à l'instant t est la détection du régime stationnaire du modèle lors du calcul de la somme d'une série. La détection du régime stationnaire peut réduire le temps de calcul si le régime stationnaire est détecté plus tôt que l'instant de temps t souhaité.

Dans [20] Ciardo et al. ont proposé une première méthode de détection du régime stationnaire. Cette méthode utilise la différence entre deux vecteurs de probabilité pour détecter la stationnarité.

Une deuxième méthode a été proposée par Sericola [91]. Cette méthode détecte le régime stationnaire par le contrôle de la valeur minimale et maximale d'une suite de vecteurs.

3.1.1 Objectif et nouveauté

L'objectif de ce chapitre est de tester l'efficacité des méthodes de détection de la distribution stationnaire pour le calcul de la disponibilité ponctuelle sur des modèles avec grand espace d'états. Deux méthodes de détection de la distribution stationnaire seront testées. L'une est basée sur la convergence du vecteur de probabilité (méthode définie par Ciardo *et al.* [20]) et l'autre sur le contrôle d'une suite de vecteurs (méthode définié par Sericola [91]).

Pour cela, on propose l'utilisation du formalisme des Réseaux d'Automates Stochastiques pour la modélisation des systèmes. Le formalisme des Réseaux d'Automates Stochastiques a été choisi pour sa facilité de modélisation et l'utilisation du format tensoriel (appelé *Descripteur Markovien*) pour la représentation de la chaîne de Markov. Le *Descripteur Markovien* est une façon compacte de représenter les transitions d'un modèle.

On peut trouver l'emploi du format tensoriel pour la représentation de la chaîne de Markov dans le calcul de distribution transitoire de la chaîne dans des travaux précédents [1, 16]. Toutefois, aucun des ces travaux n'emploie l'utilisation des fonctions, autrement dit, les travaux

développés jusqu'à présent sont basés sur les concepts de algèbre tensorielle classique où les éléments de chaque matrice sont des valeurs constantes. On propose dans ce chapitre l'adaptation des méthodes de calcul de la disponibilité ponctuelle et des méthodes de détection de la distribution stationnaire à une représentation tensorielle qui utilise l'algèbre tensorielle généralisée [34, 35].

Dans la section suivante, on présente des méthodes de calcul de la disponibilité ponctuelle. Dans la section 3.3 on propose l'adaptation des méthodes présentées dans la section 3.2 au format tensoriel. Dans la section 3.4, on présente les méthodes de détection du régime stationnaire. La section 3.5 propose différentes représentations pour le vecteur constant présent dans chaque méthode de calcul de la disponibilité ponctuelle. Enfin, la section 3.6 présente les conclusions de ce chapitre.

3.2 Calcul de la disponibilité ponctuelle

On définit la disponibilité d'un système par la probabilité que ce système fournisse un service à un instant de temps précis.

Si on considère un système modélisé par une chaîne de Markov homogène et irréductible à temps continu X sur un espace d'états fini $\mathcal{S}$, on peut définir une partie de l'espace d'états $\mathcal{S}$ où le système est en train de fournir un service correct. Cette partie définit l'ensemble d'états UP. Une autre partie de $\mathcal{S}$, où le système ne fournit pas de service, définit l'ensemble d'états $DOWN$.

Notons que les ensembles UP et $DOWN$ sont disjoints ($UP \bigcap DOWN = \emptyset$) et complémentaires ($UP \bigcup DOWN = \mathcal{S}$).

Si π est la distribution stationnaire de la chaîne de Markov X, dont la matrice de taux de transition est Q, et $r(i)$ la fonction de récompense du système défini sur les ensembles UP et $DOWN$ telle que $r(i) = 1$ si $i \in UP$ et $r(i) = 0$ si $i \in DOWN$, alors, on peut définir la disponibilité en régime stationnaire par la somme des probabilités de récompense sur la distribution stationnaire de X. On reprend l'équation (3.1), réécrite ci-dessous :

$$AV = \sum_i r(i)\pi[i]$$

où $\pi[i]$ est le i-ème élément du vecteur de probabilité de la distribution stationnaire de la chaîne de Markov X.

De la même manière, en régime transitoire, on définit la disponibilité ponctuelle de la chaîne de Markov à l'instant de temps t par :

$$PAV(t) = \sum_i r(i)\pi(t)[i] \tag{3.2}$$

où $\pi(t)[i]$ est le i-ème élément du vecteur de probabilité de la distribution transitoire de la chaîne de Markov X à l'instant de temps t.

La distribution transitoire de la chaîne de Markov X à l'instant de temps t est définit par l'équation différentielle de Chapman-Kolmogorov [33] :

$$\frac{d\pi(t)}{dt} = \pi(t)Q \tag{3.3}$$

Ce qui nous amène à la solution $\pi(t)$:

$$\pi(t) = \pi(0)e^{Qt}. \tag{3.4}$$

Par conséquent, on calcule $PAV(t)$ par :

$$PAV(t) = \pi(0)e^{Qt}\mathbb{1}_{UP}. \tag{3.5}$$

Où $\pi(0)$ est le vecteur de la distribution initial et $\mathbb{1}_{UP}$ est la fonction indicatrice de l'ensemble d'états UP.

Cependant, la solution directe de e^{Qt} ne conduit pas à de bons algorithmes [29]. Nous utilisons alors une *méthode d'uniformisation* [92], aussi connue sous les noms de *méthode de Jensen* ou *méthode de randomisation*.

3.2.1 Méthode d'uniformisation

La méthode d'uniformisation décompose la chaîne de Markov à temps continu en une chaîne de Markov à temps discret et un processus de Poisson. La matrice de probabilité de transition P représente la chaîne de Markov à temps discret et est obtenue par uniformisation de la matrice Q. On obtient la matrice P par :

$$P = I + \frac{Q}{\lambda} \tag{3.6}$$

où λ est la plus grande valeur des éléments diagonaux du générateur infinitésimal Q ($\lambda \geq max\{q_{ii}\}, i \in \mathcal{S}$), aussi connue sous le nom de *taux d'uniformisation* et I est la matrice identité de la même taille que Q.

De cette manière, on peut obtenir $\pi(t)$ par :

$$\pi(t) = \pi(0) \sum_{n=0}^{\infty} e^{-\lambda t} \frac{(\lambda t)^n}{n!} P^n \tag{3.7}$$

Par conséquent, on peut réécrire l'équation de la disponibilité ponctuelle (Équation (3.5)) par :

$$\begin{aligned} PAV(t) &= \pi(0) \sum_{n=0}^{\infty} e^{-\lambda t} \frac{(\lambda t)^n}{n!} P^n \mathbb{1}_{UP} \\ &= \sum_{n=0}^{\infty} e^{-\lambda t} \frac{(\lambda t)^n}{n!} \pi(0) P^n \mathbb{1}_{UP} \end{aligned} \tag{3.8}$$

Cependant, l'élévation de la matrice P à la puissance n continue à être très coûteuse à calculer, même avec l'emploi de méthodes itératives efficaces.

Élévation de la matrice P à la puissance n

La façon la plus naïve d'élever la matrice P à la puissance n est par multiplications successives de la matrice. Cette méthode multiplie successivement la matrice P par elle même n fois.

$$(\ldots((P \times P) \times P) \ldots \times P)$$

Toutefois, cette méthode coûte très cher, donné qu'il faut $n(|\mathcal{S}|^3)$ multiplications pour obtenir P^n, où $|\mathcal{S}|$ est la taille de matrice P.

Dans le cas du calcul de la disponibilité ponctuelle, on a la possibilité d'utiliser des méthodes itératives moins coûteuses. Ces méthodes consistent à élever la matrice P à la puissance n par multiplications successives entre un vecteur et la matrice.

Plus précisément, dans le cas du calcul de la disponibilité ponctuelle, on a deux techniques possibles :

Multiplication à gauche On calcule récursivement un vecteur v_n par la multiplication du vecteur v_{n-1} par la matrice P.

$$v_n = \begin{cases} v_0 = \pi(0) \\ v_n = v_{n-1} P \end{cases} \tag{3.9}$$

On pourrait alors réécrire la distribution transitoire à l'instant de temps t (Équation (3.7)) et la disponibilité ponctuelle (Équation (3.8)), respectivement par :

$$\pi(t) = \sum_{n=0}^{\infty} e^{-\lambda t} \frac{(\lambda t)^n}{n!} v_n \tag{3.10}$$

et

$$PAV(t) = \sum_{n=0}^{\infty} e^{-\lambda t} \frac{(\lambda t)^n}{n!} v_n \mathbb{1}_{UP} \tag{3.11}$$

Cette méthode sera appelée *multiplication à gauche*, car elle explore la propriété de l'associativité de la multiplication par des multiplications successives du vecteur v_n par la matrice P à gauche.

$$(\ldots((\pi(0) \times P) \times P) \ldots \times P)$$

Multiplication à droite Cette méthode utilise le principe de la multiplication par un vecteur à droite pour l'élévation de la matrice P à la puissance n. On exploite la propriété de l'associativité de la multiplication en multipliant successivement la matrice P par un vecteur w_n à droite :

$$(P \times \ldots (P \times (P \times \mathbb{1}_{UP})) \ldots)$$

Récursivement on calcule w_n par :

$$w_n = \begin{cases} w_0 = \mathbb{1}_{UP} \\ w_n = P w_{n-1} \end{cases} \tag{3.12}$$

On remarque que, dans ce cas là, on n'obtient plus le vecteur de la distribution transitoire $\pi(t)$, mais on peut encore réécrire équation de la disponibilité ponctuelle par :

$$PAV(t) = \sum_{n=0}^{\infty} e^{-\lambda t} \frac{(\lambda t)^n}{n!} \pi(0) w_n \tag{3.13}$$

Méthode de troncature de Fox-Glynn

De façon à obtenir une version plus facilement calculable, Fox et Glynn [43] ont défini une méthode pour tronquer cette somme infinie à une valeur minimale K_ε telle que :

$$\sum_{n=0}^{K_\varepsilon} e^{-\lambda t}\frac{(\lambda t)^n}{n!} < (1-\epsilon) \tag{3.14}$$

De cette manière, on obtient $PAV_\varepsilon(t)$, une approximation à ε près de la distribution transitoire à l'instant de temps t :

$$\begin{aligned} PAV_\varepsilon(t) &= \sum_{n=0}^{K_\varepsilon} e^{-\lambda t}\frac{(\lambda t)^n}{n!} v_n \mathbb{1}_{UP} \\ &= \sum_{n=0}^{K_\varepsilon} e^{-\lambda t}\frac{(\lambda t)^n}{n!} \pi(0) w_n \end{aligned} \tag{3.15}$$

Algorithme classique Cette méthode de troncature rend possible l'implantation de plusieurs algorithmes. Dans la suite, on va revoir brièvement deux versions de l'algorithme classique pour le calcul de la disponibilité ponctuelle.

L'algorithme 3.1 présente le calcul de la disponibilité ponctuelle basé sur la méthode d'élévation de la matrice P par multiplication à gauche.

ALG. 3.1 $\mathrm{PAV}_\varepsilon(t)$ - Multiplication à gauche

Input : $0 < t_1 < \cdots < t_\ell$, ε
Output : $\mathrm{PAV}_\varepsilon(t_1), \ldots, \mathrm{PAV}_\varepsilon(t_\ell)$
Compute K_ε par Éq. (3.14)
$v_0 = \pi(0)$
$V_0 = v_0 \mathbb{1}_{UP}$
for $n = 1$ to K_ε) **do**
    $v_n = v_{n-1} P$
    $V_n = v_n \mathbb{1}_{UP}$
end for
for i=1 to ℓ **do**
    $\mathrm{PAV}_\varepsilon(t_i) = \sum_{n=0}^{K_\varepsilon} e^{-\lambda t_i}\frac{(\lambda t_i)^n}{n!} V_n$
end for

Basé sur la même méthode, l'algorithme 3.2 implante le calcul de la disponibilité ponctuelle basé sur l'élévation de la matrice P par la multiplication à droite.

ALG. 3.2 $\mathrm{PAV}_\varepsilon(t)$ - Multiplication à droite

Input : $0 < t_1 < \cdots < t_\ell$, ε
Output : $\mathrm{PAV}_\varepsilon(t_1), \ldots, \mathrm{PAV}_\varepsilon(t_\ell)$
Compute K_ε par Éq. (3.14)
$w_0 = \mathbb{1}_{UP}$
$W_0 = \pi(0) w_0$
for $n = 1$ to K_ε **do**
    $w_n = P w_{n-1}$
    $W_n = \pi(0) w_n$
end for
for i=1 to ℓ **do**
    $\mathrm{PAV}_\varepsilon(t_i) = \sum_{n=0}^{K_\varepsilon} e^{-\lambda t_i} \frac{(\lambda t_i)^n}{n!} W_n$
end for

3.3 Adaptation des algorithmes au format tensoriel

Rappelons que la matrice P est la version uniformisée de la matrice Q et que Q et P sont représentées par un descripteur markovien [35, 34]. L'utilisation d'un format tensoriel pour la représentation du modèle entraîne l'emploi des méthodes de multiplication vecteur-matrice adaptées de façon à toujours garder le format compact de la matrice. La multiplication d'un vecteur par le générateur infinitésimal du modèle (représenté ici par la matrice P) est l'opération fondamentale pour l'élévation de la matrice P à la puissance n. On présente ci-dessous les deux méthodes de multiplication utilisées dans les algorithmes 3.1 et 3.2 adaptées au format tensoriel.

3.3.1 Multiplication vecteur-descripteur - Multiplication à gauche

La multiplication d'un vecteur v par un descripteur P est décrit par :

$$vP = v \sum_{j=1}^{(N+2E)} \left[\bigotimes_{i=1}^{N}{}_g P_j^{(i)} \right] = \sum_{j=1}^{(N+2E)} \left[v \bigotimes_{i=1}^{N}{}_g P_j^{(i)} \right]$$

donc, l'opération qui nous intéresse est la multiplication d'un vecteur par un terme produit tensoriel généralisé, soit :

$$v \bigotimes_{i=1}^{N}{}_g P^{(i)}$$

où les indices j ont été omis pour les matrices $P_j^{(i)}$ afin de simplifier les notations. Chaque

terme est composé d'une suite de N matrices notées $P^{(i)}$ avec $i \in [1..N]$.

Selon la propriété de décomposition des produits tensoriels [34] tout produit tensoriel d'un terme tensoriel de N matrices est équivalent au produit de N facteurs normaux.

Cela nous permet réécrire le terme $\bigotimes_{i=1}^{N} P^{(i)}$ sous la forme :

$$\begin{aligned} P^{(1)} \otimes P^{(2)} \otimes \ldots \otimes P^{(N-1)} \otimes P^{(N)} = \\ & P^{(1)} \otimes I_{nright_1} \\ & \times I_{nleft_2} \otimes P^{(2)} \otimes I_{nright_2} \\ & \times \cdots \\ & \times I_{nleft_{N-1}} \otimes P^{(N-1)} \otimes I_{nright_{N-1}} \\ & \times I_{nleft_N} \otimes P^{(N)} \end{aligned}$$

✍ **Avec**

n_i	la dimension de la i-ème matrice de la suite $P^{(i)}$, $i \in [1..N]$;
$nleft_i$	le produit des dimensions de toutes les matrices à gauche de la i-ème matrice de la suite, *i.e.*, $\prod_{k=1}^{i-1} n_k$ (cas particulier : $nleft_1 = 1$) ;
$nright_i$	le produit des dimensions de toutes les matrices à droite de la i-ème matrice de la suite, *i.e.*, $\prod_{k=i+1}^{N} n_k$ (cas particulier : $nright_N = 1$) ;
$\bar{n}_i$	le produit des dimensions de toutes les matrices sauf la i-ème matrice d'une suite, *i.e.*, $\prod_{k=1,k\neq i}^{N} n_k$ ($\bar{n}_i = nleft_i \; nright_i$) ;

La multiplication d'un vecteur v par le terme $\bigotimes_{i=1}^{N} P^{(i)}$ est faite par les multiplications enchaînées de chaque facteur normal. Le vecteur v doit être multiplié par le premier facteur normal, le résultat est multiplié par le deuxième facteur normal et ainsi de suite jusqu'au dernier des facteurs normaux. Donc, pour multiplier un vecteur v par un terme $\bigotimes_{i=1}^{N} P^{(i)}$, il est nécessaire et suffisant de savoir multiplier un vecteur par un facteur normal. Ceci est possible grâce à la propriété d'associativité de la multiplication (traditionnelle) de matrices.

Dans la suite, on explique la méthode de multiplication d'un vecteur par un facteur normal. Cette présentation a été inspiré dans le chapitre 5 de la thèse de Fernandes[34]. La méthode de multiplication proposée dans ces travaux est connue sous le nom *shuffle*.

Multiplication du dernier facteur normal

Comme il a été montré dans le chapitre 5 de la thèse de Fernandes[34], le cas le plus simple est celui de la multiplication du dernier facteur normal. Où on a :

$$v \ \times I_{nleft_N} \otimes P^{(N)} \tag{3.16}$$

La matrice $I_{nleft_N} \otimes P^{(N)}$ étant une matrice bloc diagonale (FIG. 3.1), il suffit de multiplier $nleft_N$ tranches de taille n_N du vecteur v par la matrice $P^{(N)}$.

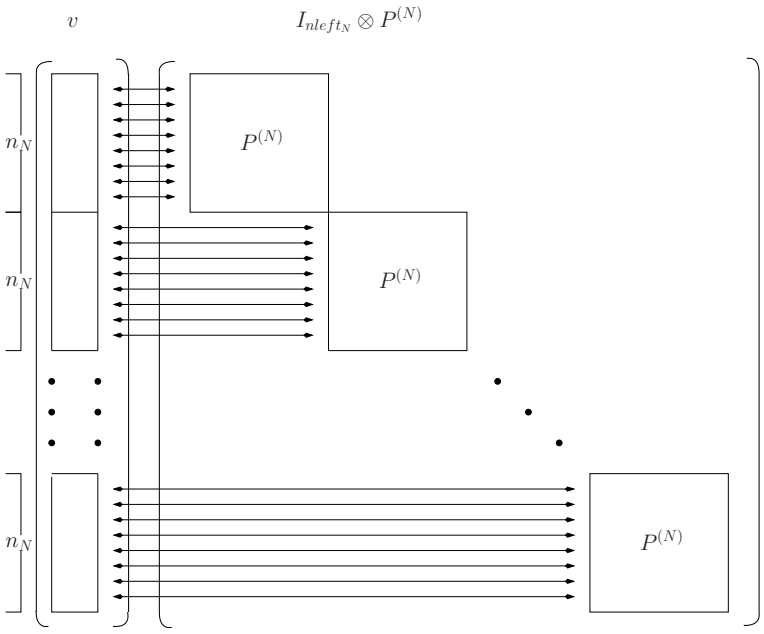


FIG. 3.1 – Multiplication $v \times I_{nleft_N} \otimes P^{(N)}$

FIG. 3.2 présente la technique de découpage des composantes du vecteur v pour la remplissage des vecteurs z_{in}. Chaque vecteur z_{in} (morceau du vecteur v) est multiplié par la matrice $P^{(N)}$ (dans le cas du dernier facteur normal).

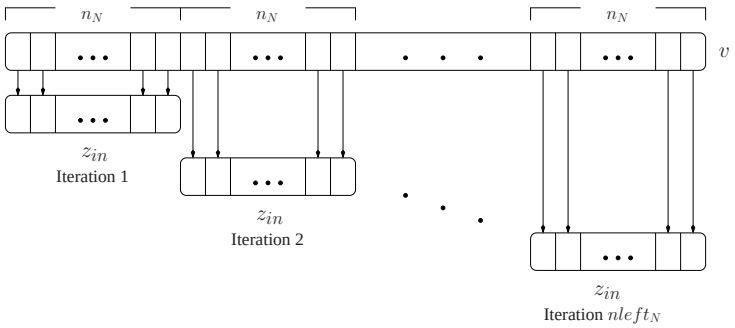


FIG. 3.2 – Principe de l'algorithme pour multiplier v par le dernier facteur normal

L'algorithme 3.3 présente la construction des tranches de taille n_N du vecteur v et leur multiplication par $P^{(N)}$ comme décrit dans FIG. 3.2.

ALG. 3.3 Multiplication $v \times I_{nleft_N} \otimes P^{(N)}$

```
1: base = 0 ;
2: for k = 1, 2, ..., nleft_N do
3:    index = base + 1 ;
4:    for l = 1, 2, ..., n_N do
5:       z_in[l] = v[index] ;
6:       index = index + 1 ;
7:    end for
8:    multiply z_out = z_in × P^(N) ;
9:    index = base + 1 ;
10:   for l = 1, 2, ..., n_N do
11:      v[index] = z_out[l] ;
12:      index = index + 1 ;
13:   end for
14:   base = base + n_N ;
15: end for
```

Multiplication par le premier facteur normal

Pour la multiplication par le premier facteur normal, on va d'abord présenter la propriété de pseudo-commutativité des facteurs normaux.

$$\bigotimes_{k=1}^{N} P^{(k)} = O_\sigma \times \left[\bigotimes_{k=1}^{N} P^{(\sigma_k)} \right] \times {O_\sigma}^T \tag{3.17}$$

où σ est une permutation sur l'intervalle $[1..N]$ et O_σ est une matrice de permutation.

Rappelons que pour la multiplication par le premier facteur normal, on part de :

$$v \ \times P^{(1)} \otimes I_{nright_1} \tag{3.18}$$

Grâce à cette propriété de pseudo-commutativité, on peut, par une permutation O_σ sur les composantes de v, se ramener à :

$$O_\sigma \times \ I_{nright_1} \otimes P^{(1)} \ {O_\sigma}^T$$

où σ est la permutation qui passe l'ordre de la suite de matrices de $\{1, 2, 3, \ldots, N\}$ à $\{2, 3, \ldots, N, 1\}$ et O_σ est la matrice de permutation qui permet de réordonner les éléments du vecteur v de façon à ce que les éléments de chaque tranche z_{in} soient dans un espace continu du vecteur.

Sous cette forme, la multiplication par le premier facteur normal (Équation (3.18)) peut

être calculée de façon analogue au cas précédent, la multiplication du dernier facteur normal (Équation (3.16)).

La multiplication d'un vecteur v est exécutée en trois étapes :

– la multiplication de v par la matrice de permutation O_σ ;
– la multiplication de $(v \times O_\sigma)$ par le facteur normal commuté $I_{nright_1} \otimes P^{(1)}$; et
– la multiplication du résultat des étapes précédentes par la matrice de permutation ${O_\sigma}^T$.

La première étape correspond à une permutation du vecteur v. Il est possible d'exécuter cette permutation lors de l'extraction de tranches du vecteur v, *i.e.*, dans les remplissages des vecteurs z_{in} de l'algorithme (lignes 3 à 7 de l'algorithme 3.3). Dans le cas non permuté (multiplication du dernier facteur normal), le vecteur z_{in} est rempli avec des tranches successives de taille n_N. La permutation nécessaire pour le premier facteur normal, en revanche, équivaut à accéder au vecteur en prenant un élément à chaque intervalle de taille $nright_1$. FIG. 3.3 représente le processus de permutation, qu'il faut comparer au processus équivalent pour le cas sans permutation (décrit dans FIG. 3.2).

La deuxième étape est identique au cas précédent, il s'agit simplement de la multiplication répétée des tranches du vecteur v par la matrice $P^{(1)}$. La troisième étape est la permutation symétrique à la première et peut être exécutée lors du stockage des éléments du vecteur temporaire z_{out} (lignes 9 à 13 de l'algorithme 3.3) dans le vecteur v. L'algorithme pour cette multiplication est numéroté 3.4.

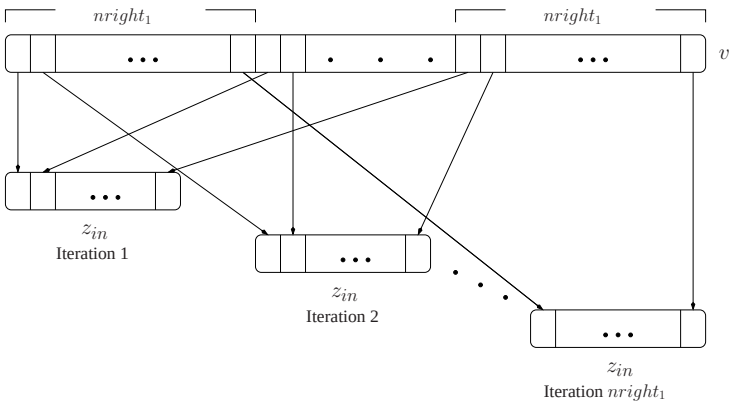


FIG. 3.3 – Permutations exécutées lors de la multiplication du premier facteur normal

Multiplication complète

Les autres facteurs normaux (du deuxième jusqu'à l'avant-dernier) sont traités comme des combinaisons des deux cas précédents. La technique de base consiste toujours à appliquer la

ALG. 3.4 Multiplication $v \times P^{(1)} \otimes I_{nright_1}$

```
1:  base = 0 ;
2:  for j = 1, 2, ..., nright_1 do
3:     index = base + j ;
4:     for l = 1, 2, ..., n_1 do
5:        z_in[l] = v[index] ;
6:        index = index + nright_1 ;
7:     end for
8:     multiply z_out = z_in × P^(1) ;
9:     index = base + j ;
10:    for l = 1, 2, ..., n_1 do
11:       v[index] = z_out[l] ;
12:       index = index + nright_1 ;
13:    end for
14: end for
```

propriété de pseudo-commutativité (Équation (3.17), page 38) au facteur normal :

$$I_{nleft_i} \otimes P^{(i)} \otimes I_{nright_i} = O_\sigma \times \left(I_{nleft_i} \otimes I_{nright_i} \otimes P^{(i)}\right) {O_\sigma}^T$$

où σ est la permutation qui passe de $\{1, \ldots, i-1, i, i+1, \ldots, N\}$ à $\{1, \ldots, i-1, i+1, \ldots, N, i\}$.

Ceci amène à toujours multiplier par des facteurs normaux de la forme :

$$I_{nleft_i} \otimes I_{nright_i} \otimes P^{(i)} = I_{\bar{n}_i} \otimes P^{(i)}$$

Les permutations sont faites lors de l'extraction et le stockage des résultats des tranches z_{in} selon la position de la matrice $P^{(i)}$. L'algorithme 3.5 implante la multiplication d'un vecteur v par un terme produit tensoriel complet $\otimes_{i=1}^{N} P^{(i)}$.

Dans cet algorithme, les facteurs normaux sont traités du premier jusqu'au dernier. Pourtant, dans le cas de l'algèbre tensorielle classique, comme ceux présentés ci-avant, la propriété de commutativité de facteurs normaux [34], permet la multiplication des facteurs normaux dans n'importe quel ordre. Cependant, cette propriété n'est plus valable pour algèbre tensorielle généralisée où, à cause de dépendances fonctionnelles, il faut que les facteurs normaux soient traités dans un certain ordre.

3.3.2 Traitement des dépendances fonctionnelles

La multiplication d'un produit tensoriel généralisé introduit deux étapes supplémentaires aux algorithmes précédents :

- calculer un ordre σ dans lequel les facteurs normaux doivent être arrangés ;
- évaluer les éléments fonctionnels des matrices avant chaque multiplication.

ALG. 3.5 Multiplication $v \times \otimes_{i=1}^{N} P^{(i)}$

```
1: for i = 1, 2, ..., N do
2:    base = 0 ;
3:    for k = 1, 2, ..., nleft_i do
4:       for j = 1, 2, ..., nright_i do
5:          index = base + j ;
6:          for l = 1, 2, ..., n_i do
7:             z_in[l] = v[index] ;
8:             index = index + nright_i ;
9:          end for
10:         multiply z_out = z_in × P^(i) ;
11:         index = base + j ;
12:         for l = 1, 2, ..., n_i do
13:            v[index] = z_out[l] ;
14:            index = index + nright_i ;
15:         end for
16:      end for
17:      base = base + ( nright_i × n_i)
18:   end for
19: end for
```

Établissement de l'ordre de décomposition des facteurs normaux

Pour établir un ordre de décomposition des facteurs normaux, on va d'abord présenter les règles de décomposition pour les deux cas possibles.

Considérons deux matrices A et $B(\mathcal{A})$, on décompose $A \underset{g}{\otimes} B(\mathcal{A})$ en :

$$A \underset{g}{\otimes} B(\mathcal{A}) = I_{n_A} \underset{g}{\otimes} B(\mathcal{A}) \;\times\; A \underset{g}{\otimes} I_{n_B} \tag{3.19}$$

Considérons deux matrices $A(\mathcal{B})$ et B, on décompose $A(\mathcal{B}) \underset{g}{\otimes} B$ en :

$$A(\mathcal{B}) \underset{g}{\otimes} B = A(\mathcal{B}) \underset{g}{\otimes} I_{n_B} \;\times\; I_{n_A} \underset{g}{\otimes} B \tag{3.20}$$

Selon ces règles de décomposition en facteurs normaux des produits tensoriels généralisés, le facteur normal d'une matrice $P^{(i)}$ doit toujours précéder les facteurs normaux des matrices $P^{(j)}(\ldots, \mathcal{A}^{(i)}, \ldots)$, *i.e.,* les facteurs normaux des matrices qui dépendent de l'automate $\mathcal{A}^{(i)}$. Ceci définit un relation de précédence entre les facteurs normaux.

Il faut remarquer que les cas présentés ci-dessous n'ont pas des cycles de dépendances, *i.e.*, $A(\mathcal{B}) \underset{g}{\otimes} B(\mathcal{A})$. Toutefois, un cycle de dépendance peut être décomposé de façon à devenir une

somme de termes tensoriels avec des dépendances fonctionnelles sans cycle. Le traitement de cycles de dépendances est présenté en [34] et ne sera pas présenté ici.

Cette relation de dépendance acyclique est donc un ordre partiel. L'établissement de l'ordre des facteurs normaux est fait par un tri topologique sur le graphe de dépendances fonctionnelles des matrices[1] $P^{(i)}(\ldots)$.

Précisément, dans la construction du graphe de dépendances fonctionnelles d'un terme tensoriel, chaque automate $\mathcal{A}^{(i)}$ est représenté par un sommet. Chaque sommet indique quelles sont les dépendances fonctionnelles d'un matrice $P^{(i)}(\ldots)$. Lorsqu'une matrice $P^{(i)}(\ldots,\mathcal{A}^{(j)},\ldots)$ a un élément fonctionnel $f(\mathcal{A}^{(j)})$ qui dépend de l'état de l'automate $\mathcal{A}^{(j)}$, cette dépendance fonctionnelle est représentée par un arc orienté du sommet i (automate $\mathcal{A}^{(i)}$) vers le sommet j (automate $\mathcal{A}^{(j)}$). Un sommet sans arc sortant représente une matrice qui n'a aucun élément fonctionnel. De la même façon, un sommet sans arc entrant représente une matrice dont l'état de l'automate correspondant n'est jamais argument d'une autre matrice du terme tensoriel.

Le tri topologique de ce graphe permet d'extraire un ordre de décomposition total compatible avec l'ordre partiel défini par ce graphe par un algorithme de tri topologique [10]. Ce tri topologique est présenté par Fernandes [34] (Chapitre 5) et ne sera pas présenté ici.

Par exemple, les termes tensoriels avec les dépendances suivantes :

$$\bigotimes_{i=1}^{N}{}_{g} P^{(i)}(\mathcal{A}^{(1)},\ldots,\mathcal{A}^{(i-1)}) \text{ (appelé cas 1) et}$$

$$\bigotimes_{i=1}^{N}{}_{g} P^{(i)}(\mathcal{A}^{(i+1)},\ldots,\mathcal{A}^{(N)}) \text{ (appelé cas 2)}$$

sont représentés par les graphes dans FIG. 3.4.

Les cas 1 et 2 sont des cas où il faut appliquer les règles de décomposition en facteurs normaux (soit (3.20) pour le cas 1, soit (3.19) pour le cas 2). Ces deux cas donnent respectivement :

$$\prod_{i=N}^{1}\left[I_{nleft_i} \underset{g}{\otimes} P^{(i)}(\mathcal{A}^{(1)},\ldots,\mathcal{A}^{(i-1)}) \underset{g}{\otimes} I_{nright_i}\right] \text{ (pour le cas 1) et}$$

$$\prod_{i=1}^{N}\left[I_{nleft_i} \underset{g}{\otimes} P^{(i)}(\mathcal{A}^{(i+1)},\ldots,\mathcal{A}^{(N)}) \underset{g}{\otimes} I_{nright_i}\right] \text{ (pour le cas 2).}$$

[1] La notation $P^{(i)}(\ldots)$ est utilisée pour décrire de manière générique des matrices comportant différentes dépendances fonctionnelles.

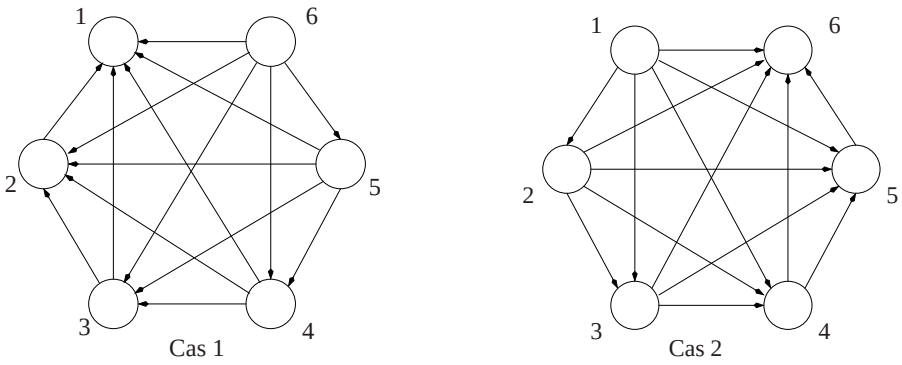


FIG. 3.4 – Graphes de dépendances fonctionnelles pour les cas 1 et 2

Notons que, entre ces deux cas, la seule différence est l'ordre de multiplication des facteurs normaux[2]. Il est utile de remarquer que les ordres de décomposition des facteurs normaux définis pour les cas 1 et 2 sont obligatoires, *i.e.*, la seule décomposition en facteurs normaux valide est celle utilisée (de N à 1 pour le cas 1 et de 1 à N pour le cas 2).

Le cas général reste l'application alternée des propriétés (3.19) et (3.20) qui permet l'établissement, en général, d'un ordre de décomposition pour la multiplication des facteur normaux. Cet ordre va être caractérisé par une permutation σ sur l'intervalle $[1..N]$ qui permet de passer de l'ordre de définition du modèle $[1..N]$ au nouvel ordre après tri topologique. Pour cela utilisons les notations suivantes :

✍ **Notons**

σ	une permutation nommée σ sur l'intervalle $[1..N]$. Une permutation σ établit une nouvel ordre pour une suite de N matrices ;
$\sigma(i)$	le rang de la matrice $P^{(i)}$ dans l'ordre représenté par la permutation σ ;
σ_k	l'indice de la matrice placée au rang k de l'ordre représenté par la permutation σ (si $\sigma_k = i$, $\sigma(i) = k$) ;

Le terme tensoriel $\bigotimes_{g\,i=1}^{N} P^{(i)}(\ldots)$ doit être traité selon la formule suivante :

$$\prod_{i=1}^{N} \left[I_{nleft_{\sigma_i}} \underset{g}{\otimes} P^{(\sigma_i)}(\ldots) \underset{g}{\otimes} I_{nright_{\sigma_i}} \right]$$

Cet *ordre de décomposition* oblige un changement dans l'algorithme 3.5 comme il est indiqué dans l'algorithme 3.6. Notons que ce changement demande le calcul d'un ordre de traitement (permutation σ) qui n'est pas inclus dans l'algorithme 3.6.

L'ordre dans lequel les facteurs normaux doivent être multipliés est décrit par la permutation

[2] Il faut rappeler que la multiplication de facteurs normaux avec éléments fonctionnels n'est pas commutative

ALG. 3.6 Multiplication $v \times \bigotimes_{g_{i=1}}^{N} P^{(i)}(\ldots)$

1: **for** $i = \sigma_1, \sigma_2, \ldots, \sigma_N$ **do**
⋮

σ et sera appelé *ordre de décomposition*. Cet ordre est l'ordre dans lequel le produit tensoriel est décomposé en facteurs normaux selon les propriétés (3.19) et (3.20).

Évaluation des éléments fonctionnels

Après le calcul de l'*ordre de décomposition*, la deuxième préoccupation pour la multiplication des produits tensoriels généralisés est l'évaluation des éléments fonctionnels de chaque matrice $P^{(i)}(\ldots)$ avant leur multiplication par une tranche z_{in} du vecteur v (correspondant à la ligne 10 de l'algorithme 3.5).

À chaque exécution de la multiplication d'une tranche z_{in} du vecteur v par la matrice $P^{(i)}(\ldots)$, cette matrice doit être évaluée pour les états des automates arguments de $P^{(i)}$. L'algorithme 3.7 apporte les modifications nécessaires (ligne 3, 6, 12, 19 et 22) à l'évaluation des éléments fonctionnels[3]. La méthode d'évaluation des éléments fonctionnels est décrite en détail par Fernandes [34] (Chapitre 5).

3.3.3 Multiplication descripteur-vecteur - Multiplication à droite

Pour la multiplication descripteur-vecteur (*multiplication à droite*), on utilise la même idée de base que pour la méthode de multiplication vecteur-descripteur (*multiplication à gauche*). Autrement dit, on utilise de façon identique la décomposition du terme tensoriel en facteurs normaux et l'application de permutations de façon à ce qu'on puisse toujours multiplier par un facteur normal sous la forme $I_{\bar{n}_i} \otimes P^{(i)}$. Cependant, quelques adaptations sont nécessaires à l'algorithme du *shuffle* original à cause de l'utilisation de l'algèbre tensorielle généralisée.

Plusieurs travaux [61, 16] font référence à la possibilité de l'utilisation de la multiplication à droite pour le calcul de la disponibilité ponctuelle, soit directement sur des chaînes de Markov, soit sur des formalismes structurés utilisant la représentation tensorielle de la matrice [16].

Toutefois, il faut souligner que tous les travaux précédents ont été développés avec l'algèbre tensorielle classique. Dans ce cas, la propriété de commutativité de facteurs normaux permet d'utiliser la même méthode de multiplication pour la multiplication à gauche ou à droite.

Rappelons que la multiplication d'un descripteur P par un vecteur w peut être décrite par :

[3] L'algorithme 3.7 présente les modifications nécessaires à l'évaluation des éléments fonctionnels sur un multiplication à gauche. Cependant, ces modifications sont tout à fait pareil lorsqu'on fait la multiplication à droite (présenté dans la section 3.3.3)

ALG. 3.7 Multiplication $v \times \bigotimes_{g_{i=1}}^{N} P^{(i)}(\ldots)$

for $i = \sigma(1), \sigma(2), \ldots, \sigma(N)$ **do**
  $base = 0$;
  initialize $k_1 = 1, k_2 = 1, \ldots, k_{i-1} = 1$;
  for $k = 1, 2, \ldots, nleft_i$ **do**
    for $j = 1, 2, \ldots, nright_i$ **do**
      initialize $j_{i+1} = 1, j_{i+2} = 1, \ldots, j_N = 1$;
      $index = base + j$;
      for $l = 1, 2, \ldots, n_i$ **do**
        $z_{in}[l] = v[index]$;
        $index = index + nright_i$;
      end for
      evaluate $P^{(i)}(a^{(1)}_{k_1}, \ldots, a^{(N)}_{j_N})$;
      multiply $z_{out} = z_{in} \times P^{(i)}$;
      $index = base + j$;
      for $l = 1, 2, \ldots, n_i$ **do**
        $v[index] = z_{out}[l]$;
        $index = index + nright_i$;
      end for
      next $j_{i+1}, j_{i+2}, \ldots, j_N$;
    end for
    $base = base + (nright_i \times n_i)$
    next $k_1, k_2, \ldots, k_{i-1}$;
  end for
end for

$$Pw = \sum_{j=1}^{(N+2E)} \left[\bigotimes_{i=1}^{N}{}_{g} P_j^{(i)} \right] w = \sum_{j=1}^{(N+2E)} \left[\bigotimes_{i=1}^{N}{}_{g} P_j^{(i)} w \right]$$

De la même manière que pour la multiplication à gauche, l'opération qui nous intéresse est la multiplication d'un terme produit tensoriel généralisé par un vecteur. C'est-à-dire :

$$\bigotimes_{i=1}^{N}{}_{g} P^{(i)} w$$

Rappelons qu'il faut décomposer le terme produit tensoriel généralisé selon les règles de décomposition de facteurs normaux.

Une fois l'*ordre de décomposition* des facteurs normaux établi, on peut effectuer la multiplication de chaque facteur normal par le vecteur w. Toutefois, il faut remarquer que l'ordre de

multiplication des facteurs normaux doit être inversé : on doit multiplier d'abord le dernier facteur normal de l'*ordre de décomposition*. Ceci est plus visible lorsqu'on voit la multiplication comme :

$$
\begin{aligned}
(P^{(1)}(\ldots) \otimes P^{(2)}(\ldots) \otimes \ldots \otimes P^{(N-1)}(\ldots) \otimes P^{(N)}(\ldots))w = \\
&\left[I_{nleft_{\sigma_1}} \underset{g}{\otimes} P^{(\sigma_1)}(\ldots) \underset{g}{\otimes} I_{nright_{\sigma_1}} \right] \\
&\times \left[I_{nleft_{\sigma_2}} \underset{g}{\otimes} P^{(\sigma_2)}(\ldots) \underset{g}{\otimes} I_{nright_{\sigma_2}} \right] \\
&\times \cdots \\
&\times \left[I_{nleft_{\sigma_{N-1}}} \underset{g}{\otimes} P^{(\sigma_{N-1})}(\ldots) \underset{g}{\otimes} I_{nright_{\sigma_{N-1}}} \right] \\
&\times \left[I_{nleft_{\sigma_N}} \underset{g}{\otimes} P^{(\sigma_N)}(\ldots) \underset{g}{\otimes} I_{nright_{\sigma_N}} \right] w
\end{aligned}
$$

L'évaluation des éléments fonctionnels est toujours faite de la même façon (Section 3.3.2).

Il faut juste se rappeler que dans la multiplication à droite on multiplie chaque ligne de la matrice $P^{(i)}$ par le vecteur z_{in}. L'algorithme 3.8 présente les modifications nécessaires à la multiplication à droite.

3.4 Détection du régime stationnaire

Lorsqu'on fait de l'analyse transitoire, on peut avoir des valeurs de t et λ[4] très grandes et le nombre d'itérations nécessaires pour calculer la distribution transitoire à l'instant de temps t devient aussi très grand. Toutefois, il est possible que la distribution transitoire à l'instant de temps t soit déjà au régime stationnaire.

Considérons R le nombre de multiplications vecteur-matrice nécessaires pour atteindre le régime stationnaire et K_ε le nombre de multiplications nécessaires pour calculer la distribution transitoire à l'instant de temps t avec une erreur ε. Si R est plus petit que K_ε on peut alors économiser des temps de calcul en arrêtant les itérations à R itérations. Alors, on utilise le vecteur de la distribution stationnaire (obtenu à la R-ème itération) à la place des vecteurs qui suivent la R-ème itération pour calculer la distribution transitoire à l'instant de temps t.

Cependant, le régime stationnaire du système n'est pas toujours simple à détecter de façon performante. Selon Stewart dans [92], le nombre d'itérations requis pour satisfaire une erreur maximum ε peut être calculé par :

$$R \approx \frac{log_2\, \varepsilon}{log_2\, |\lambda_2|}$$

[4]On rappelle que λ est la plus grande valeur des éléments diagonaux du générateur infinitésimal Q.

ALG. 3.8 Multiplication $\bigotimes_{g_{i=1}}^{N} P^{(i)}(\ldots) \times w$

for $i = \sigma(N), \sigma(N-1), \ldots, \sigma(1)$ **do**
  $base = 0$;
  initialize $k_1 = 1, k_2 = 1, \ldots, k_{i-1} = 1$;
  for $k = 1, 2, \ldots,$ $nleft_i$ **do**
    for $j = 1, 2, \ldots,$ $nright_i$ **do**
      initialize $j_{i+1} = 1, j_{i+2} = 1, \ldots, j_N = 1$;
      $index = base + j$;
      for $l = 1, 2, \ldots, n_i$ **do**
        $z_{in}[l] = w[index]$;
        $index = index +$ $nright_i$;
      end for
      evaluate $P^{(i)}(a_{k_1}^{(1)}, \ldots, a_{j_N}^{(N)})$;
      multiply $z_{out} = P^{(i)} \times z_{in}$;
      $index = base + j$;
      for $l = 1, 2, \ldots, n_i$ **do**
        $w[index] = z_{out}[l]$;
        $index = index +$ $nright_i$;
      end for
      next $j_{i+1}, j_{i+2}, \ldots, j_N$;
    end for
    $base = base + (\, nright_i \times n_i)$
    next $k_1, k_2, \ldots, k_{i-1}$;
  end for
end for

où λ_2 est la valeur propre de la sous-dominante de la matrice P. Malheureusement, la valeur de λ_2 est difficile à calculer et fréquemment on observe que la différence entre les conditions théoriques et pratiques pour la convergence d'une méthode itérative peut être très grande lorsque le module de la valeur propre de la sous-dominante λ_2 est proche de l'unité de base[5]. Dans ces cas, on a une convergence qui s'annonce très lente lorsqu'on utilise la méthode de la puissance [92].

Dans la pratique, d'autres méthodes de détection du régime stationnaire sont utilisées. Plusieurs méthodes de détection sont proposées dans la littérature. Les plus connues sont les méthodes qui étudient la convergence du vecteur de probabilité, parmi lesquelles, on souligne les travaux de Ciardo et al. dans [20]. Plusieurs améliorations à l'algorithme de Ciardo ont été proposées au cours des 10 dernières années, mais la plus grande partie se limitait à des modèles spécifiques, comme par exemple les travaux de Katoen et Zapreev [61]. En 1999, Sericola a proposé une nouvelle méthode basée sur l'étude de la suite des valeurs maximum et minimum d'un vecteur [91].

[5] L'unité de base est la plus petite valeur présente dans une matrice.

Dans cette section on va décrire deux méthodes de détection du régime stationnaire. L'efficacité de ces méthodes sera testé sur des modèles avec grand espace d'états. Une comparaison de l'efficacité de ces méthodes est présenté dans le chapitre 5.

3.4.1 Convergence du vecteur de probabilité

La première méthode de détection du régime stationnaire a été proposée par Ciardo *et al.* dans [20]. Cette méthode est basée sur la convergence du vecteur de probabilité.

Rappelons l'équation d'élévation de la matrice P à la puissance n par la multiplication à gauche (Équation (3.9)) réécrit ci-dessous.

$$v_n = \begin{cases} v_0 = \pi(0) \\ v_n = v_{n-1}P \end{cases}$$

La méthode définie par Ciardo consiste à détecter le régime stationnaire par la convergence du vecteur de probabilité. Plusieurs approches sont possibles pour établir la convergence du vecteur de probabilité [92]. Les approches les plus souvent utilisées sont, en prenant la norme du maximum :

– *Test de convergence absolue individuel* :

$$max_{i\in 1,..,S}\ (|v_n[i] - v_{n-1}[i]|) < \varepsilon$$

– *Test de convergence absolue accumulée* :

$$\| v_n[i] - v_{n-1}[i] \|_1 < \varepsilon$$

– *Test de convergence relative individuel* :

$$max_{i\in 1,..,S}\ \left(\frac{|v_n[i] - v_{n-1}[i]|}{|v_n[i]|}\right) < \varepsilon$$

Selon Stewart [92], le test de convergence relative individuel est recommandé pour les modèles avec grand espace d'états, car le test de convergence absolue individuel n'est pas très fiable lorsque toutes les valeurs du vecteur de probabilité sont petites. Si l'erreur admissible est plus grande que les valeurs du vecteur de probabilité, le test de convergence va détecter une fausse convergence du modèle. Cependant, lorsqu'on utilise la multiplication à droite le test de convergence relative individuel n'est plus utilisable car la valeur de la disponibilité ponctuelle peut être très proche de 0.

On suppose R_ε le nombre d'itérations nécessaires pour atteindre le régime stationnaire selon le test de convergence choisi avec un erreur ε. On utilise alors le vecteur de probabilité v_{R_ε} comme une approximation à ε près du vecteur de probabilité de la distribution stationnaire π.

Une fois le régime stationnaire détecté, on a deux cas possibles :

Soit $R_\varepsilon > K_\varepsilon$ et le régime stationnaire n'a pas été détecté selon les critères établis. Autrement dit, la méthode de détection du régime stationnaire n'a pas d'effet sur le calcul de la disponibilité ponctuelle et on garde l'équation (3.15) définie dans la section 3.2.1 (page 34) ;

Soit $R_\varepsilon \leq K_\varepsilon$ et le test de convergence a détecté le régime stationnaire à la R_ε-ème itération. On peut donc utiliser le vecteur v_{R_ε} comme une approximation du vecteur stationnaire π pour tout $n \geq R_\varepsilon$ et réécrire l'équation (3.15) par :

$$\mathrm{PAV}_\varepsilon(t) = \sum_{n=0}^{R_\varepsilon} e^{-\lambda t}\frac{(\lambda t)^n}{n!} v_n \mathbb{1}_{UP} + \sum_{n=R_\varepsilon+1}^{K_\varepsilon} e^{-\lambda t}\frac{(\lambda t)^n}{n!} v_{R_\varepsilon} \mathbb{1}_{UP} \tag{3.21}$$

Détection sur la multiplication à droite

La méthode de détection du régime stationnaire sur la convergence des vecteurs de probabilité peut être aussi employée avec la méthode de calcul de la disponibilité ponctuelle lorsque la élévation de la matrice P à la puissance K est faite par la multiplication à droite. Dans ce cas, la comparaison des éléments de chaque vecteur est faite sur les vecteurs w_n et w_{n-1} (Équation (3.12), page 33).

Le test de convergence sur les vecteurs de la multiplication à droite se déroule de la même manière, par comparaison de la différence entre la norme de deux vecteurs.

Lorsque la convergence est détectée, on a les mêmes possibilités de calcul, soit on a $R_\varepsilon > K_\varepsilon$ et on garde l'équation classique (Équation (3.15), page 34), soit on a $R_\varepsilon \leq K_\varepsilon$ et on divise la somme en deux parties : la première partie calcule la suite de 0 à R_ε et la deuxième partie calcule la suite de $R_\varepsilon + 1$ à K_ε. En réécrivant l'équation (3.15) pour la multiplication à droite on obtient :

$$\mathrm{PAV}_\varepsilon(t) = \sum_{n=1}^{R_\varepsilon} e^{-\lambda t}\frac{(\lambda t)^n}{n!} \pi(0) w_n + \sum_{n=R_\varepsilon+1}^{K_\varepsilon} e^{-\lambda t}\frac{(\lambda t)^n}{n!} \pi(0) w_{R_\varepsilon} \tag{3.22}$$

3.4.2 Contrôle de la suite de w_n

Une deuxième méthode de détection du régime stationnaire a été proposée par Sericola dans [91] et est basée sur le contrôle de la suite des vecteurs $w_n = P^n \mathbb{1}_{UP}$.

Sericola définit deux suites, une suite des valeurs minimums m_n et une autre des valeurs maximums M_n. Chaque suite est définie sur les valeurs des vecteurs w_n pour tout $n > 0$ par :

$$m_n = \min_{i \in S} w_n[i] \quad \text{et} \quad M_n = \max_{i \in S} w_n[i].$$

ALG. 3.9 Algorithme pour calculer $\mathrm{PAV}_\varepsilon(t)$ avec détection du régime stationnaire.

Input : $0 < t_1 < \cdots < t_\ell$, ε
Output : $\mathrm{PAV}_\varepsilon(t_1), \ldots, \mathrm{PAV}_\varepsilon(t_\ell)$
Compute K_ε par Éq. (3.14)
$w_0 = \mathbb{1}_U$
$W_0 = \pi(0)w_0$
$n = 1$
while $R \leq K_\varepsilon$ && $!conv$ **do**
  $w_n = Pw_{n-1}$
  $W_n = \pi(0)w_n$
  $n = n + 1$
  $conv = convergence\_test$
end while
for j=1 to ℓ **do**

$$\mathrm{PAV}_\varepsilon(t_j) = \sum_{n=0}^{R} e^{-\lambda t}\frac{(\lambda t)^n}{n!}W_n + \sum_{n=R+1}^{K_\varepsilon} e^{-\lambda t}\frac{(\lambda t)^n}{n!}W_R$$

end for

Dans [91], il est montré que les suites m_n et M_n sont, respectivement croissantes et décroissantes pour tout $n > 0$ et convergent toutes les deux vers $\pi\mathbb{1}_{UP}$ quand $n \longrightarrow \infty$. Autrement dit, les suites convergent vers la disponibilité en régime stationnaire.

Alors, il existe une valeur entière R_ε telle que,

$$R_\varepsilon = min\ (n \geq 0 \mid M_n - m_n \leq \varepsilon/2)$$

La valeur R_ε représente le nombre d'itérations à partir duquel la suite W_n[6] est une approximation à ε près de la vraie valeur W_n dans le régime stationnaire. Ainsi, on peut réécrire la disponibilité ponctuelle (Équation (3.8)) par :

$$\mathrm{PAV}_\varepsilon(t) = \sum_{n=0}^{R_\varepsilon} e^{-\lambda t}\frac{(\lambda t)^n}{n!}W_n + \left(1 - \sum_{n=0}^{R_\varepsilon} e^{-\lambda t}\frac{(\lambda t)^n}{n!}\right)\frac{M_{R_\varepsilon} + m_{R_\varepsilon}}{2} \tag{3.23}$$

Puisque les suites m_n et M_n convergent vers la disponibilité en régime stationnaire, on peut calculer l'instant de temps t à partir duquel on atteint le régime stationnaire. Alors, on définit une fonction $F_l(t)$ telle que :

$$F_l(t) = \sum_{n=0}^{l} e^{-\lambda t}\frac{(\lambda t)^n}{n!}(M_n - m_n). \tag{3.24}$$

[6] W_n est le résultat de la multiplication du vecteur de probabilité initiale par le vecteur w_n (voir Algorithme 3.2, page 35).

On peut facilement vérifier que pour une valeur fixe de l, la fonction $F_l(t)$ décroît de 1 vers 0 dans l'intervalle $[0, \infty[$.

Sericola prouve dans [91] que, en prenant la valeur de l comme N_ε, on peut calculer l'instant de temps t_ε auquel on atteint le régime stationnaire avec N_ε itérations à ε près par :

$$t_\varepsilon = min\ (t \geq 0; F_{N_\varepsilon}(t) \leq \varepsilon/4) \tag{3.25}$$

On peut alors réécrire l'équation de la disponibilité ponctuelle (Équation (3.23)) pour toute valeur de $t > t_\varepsilon$ par :

$$PAV_\varepsilon(t) = \frac{M_{N_\varepsilon} + m_{N_\varepsilon}}{2} \tag{3.26}$$

Par la suite, Sericola a aussi présenté dans [91] l'algorithme suivant comme une adaptation de l'algorithme classique (Algorithme 3.2, page 35) en prenant en compte le calcul de la disponibilité ponctuelle par les équations (3.23) et (3.26) et la détection du régime stationnaire.

Dans la section suivante, on présente 3 possibilités de représentation du vecteur d'état initial et du vecteur d'états UP lorsqu'ils restent constants sur les algorithmes de calcul de la disponibilité ponctuelle.

3.5 Représentation du vecteur constant

En considérant l'équation de la disponibilité ponctuelle (Équation (3.8), page 32) et les algorithmes 3.1 (page 34) et 3.2 (page 35), on peut voir que dans la boucle (lignes 7), il y a toujours un vecteur constant : lorsqu'on calcule la disponibilité ponctuelle par une méthode de multiplication à gauche, on a le vecteur d'états UP qui reste constant. Pour une méthode basée sur la multiplication à droite, c'est le vecteur de la distribution initiale qui reste constant.

De façon à profiter de cette caractéristique, on propose 3 formats de représentation de ces vecteurs. Deux formats de représentation du vecteur sont des approches classiques basées sur une représentation par tableau et le troisième format représente les ensembles sous forme d'une fonction.

- Vecteur plein : représente l'ensemble d'états initiaux ou l'ensemble d'états UP par un tableau à 1 dimension standard. Ce tableau a la taille de l'espace d'états produit du modèle. Cette approche peut être très coûteuse en espace de stockage pour de grands modèles ;
- Vecteur creux : ce format de représentation utilise aussi des tableaux pour représenter les ensembles d'états initiaux et des états UP. Par contre, uniquement les états qui ont une valeur de probabilité différente de zéro sont stockés. Ce format est très économique lorsque le nombre d'états initiaux (ou d'états UP) est très petit par rapport au nombre total d'états (espace d'états produit).

ALG. 3.10 Algorithme de Sericola pour calculer $\mathrm{PAV}_\varepsilon(t)$

Input : $0 < t_1 < \cdots < t_\ell$, ε
Output : $\mathrm{PAV}_\varepsilon(t_1), \ldots, \mathrm{PAV}_\varepsilon(t_\ell)$
Compute K_ε par Éq. (3.14)
$w_0 = \mathbb{1}_U$
$W_0 = \pi(0)w_0$
$M_0 = 1$; $m_0 = 0$
$n = 1$
while $((n \leq K_\varepsilon))$ && $((M_n - m_n) > \varepsilon/2)$ **do**
  $w_n = Pw_{n-1}$
  $W_n = \pi(0)w_n$
  $M_n = \max_{i \in S} w_n(i)$; $m_n = \min_{i \in S} w_n(i)$
  $n = n + 1$
end while
$R = n$
if $(R == K_\varepsilon + 1)$ **then**
  for $j = 1$ to ℓ **do**
    $\mathrm{PAV}(t_j) = \sum_{n=0}^{K_\varepsilon} e^{-\lambda t_j} \frac{(\lambda t_j)^n}{n!} W_n$
  end for
if $(R \leq K_\varepsilon + 1)$ **then**
  Compute $T_R = \inf\{t \geq 0,\ F_R(t) \leq \varepsilon/4\}$
  for $j = 1$ to ℓ do **do**
    if $t_j \leq T_R$ **then**
      $\mathrm{PAV}(t_j) = \sum_{n=0}^{R} e^{-\lambda t_j} \frac{(\lambda t_j)^n}{n!} W_n + \frac{M_R + m_R}{2} \left(1 - \sum_{n=0}^{R} e^{-\lambda t_j} \frac{(\lambda t_j)^n}{n!}\right)$
    else
      $\mathrm{PAV}(t_j) = \frac{M_R + m_R}{2}$
  end for

– Fonction : la représentation par fonction profite du formalisme SAN pour représenter les états initiaux ou les états UP. Cette fonction est alors évaluée pour chaque état de l'espace d'états produit à chaque itération, ce qui peut ralentir l'algorithme. Cependant, c'est une méthode très économique en mémoire.

Une comparaison de ces trois représentation par rapport à la vitesse de calcul et par rapport à l'espace de stockage est présenté dans le chapitre 5 section 5.4.

3.6 Conclusion

Dans ce chapitre, on a présenté, tout d'abord, les méthodes de calcul de la disponibilité ponctuelle d'un modèle. Pour permettre l'application de ces méthodes sur des grands modèles, on a proposé une adaptation de ces méthodes à la représentation tensorielle. Cette adaptation a entraîné l'adaptation de l'algorithme *shuffle* pour la multiplication à droite.

Ensuite, on a présenté les méthodes de détection du régime stationnaire qui seront testées sur de grands modèles. La comparaison de l'efficacité de ces méthodes est présentée dans le chapitre 5.

Un dernier point étudié dans ce chapitre a été l'utilisation de différentes implantations pour le vecteur constant. Une comparaison de ces trois représentations est également présentée dans le chapitre 5.

Chapitre 4

Exemples de modélisation de systèmes sujets aux défaillances

Dans ce chapitre on présente quelques exemples de systèmes sujets aux défaillances. Ces exemples sont utilisés pour tester l'efficacité de méthodes de détection du régime stationnaire sur le calcul de la disponibilité ponctuelle pour des grands modèles (Chapitre 3).

Ce chapitre présente 6 exemples de systèmes sujets aux défaillances rencontrés dans la littérature. Ces exemples illustrent différents types de pannes et de réparations et ont pour but de présenter plusieurs type d'interactions entre les composantes. Les exemples développés dans ce chapitre sont :

- Discipline de premier serveur disponible (Section 4.1) ;
- Système multiprocesseurs (Section 4.2) ;
- Grappe de processeurs reconfigurable (Section 4.3) ;
- Base de données distribuée (Section 4.4) ;
- Patron de mobilité aléatoire (Section 4.5) ;
- Grappe de stations de travail (Section 4.6).

Ces exemples sont modélisés à l'aide du formalisme des Réseaux d'Automates Stochastiques (SAN) présenté dans le chapitre 2. On rappelle que le formalisme SAN modélise un système par la composition de sous-système qui interagent occasionnellement. Ce formalisme est adapté à la modélisation de grands modèles et a une représentation tensorielle basée sur l'algèbre tensorielle généralisée.

4.1 Discipline du premier serveur disponible - FAS (*First Available Server*)

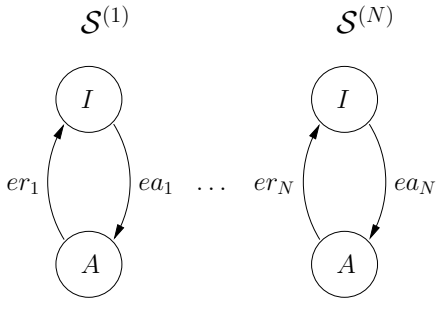
Cet exemple décrit une file simple avec arrivée exponentielle et un nombre fini (N) de serveurs ordonnés et distinguables ($\mathcal{S}^{(i)}$, où $i \in [1..N]$) [14].

Lorsqu'un client arrive, il est tout de suite servi par le premier serveur disponible (*First Available Server*), *i.e.*, si le serveur $\mathcal{S}^{(1)}$ est disponible, le client est servi par celui-ci, sinon, si le serveur $\mathcal{S}^{(2)}$ est disponible, le client est servi par le serveur $\mathcal{S}^{(2)}$ et ainsi de suite. Le taux d'arrivée des clients dans la file est λ et le taux de service de chaque client est μ.

4.1.1 Modèle SAN

La technique utilisée pour modéliser une file avec ces caractéristiques a été de représenter chaque serveur i par un automate $\mathcal{S}^{(i)}$. Chaque automate a 2 états (I pour *Inactif* et A pour *Actif*). FIG. 4.1 présente le modèle SAN de cet exemple.

Le taux d'arrivée dans chaque serveur est exprimé par un événement local ea_i avec un taux fonctionnel qui vaut λ si tout les serveurs précédents sont occupés (tous dans l'état A) et vaut 0 s'il y a un serveur précédent qui est disponible. On assure avec cette fonction que, à un moment donné, un seul serveur (le premier disponible) a un taux d'arrivée diffèrent de 0. La fin de service dans chaque serveur est exprimée par un simple événement local er_i avec un taux constant μ.



Type	Événement	Taux	Type	Événement	Taux
loc	ea_1	λ	loc	er_1	μ
loc	ea_2	$\lambda.f_2$	loc	er_2	μ
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
loc	ea_N	$\lambda.f_N$	loc	er_N	μ

$$f_i = (nb\ [\mathcal{S}^{(1)}\ ..\ \mathcal{S}^{(i-1)}]\ A\ ==\ (i-1))$$

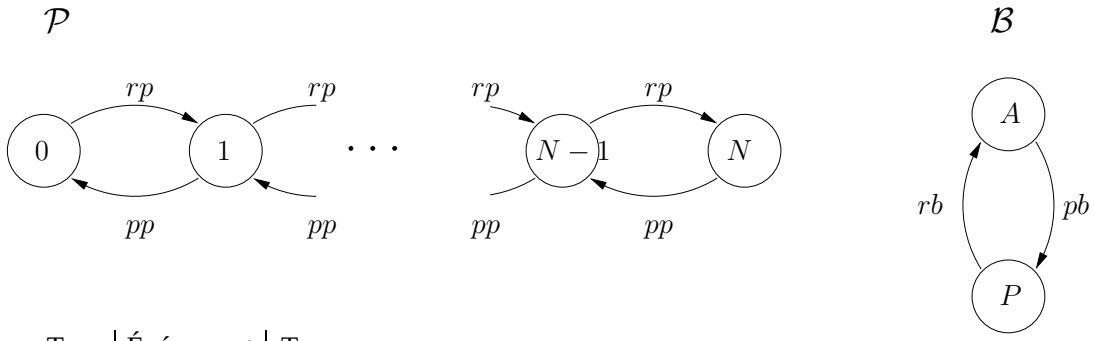
FIG. 4.1 – Modèle SAN de l'exemple FAS

4.2 Système multiprocesseurs - MSA (*Multiprocessors System Availability*)

Le modèle MSA décrit un système multiprocesseurs tolérant aux défaillances et une mémoire tampon de tâches [64, 91]. Ce modèle représente un groupe de N processeurs identiques et une mémoire tampon simple. Chaque processeur tombe en panne avec un taux λ et est réparé avec un taux μ. La mémoire tampon peut être dans l'état de marche ou défaillante. La mémoire tampon a un taux de défaillance γ et un taux de réparation de τ. Les taux de défaillance des processeurs et de la mémoire tampon sont indépendantes, cependant, lorsque la mémoire tampon est dans l'état de défaillance, aucune autre panne des processeurs n'est admise.

4.2.1 Modèle SAN

Le modèle SAN pour cet exemple est composé de 2 automates (FIG. 4.2). Le premier automate $\mathcal{P}$ décrit le groupe de N processeurs. On a alors un automate avec $N+1$ états. Chaque état représente le nombre de processeurs disponibles. On utilise 2 événements locaux pour représenter la défaillance et la réparation des processeurs. L'événement pp exprime la défaillance d'un processeur et a un taux λ. À ce taux on associe une fonction f qui indique le nombre de processeurs qui peuvent tomber en panne. On associe également au taux de déclenchement de l'événement pp, un e fonction g qui indique l'état de la mémoire tampon. Cette fonction vaut 1 si la mémoire tampon est en état de marche et 0 si elle en panne. L'événement local rp a un taux constant μ et représente la réparation d'un processeur. Un automate $\mathcal{B}$ à 2 états est utilisé pour décrire l'état de la mémoire tampon. Les 2 événement locaux utilisés pour décrire la défaillance de la mémoire tampon (évènement pb) et la réparation de celle-ci (événement rb) ont, respectivement, des taux constant égaux à γ et τ.



Type	Événement	Taux
loc	pp	$\lambda.f.g$
loc	rp	μ
loc	pb	δ
loc	rb	γ

$f = (st\ \mathcal{P})$

$g = (st\ \mathcal{B}\ ==\ A)$

FIG. 4.2 – Modèle SAN de l'exemple MSA

4.3 Grappe de processeurs reconfigurable - CFR (*Cluster Failure Reconfiguration*)

Nous traitons dans cet exemple une vision simple du comportement d'un logiciel de distribution de tâches dans une grappe de processeurs [89, 90]. Considérons une grappe avec N noeuds où chaque noeud à une connexion directe avec 2 noeuds voisins, c'est-à-dire que cette grappe a une connexion logique en forme d'anneau.

Le système a un certain nombre de tâches. Ces tâches sont réparties parmi les noeuds actifs du système. Chaque noeud exécute un certain nombre des tâches. Un noeud peut alterner entre 3 états : *actif*, *panne* ou *reconfiguration*. Dans l'état *actif* le noeud fonctionne correctement. L'état *panne* représente une phase de réparation logicielle (locale, qui consiste à le redémarrer), après avoir mal fonctionné. Lorsque le noeud i entre dans l'état de *panne*, ses noeuds voisins doivent entrer dans un état de *reconfiguration* de la grappe pour prendre en charge le travail du processeur i défaillant. Lorsque le noeud i est réparé, il entre avec ses voisins dans la phase de *reconfiguration* pour qu'il puisse reprendre son travail. Après une reconfiguration de la grappe, chaque noeud revient à son état *actif*. La reconfiguration d'un noeud est donc locale à ce noeud.

4.3.1 Modèle SAN

Le modèle SAN proposé pour cet exemple traite chaque noeud par un automate. Chaque automate $\mathcal{N}^{(i)}$ a 3 états : A, P et R, qui représentent, respectivement, les états *actif*, *panne* et *reconfiguration* de chaque noeud. L'automate associé au noeud i de la grappe est donné par FIG. 4.3.

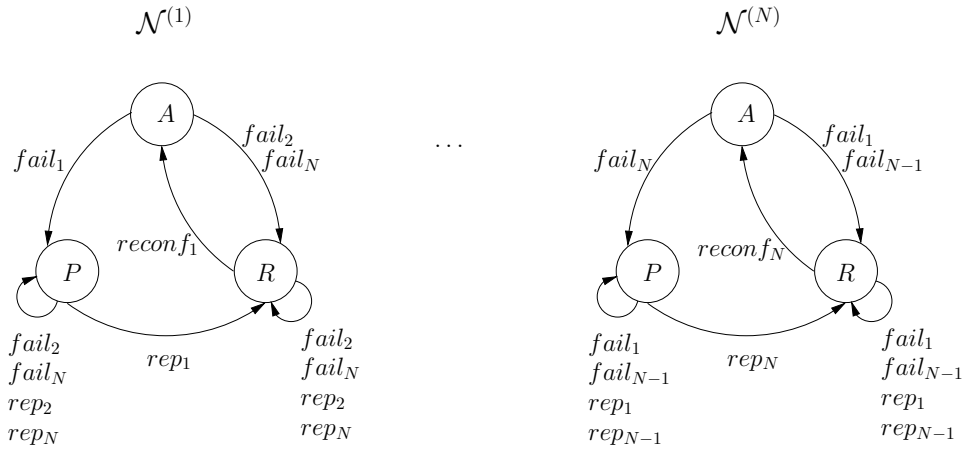
La panne d'un noeud i est modélisé par l'événement synchronisant $fail_i$, auquel on associe un taux λ_i. L'événement rep_i représente la réparation du noeud i avec un taux μ_i. L'étape de reconfiguration du noeud i est associée à l'événement local $reconf_i$ avec un taux τ_i.

On associe également, à chaque automate $\mathcal{N}^{(i)}$, les événements de panne et réparation ($fail_{i-1}$, $fail_{i+1}$, rep_{i-1} et rep_{i+1}) associé aux noeuds voisins.

4.4 Base de données distribuée - DDS (*Distributed Database System*)

Cet exemple est basé sur un problème classique présenté par Muntz, Souza e Silva, et Goyal dans [66] et repris par Carrasco dans [17].

L'exemple décrit un système de base de données distribuée (*Distributed Database System* - DDS). Le système est composé de deux ensembles de processeurs connectés à des contrôleurs qui sont connectés à des ensembles de disques, comme présenté dans FIG. 4.4.



Type	Événement	Taux	Type	Événement	Taux	Type	Événement	Taux
syn	$fail_1$	λ_1	syn	rep_1	μ_1	loc	$reconf_1$	τ_1
syn	$fail_2$	λ_2	syn	rep_2	μ_2	loc	$reconf_2$	τ_2
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
syn	$fail_{N-1}$	λ_{N-1}	syn	rep_{N-1}	μ_{N-1}	loc	$reconf_{N-1}$	τ_{N-1}
syn	$fail_N$	λ_N	syn	rep_N	μ_N	loc	$reconf_N$	τ_N

FIG. 4.3 – Modèle SAN de l'exemple CFR

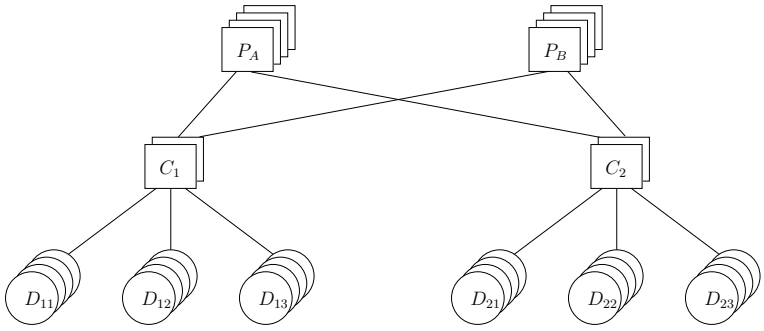


FIG. 4.4 – Architecture d'une base de données distribuée

Le système est composé de deux types de processeurs P_A et P_B. On a 4 processeurs de chaque type. On considère que un seul processeur de chaque type est actif et que les processeurs inactifs ne sont pas sujet aux défaillance. Les processeurs du type P_A tombent en panne avec un taux λ_a. Lorsqu'un processeur du type P_A tombe en panne, cette défaillance est peut être propagée au processeur du type P_B avec une probabilité de $0,1$. Les processeurs du type P_B ont un taux de défaillance de λ_b. Une défaillance d'un processeur du type P_B n'est jamais propagée au processeur du type P_A.

Chaque ensemble de processeurs est connecté à deux ensembles de contrôleurs. Les contrôleurs sont du type C_1 ou C_2. Chaque ensemble de contrôleur est composé de 2 contrôleurs. Les contrôleurs du type C_1 tombent en panne avec un taux γ_1 et les contrôleurs du type C_2 avec

un taux γ_2. Chaque ensemble de contrôleurs est connecté à 3 ensembles de disques. Chaque ensemble de disque est composé de 4 disques. Les données *primaires* de chaque disque sont répliquées sur les autres disques de l'ensemble de façon à ce que chaque disque ait $1/3$ des donnés. Les ensembles de disques D_{11}, D_{12} et D_{13}, connectés aux contrôleurs C_1, tombent en panne avec les taux τ_{11}, τ_{12} et τ_{13}, respectivement. Les ensembles de disques D_{21}, D_{22} et D_{23}, connectés aux contrôleurs C_2, tombent en panne, respectivement, avec les taux τ_{21}, τ_{22} et τ_{23}.

Toutes les réparations ont les même taux μ, pour n'importe le type de composants. Cependant, le taux de réparation est multiplié par 10 lorsque le système est dans un état $DOWN$.

4.4.1 Modèle SAN

Le modèle SAN pour cet exemple est composé de 10 automates. Chaque automate représente un type de composants (10 types de composants différents sont présents dans le système).

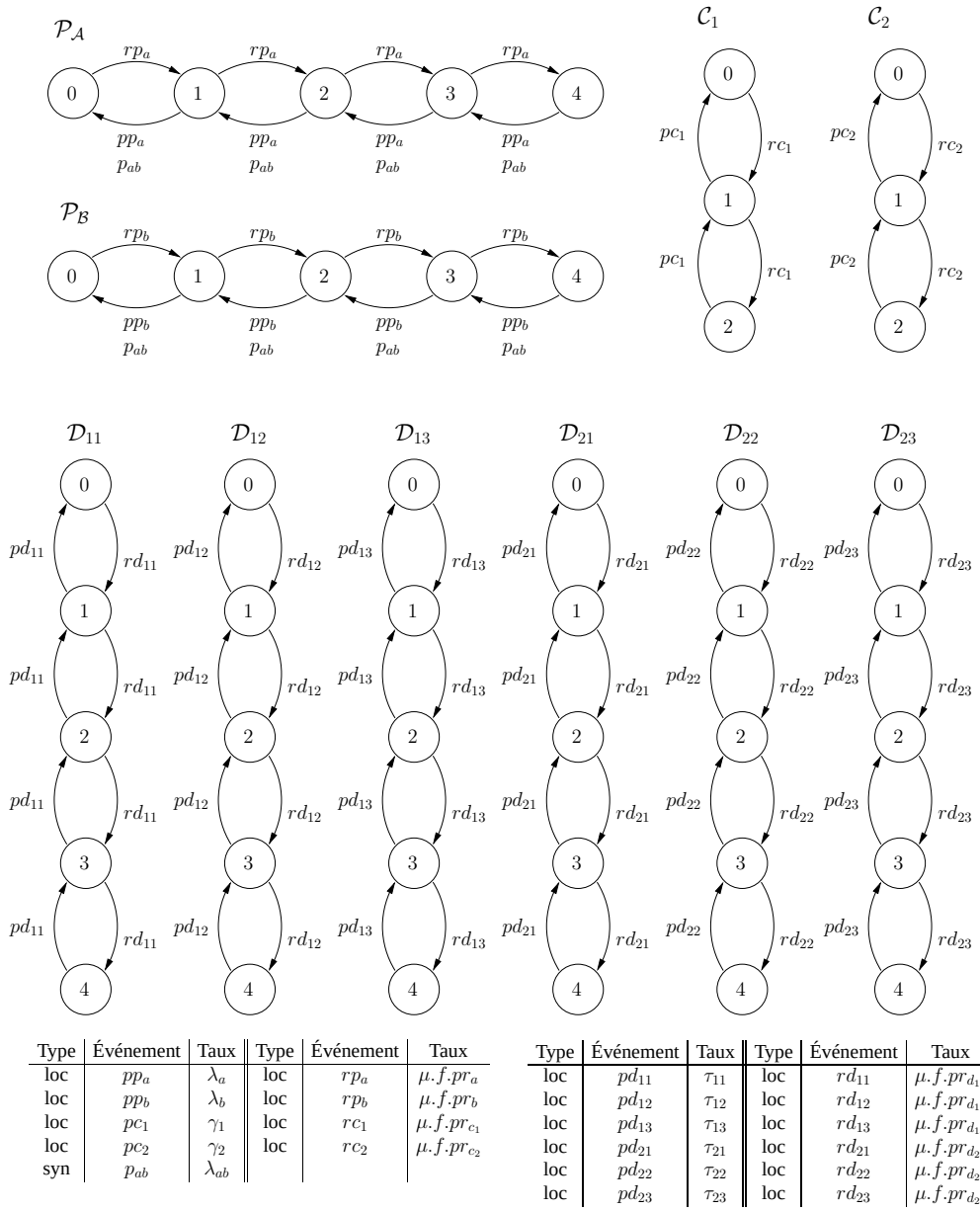
Dans FIG. 4.5, les deux premiers automates $\mathcal{P}_\mathcal{A}$ et $\mathcal{P}_\mathcal{B}$, représentent, respectivement, les processeurs du P_A et P_B. Les événements pp_a et pp_b représentent, respectivement, la défaillance d'un processeur du type P_A et P_B. L'événement p_{ab} représente une défaillance d'un processeur du type P_A propagée à un processeur du type P_B. Les réparations sont représentées par les événements rp_a et rp_b.

Les automates $\mathcal{C}_1$ et $\mathcal{C}_2$ représentent les contrôleurs du type C_1 et C_2, respectivement. Les défaillances de contrôleurs sont représentées par les événements pc_1 et pc_2, tandis que les réparations sont représentées par les événements rc_1 et rc_2.

Les 6 derniers automates ($\mathcal{D}_{11}$, $\mathcal{D}_{12}$, $\mathcal{D}_{13}$, $\mathcal{D}_{21}$, $\mathcal{D}_{22}$ et $\mathcal{D}_{23}$) représentent les ensembles de disques. Les événements pd_{ij} et rd_{ij} représentent les défaillances et les réparations d'un disque sur l'ensemble de disques D_{ij}.

On remarque qu'on associe aux taux des événements de réparations plusieurs fonctions. Ces fonctions sont détaillées ci-dessous :

- f la fonction f vaut 1 si le système est UP et 10 si le système est $DOWN$. C'est-à-dire que les taux de réparation sont multipliés par 10 lorsque le système est dans un état $DOWN$;
- pr_i cette fonction décrit la probabilité de réparation d'un composant lorsqu'il y a plusieurs composants en panne. La fonction est le résultat du nombre de composants du type i en panne divisé par le nombre total de composants en panne dans le système.



Type	Événement	Taux	Type	Événement	Taux
loc	pp_a	λ_a	loc	rp_a	$\mu.f.pr_a$
loc	pp_b	λ_b	loc	rp_b	$\mu.f.pr_b$
loc	pc_1	γ_1	loc	rc_1	$\mu.f.pr_{c_1}$
loc	pc_2	γ_2	loc	rc_2	$\mu.f.pr_{c_2}$
syn	p_{ab}	λ_{ab}			

Type	Événement	Taux	Type	Événement	Taux
loc	pd_{11}	τ_{11}	loc	rd_{11}	$\mu.f.pr_{d_{11}}$
loc	pd_{12}	τ_{12}	loc	rd_{12}	$\mu.f.pr_{d_{12}}$
loc	pd_{13}	τ_{13}	loc	rd_{13}	$\mu.f.pr_{d_{13}}$
loc	pd_{21}	τ_{21}	loc	rd_{21}	$\mu.f.pr_{d_{21}}$
loc	pd_{22}	τ_{22}	loc	rd_{22}	$\mu.f.pr_{d_{22}}$
loc	pd_{23}	τ_{23}	loc	rd_{23}	$\mu.f.pr_{d_{23}}$

$$f = ((st\ \mathcal{P}_\mathcal{A}\ !=\ 0)\ ||\ (st\ \mathcal{P}_\mathcal{B}\ !=\ 0))\ \&\&\ ((st\ \mathcal{C}_1\ !=\ 0)\ \&\&\ (st\ \mathcal{C}_2\ !=\ 0))\ \&\&\ (((nb[\mathcal{D}_{11}..\mathcal{D}_{23}]4) + (nb[\mathcal{D}_{11}..\mathcal{D}_{23}]3)) == 6)$$

$$pr_i = ((st\ \mathcal{P}_\mathcal{A}\ !=\ 0)\ ||\ (st\ \mathcal{P}_\mathcal{B}\ !=\ 0))\ \&\&\ ((st\ \mathcal{C}_1\ !=\ 0)\ \&\&\ (st\ \mathcal{C}_2\ !=\ 0))$$

FIG. 4.5 – Modèle SAN de l'exemple DDS

4.5 Patron de mobilité aléatoire - RWP (*Random Waypoint Mobility Pattern*)

Cet exemple décrit un patron de mobilité aléatoire par point tournant (*waypoint*). Ce patron de mobilité est largement utilisé dans les réseaux sans fil. Le modèle SAN original proposé par

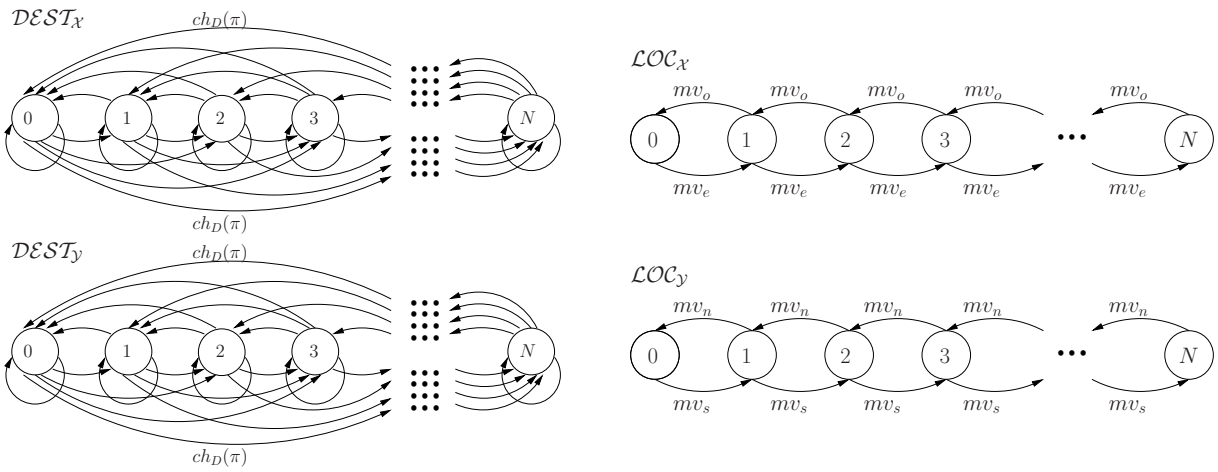
Delamare *et al.* dans [27] est utilisé pour obtenir la distribution spatiale d'un noeud et calculer la densité du réseau.

Dans le patron de mobilité aléatoire par point tournant, le noeud choisit aléatoirement un point de destination et se déplace vers cette destination à vitesse constante. Lorsque la destination est atteinte, le noeud reste dans cette destination quelques instants jusqu'à ce qu'une nouvelle destination soit choisie.

Dans cet exemple, on étudie ce patron de mobilité sur 2 dimensions. Autrement dit, une surface où le noeud a 4 possibilités de mouvement : Nord, Sud, Est et Ouest.

4.5.1 Modèle SAN

Pour représenter ce patron de mobilité, on utilise 4 automates. Ce modèle est présenté dans FIG. 4.6.



Type	Événement	Taux
loc	mv_o	f_1
loc	mv_e	f_2
loc	mv_n	f_3
loc	mv_s	f_4
syn	ch_D	f_5

$f_1 = rtSpeed * (((st\ \mathcal{DEST}_\mathcal{X}) < (st\ \mathcal{LOC}_\mathcal{X}))$
$f_2 = rtSpeed * (((st\ \mathcal{DEST}_\mathcal{X}) > (st\ \mathcal{LOC}_\mathcal{X})))$
$f_3 = rtSpeed * (((st\ \mathcal{DEST}_\mathcal{Y}) > (st\ \mathcal{LOC}_\mathcal{Y})))$
$f_4 = rtSpeed * (((st\ \mathcal{DEST}_\mathcal{Y}) < (st\ \mathcal{LOC}_\mathcal{Y})))$
$f_D = (1/Pause * (((st\ \mathcal{DEST}_\mathcal{X}) == (st\ \mathcal{LOC}_\mathcal{X}))\&\&((st\ \mathcal{DEST}_\mathcal{Y}) == (st\ \mathcal{LOC}_\mathcal{Y})))$
$\pi = 1/N$

FIG. 4.6 – Modèle SAN pour le patron de mobilité aléatoire par point tournant.

Sur ce modèle, les automates $\mathcal{LOC}_\mathcal{X}$ et $\mathcal{LOC}_\mathcal{Y}$ modélisent la position courante du noeud dans les axes x et y. Les événements locaux mv_e, mv_o, mv_s et mv_n représentent, respectivement, le déplacement du noeud dans la direction *est*, *ouest*, *sud* et *nord*. Ces événements ont des taux fonctionnels qui indiquent non seulement la vitesse mais aussi la direction du déplacement.

Dans cet exemple, on adopte une stratégie de déplacement basée sur une décision aléatoire. Cependant, différentes stratégies de déplacement peuvent être modélisées [27]. Cette stratégie de déplacement basée sur une décision aléatoire est aussi modélisée par les fonctions associées aux événements de déplacement mv_e, mv_o, mv_s et mv_n.

Les automates $\mathcal{DEST}_{\mathcal{X}}$ et $\mathcal{DEST}_{\mathcal{Y}}$ modélisent la destination du noeud dans le repère d'axes x et y. Un seul événement synchronisant (ch_D) est utilisé pour le choix de la nouvelle destination. Le taux cet événement est calculé à partir du temps de repos du noeud dans le site de destination. Ce taux est modélisé par la fonction f_D. Dans ce modèle, on fixe une probabilité de routage π équitablement distribuée.

4.6 Grappe de stations de travail - WORKSTATION (*Workstation Cluster*)

Cet exemple présente un système structuré qui s'inspire d'une grappe de stations de travail connue dans les commutateurs de téléphonie mobile [53].

On considère dans cet exemple une grappe de stations de travail tolérante aux pannes, décrite dans FIG. 4.7. Cet exemple a été proposé par Haverkort, Hermanns et Katoen dans [52] pour analyser des problèmes de confiance par de techniques de *model checking*. Le système consiste en C sous-grappe de N stations de travail connectés à un commutateur par une topologie en *étoile*. Les commutateurs assurent la connexion entre chaque grappe par l'*épine dorsale*.

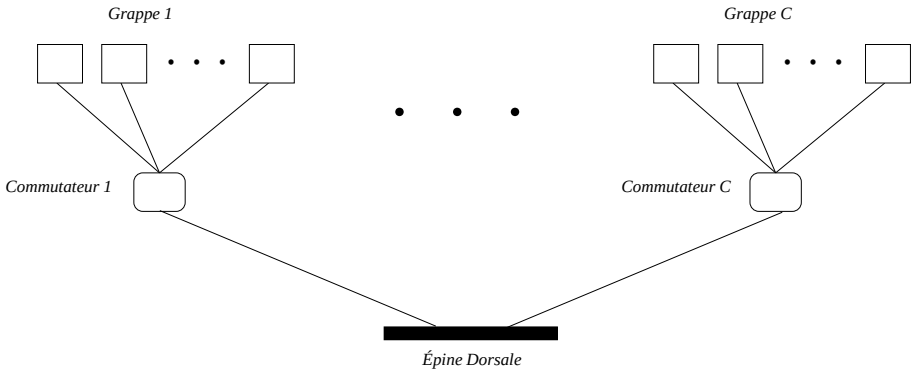


FIG. 4.7 – Structure de la grappe de postes de travail tolérante aux panne.

Dans chaque grappe de stations de travail, tous les composants sont identiques et ont un taux de défaillance λ_i où $i \in [1..C]$. Les commutateurs et l'épine dorsale ont un taux de défaillance de ρ_i où $i \in [1..C]$ pour les commutateurs et γ pour l'épine dorsale.

Le système a une seule unité de réparation qui doit se charger de tous les composants. Aucune politique de priorité de réparation n'est adoptée. L'unité de réparation inspecte le système avec un taux α. Chaque type de composant a un taux de réparation différent. L'épine dorsale est réparée avec un taux δ tandis que les commutateurs sont réparés avec un taux τ_i et les postes de travail avec un taux μ_i où $i \in [1..C]$.

4.6.1 Modèle SAN

Le modèle SAN qui modélise cet exemple utilise C automates pour modéliser les grappes de station de travail. Ces automates sont nommés $\mathcal{G}^{(i)}$, où $i \in [1..C]$. Les défaillances et les réparations des stations de travail sont représentées, respectivement, par les événements pp_i et rp_i où $i \in [1..C]$. On utilise aussi C automates pour modéliser les commutateurs liés à chaque grappes. Les automates des commutateurs sont nommés $\mathcal{S}^{(i)}$, où $i \in [1..C]$. Les événements ps_i, où $i \in [1..C]$, représentent la défaillance du commutateur $\mathcal{S}^{(i)}$ qui passe alors de l'état $A^{(i)}$ à l'état $P^{(i)}$. En revanche, les événements rs_i, où $i \in [1..C]$, représentent la réparation du commutateur $\mathcal{S}^{(i)}$ et fait passer l'automate de l'état $P^{(i)}$ à l'état $A^{(i)}$.

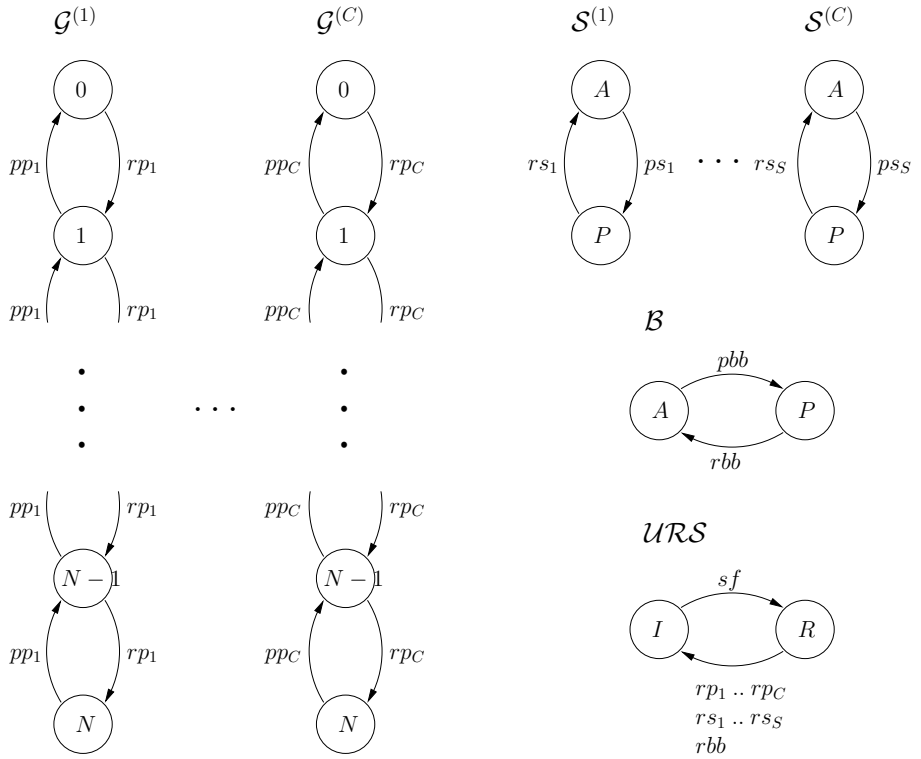
L'automate $\mathcal{B}$ modélise l'épine dorsale du système. Dans l'état A l'épine dorsale est *active* et dans l'état P elle est *en panne*. L'événement pbb représente une défaillance de l'épine dorsale et change l'état de l'automate de A vers P, tandis que l'événement rbb représente une réparation de l'épine dorsale et change l'état de l'automate de P vers A.

Le dernier automate $\mathcal{URS}$ modélise l'unité de réparation du système. L'unité de réparation peut être *inactive* (état I) ou en *réparation* (état R). L'événement sf représente l'inspection du système à la recherche d'un composant défaillant. Cette inspection est modélisée par la fonction f associée à l'événement. S'il y a un composant en panne, l'automate passe à l'état de réparation (R). À la fin de la réparation, une transition synchronisée fait passer l'automate à l'état I.

4.7 Conclusion

Dans ce chapitre, on a présenté différents exemples de modélisation à l'aide du formalisme des Réseaux d'Automates Stochastiques (SAN). Nous illustrons à travers de ces exemples les différents systèmes sujets aux défaillances pour lesquels on peut calculer à disponibilité ponctuelle.

Ces exemples sont utilisés pour les tests d'efficacité des méthodes de détection du régime stationnaire présentées dans le chapitre précédent (Chapitre 3). La comparaison de ces méthodes est présentée dans le chapitre suivant (Chapitre 5).



Type	Événement	Taux	Type	Événement	Taux
loc	pp_1	$\lambda_1.g_1$	syn	rp_1	μ_1
loc	pp_2	$\lambda_2.g_2$	syn	rp_2	μ_2
⋮	⋮	⋮	⋮	⋮	⋮
loc	pp_C	$\lambda_C.g_C$	syn	rp_C	μ_C

Type	Événement	Taux	Type	Événement	Taux
loc	ps_1	ρ_1	syn	rs_1	τ_1
loc	ps_2	ρ_2	syn	rs_2	τ_2
⋮	⋮	⋮	⋮	⋮	⋮
loc	ps_S	ρ_S	syn	rs_S	τ_S

Type	Événement	Taux
loc	pbb	γ
syn	rbb	δ
loc	sf	$\alpha.f$

$g_i = (st\ \mathcal{G}^{(i)})$

$f = ((nb\ [\mathcal{G}^{(1)} .. \mathcal{G}^{(C)}]\ N\ !=\ C)\ ||$
$(nb\ [\mathcal{S}^{(1)} .. \mathcal{S}^{(C)}]\ A\ !=\ C)\ ||$
$(st\ \mathcal{B} == P))$

FIG. 4.8 – Modèle SAN pour une grappe de postes de travail tolérante aux pannes

Chapitre 5

Analyse de résultats

Dans ce chapitre, on va procéder à une analyse et comparaison des mesures de performance des deux méthodes de détection du régime stationnaire présentées dans le chapitre 3 lorsque ces méthodes sont utilisées sur des grands modèles.

5.1 Efficacité des méthodes de détection du régime stationnaire

On rappelle que notre objectif est de tester l'efficacité des méthodes de détection du régime stationnaire sur de grands modèles. On a choisi différents modèles de façon à vérifier l'évolution des algorithmes par rapport à l'augmentation de la taille du modèle, aux différentes tailles d'ensembles d'états UP et aussi à l'erreur maximum acceptée.

On compare les méthodes de façon empirique sur la base des tests réalisés sur un ensemble de 6 modèles de systèmes sujets aux défaillances. L'ensemble de modèles utilisés pour l'obtention des mesures de performance sont ceux présentés dans le chapitre précédent (Chapitre 4). Chaque modèle présente différentes caractéristiques, telles que, différents taux de convergence, des états initiaux hors de l'ensemble d'états UP. Les différentes caractéristiques de chaque modèle permettent une analyse expérimentale plus complète des méthodes de détection du régime stationnaire, car différents scénarios sont étudiés.

On peut voir la nécessité d'avoir des modèles avec différentes caractéristiques et sur différents scénarios lorsqu'on analyse, par exemple, un modèle avec une très forte probabilité dans un seul état. Si on utilise la méthode de calcul de la disponibilité ponctuelle par la multiplication à gauche, on atteint le régime stationnaire plus tôt si l'état initial est le même que l'état avec forte probabilité. Si l'état initial est un état à très faible probabilité au régime stationnaire, on prend plus de temps pour atteindre le régime stationnaire et détecter la stationnarité.

De façon à prendre en compte les différentes caractéristiques et scénarios, on va analyser

l'efficacité des méthodes de détection du régime stationnaire de grands modèles sur différents critères, tel que la taille des modèles, l'ensemble d'états UP et l'erreur maximum acceptée.

Dans la comparaison et analyse des méthodes présentées ci-après, on utilise la notation suivante :

✍

UC	Nombre d'itérations calculé par la méthode de troncature de Fox-Glynn (Section 3.2.1) pour la méthode d'uniformisation classique ;
CRI	Méthode de convergence de vecteur (Algorithme 3.9) avec les tests de convergence du vecteur selon la convergence relative du vecteur (norme infinie relative) pour la multiplication à gauche ;
CAI	Méthode de convergence de vecteur (Algorithme 3.9) avec les tests de convergence du vecteur selon la convergence absolue du vecteur (norme infinie) pour la multiplication à droite ;
CAA	Méthode de convergence de vecteur (Algorithme 3.9) avec les tests de convergence du vecteur selon la convergence absolue accumulée du vecteur (norme L1) pour la multiplication à droite ;
CSW	Méthode de contrôle de la suite de w_n (Algorithme 3.10).

La méthode de convergence du vecteur (Algo. 3.9) avec un test de convergence relative (norme infinie relative) n'est pas applicable pour la multiplication à droite, car ce test de convergence présente des problèmes lorsque la disponibilité ponctuelle tend vers 0.

Les méthodes présentées ont été appliquées à l'ensemble des modèles choisis. Cependant, dans cette comparaison, on s'intéresse uniquement aux méthodes avec détection du régime stationnaire qui utilisent la multiplication à droite (les méthodes CAI, CAA et CSW), car ces méthodes utilisent la même méthodologie, c'est-à-dire, une multiplication matrice-vecteur. Cependant, les mesures obtenues pour les méthodes UC et CRI sont aussi présentées, pour le critère nombre d'itérations, à titre indicatif dans l'annexe A.

Les conclusions présentées dans ce chapitre font référence à un nombre limité de graphes, cependant elles sont faites sur la base d'une variété d'expérimentations dont les mesures complètes son présentés dans l'annexe A.

5.2 Comparaison des méthodes

La comparaison des méthodes de détection du régime stationnaire est réalisée selon deux critères : le nombre d'itérations nécessaires à la détection du régime stationnaire et la précision des résultats.

5.2.1 Nombre d'itérations

Les mesures relevées du nombre d'itérations nécessaires à la détection du régime stationnaire montrent que la méthode CAI est la plus performante parmi les 3 méthodes testées. FIG. 5.1 exemplifie cette constatation.

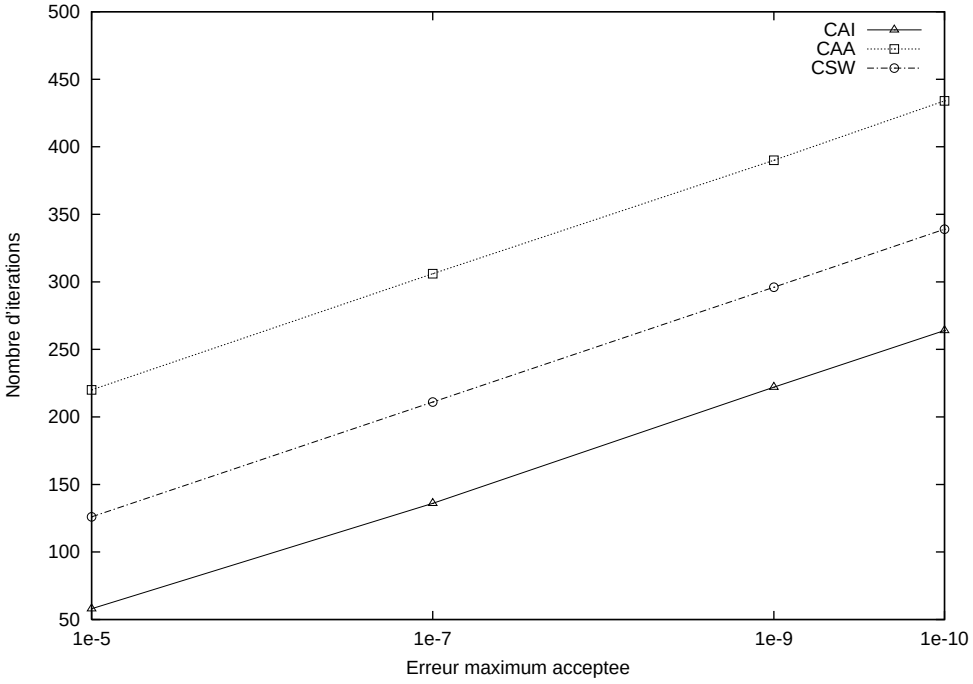


FIG. 5.1 – FAS- 15 serveurs

Cependant, la méthode CAI présente des problèmes de fausse détection du régime stationnaire. La fausse détection du régime stationnaire peut entraîner des erreurs assez importantes au niveau de la valeur de la disponibilité ponctuelle. Ce problème est présenté dans la Section 5.2.2.

On constate aussi que la différence du nombre d'itérations entre les méthodes ne suit pas une constante ni une fonction linéaire. Dans certains cas, la différence du nombre d'itérations nécessaires entre les méthodes est très importante et évolue de façon imprévisible. Cette constatation est clairement visible dans FIG. 5.2, quoique ce comportement est observé aussi sur d'autres modèles.

Lorsqu'on regarde de plus près les méthodes CAA et CSW, on observe une meilleure performance de la méthode CAA sur les petits modèles, notamment, les modèles avec moins d'un millier d'états environ, tandis que la méthode CSW a une meilleure performance sur les grands modèles.

La meilleure performance de la méthode CAA est visible pour les modèles FAS, MSA et CFR. Sur les modèles MSA ce comportement est observé sur tous les tests (Annexe A, Section

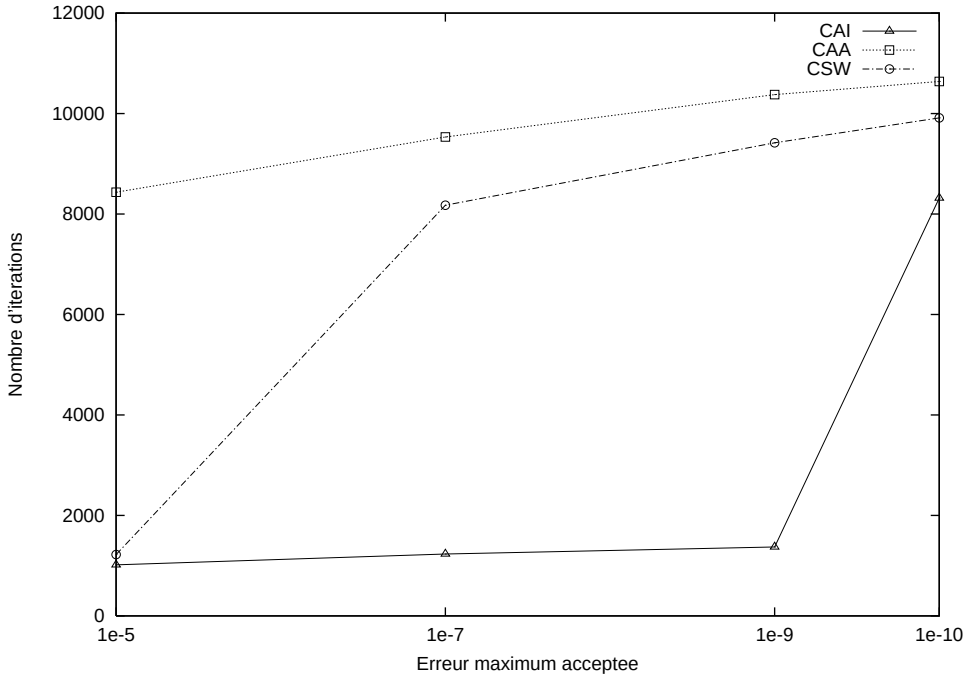


FIG. 5.2 – WORKSTATION- 512 serveurs et 10% des serveurs disponibles

A.2, page 219). FIG. 5.3 illustre cette observation. On remarque que les modèles MSA ont un espace d'états entre 34 et 514 états.

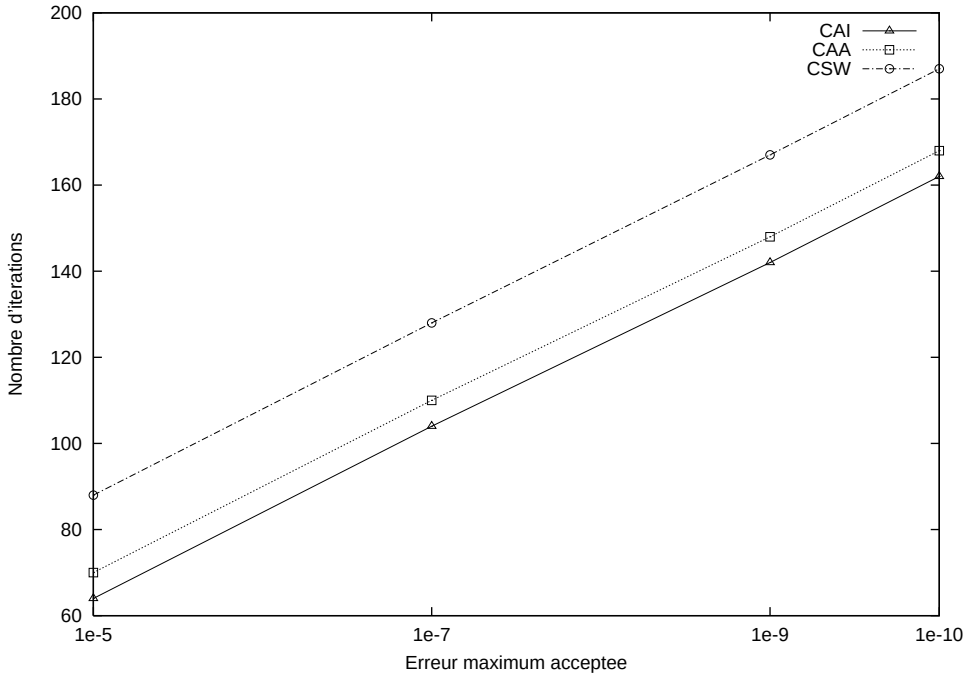


FIG. 5.3 – MSA- 256 serveurs

Ce même comportement est observé sur les mesures relevées sur le modèle FAS. Lorsque le modèle est testé avec seulement 5 serveurs (32 états), comme le montre FIG. 5.4 ci-après, la méthode CAA présente des meilleures résultats. Cependant, lorsqu'on teste ce modèle avec 10, 15 et 20 serveurs et que l'on a un espace d'états plus grand, $1\ 024$, $32\ 768$ et $1\ 048\ 576$ états respectivement, la méthode CSW prend moins d'itérations pour détecter le régime stationnaire pour n'importe quelle erreur maximum acceptée, comme on peut le voir sur des graphes supplémentaires dans l'annexe A (Section A.1, page 214).

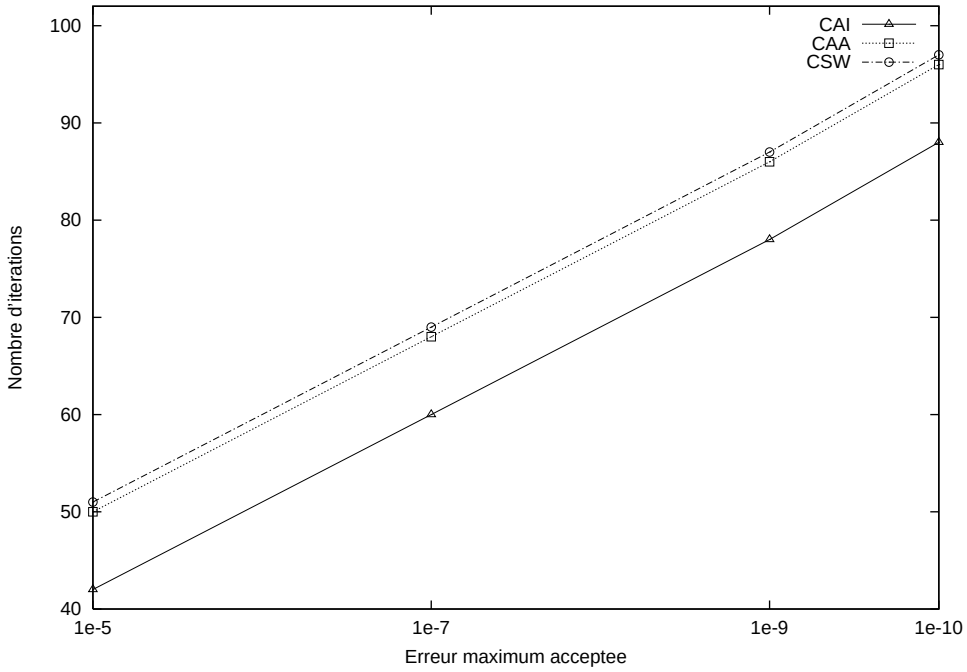


FIG. 5.4 – FAS- 5 serveurs

Même si on a observé une meilleure performance de la méthode CAA sur les petits modèles tandis que la méthode CSW obtient de meilleurs résultats sur les grands modèles, ce comportement n'est pas une règle stricte et on observe une exception.

Les mesures relevées sur les modèles CFR (Annexe A, Section A.3, page 224) montrent une meilleure performance de la méthode CAA lorsque l'erreur maximum acceptée est grande, notamment pour les erreurs maximum acceptées de $1e^{-5}$ et $1e^{-7}$, même sur les deux plus grands modèles (10 noeuds, $50\ 949$ états et 15 noeuds, $14\ 348\ 907$ états). Par contre, la performance de ces deux méthodes s'inverse sur les deux plus grands modèles lorsque l'erreur maximum acceptée devient plus petite ($1e^{-9}$ et $1e^{-10}$). On observe ce comportement sur les mesures relevées sur modèle CFR avec 15 noeuds présentées sur FIG. 5.5.

Sur les mesures relevées pour les modèles DDS (Annexe A, Section A.4, page 229), RWP (Annexe A, Section A.5, page 231) et WORKSTATION (Annexe A, Section A.6, page 236), la méthode CSW présente toujours des meilleurs résultats par rapport à la méthode CAA.

Dans la section suivante on analyse la précision des résultats obtenus pour chaque méthode.

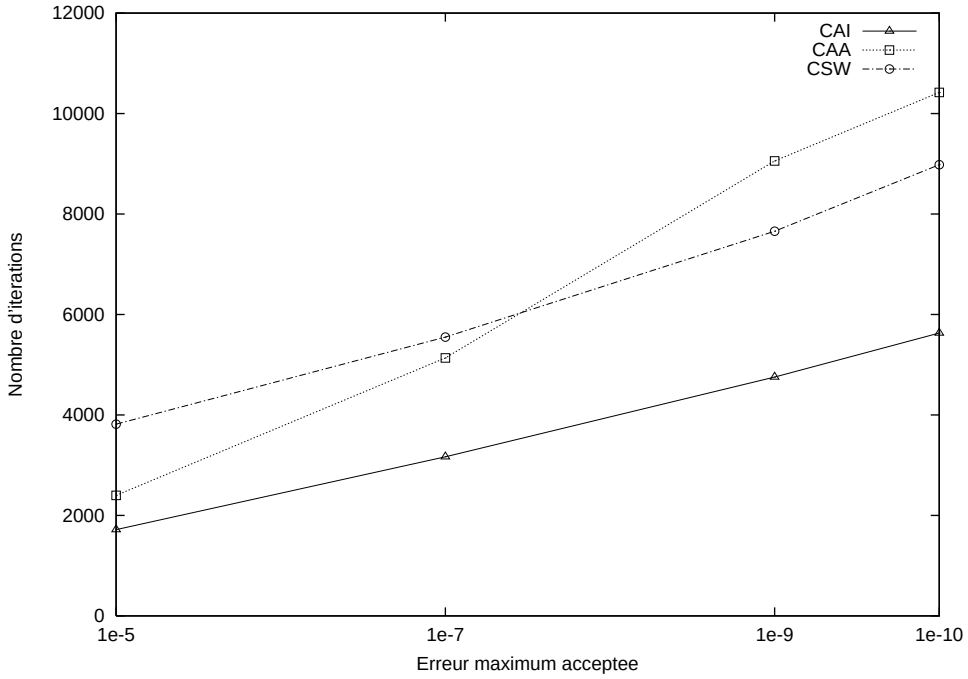


FIG. 5.5 – CFR- 15 noeuds

5.2.2 Précision des résultats

Un des principaux avantages de la méthode d'uniformisation est que le nombre de termes dans la somme de l'équation (3.8) (page 32) peut être tronqué et pré-calculé de façon à ce que l'erreur soit plus petite que une erreur maximum acceptée prédéfinie (Équation (3.14)). Cependant, cette propriété est perdue lorsqu'on incorpore une méthode de détection du régime stationnaire car le nombre d'itérations pour atteindre le régime stationnaire ne peut pas être prédit a priori.

Étant donné la taille des modèles testés, une solution exacte est infaisable pour plusieurs modèles. Donc, on ne peut pas calculer l'erreur exacte de la valeur de la disponibilité ponctuelle obtenue pour chaque méthode par rapport à la solution exacte. Cependant, Sericola prouve dans [91] que l'erreur obtenue par la méthode CSW ne dépasse pas l'erreur maximum acceptée prédéfinie. Donc, on utilise la valeur de la disponibilité ponctuelle obtenue pour une erreur maximum acceptée de $1e^{-11}$ par la méthode CSW pour calculer une borne d'erreur inférieure. La borne d'erreur inférieure est calculée par :

$$erreur = \left(X_{\varepsilon}(t) - CSW_{1e^{-11}}(t)\right) - 1e^{-11} \tag{5.1}$$

X_{ε} est la valeur de la disponibilité ponctuelle pour une erreur maximum acceptée ε avec l'une des méthodes CAI, CAA et CSW à l'instant de temps t.

On considère que la méthode est précise si la borne d'erreur calculée est inférieure à l'erreur

maximum acceptée prédéfinie.

La borne d'erreur est calculée sur la valeur de la disponibilité ponctuelle à l'instant de temps t, le plus grand utilisé sur chaque modèle. Pour chaque modèle, l'instant de temps t a été choisi suffisamment large de façon à ce que toutes les méthodes puissent détecter le régime stationnaire. Ces valeurs sont présentées dans l'annexe A avec la description des valeurs utilisées pour les tests de chaque modèle.

Sur les graphes présentés ci-après, une courbe avec l'erreur maximum acceptée a été introduite. Cette courbe est identifiée par l'étiquette *Erreur acceptée* et vise à faciliter l'analyse des graphes.

On constate que les méthodes CAI et CAA, qui sont basées sur la méthode de tests de convergence du vecteur (Algorithme 3.9, page 50), présentent des problèmes de précision causés par une fausse détection du régime stationnaire. Cette fausse détection résulte en une valeur de disponibilité ponctuelle qui ne respecte pas l'erreur maximum acceptée. Même si les deux méthodes ont présenté des problèmes, la méthode CAI a les résultats les plus critiques. Les pires cas sont observés sur le modèle RWP, comme on peut le voir sur FIG. 5.6. On observe sur FIG. 5.6 que la borne d'erreur calculée pour la méthode CAI est supérieure à l'erreur maximum acceptée de $1e^{-9}$ prédéfinie.

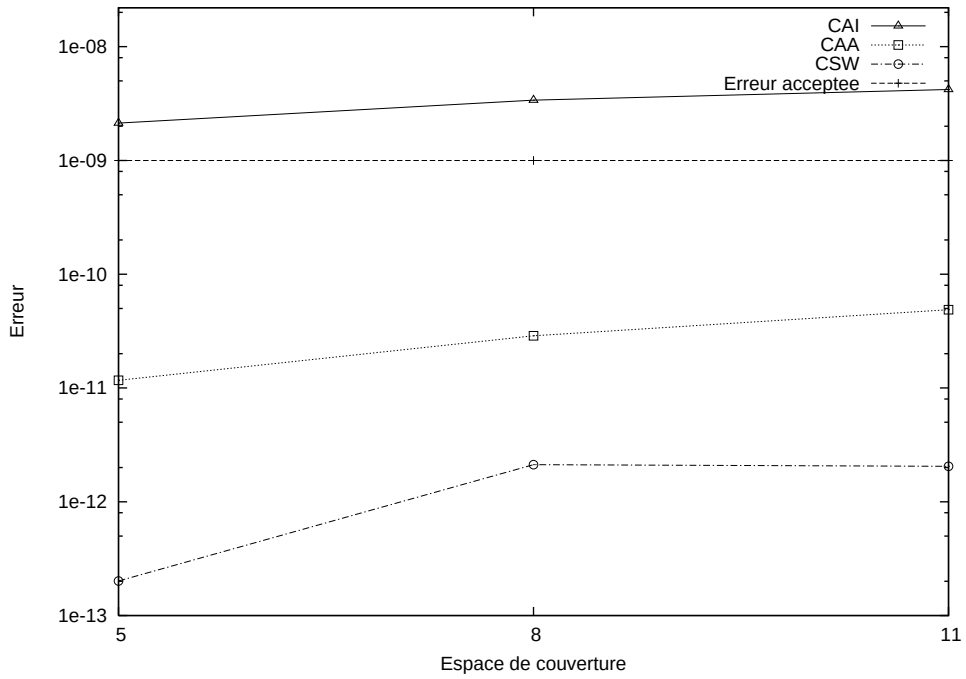


FIG. 5.6 – RWP- erreur maximum acceptée de $1e^{-9}$

La méthode CAI a présenté aussi ce problème de fausse détection du régime stationnaire sur les modèles FAS, CFR, RWP et WORKSTATION, comme on peut le constater sur l'ensemble des mesures présentés dans l'annexe A sections A.1, A.3, A.5 et A.6.

Même si la méthode CAA a présenté moins des problèmes que la méthode CAI, elle n'est pas fiable à 100%. La méthode CAA a présenté ce problème uniquement sur le modèle CFR avec 5 noeuds (243 états) pour toutes les erreurs maximum acceptées testées. FIG. 5.7 présente la borne d'erreur calculée pour le modèle CFR avec 5 noeuds.

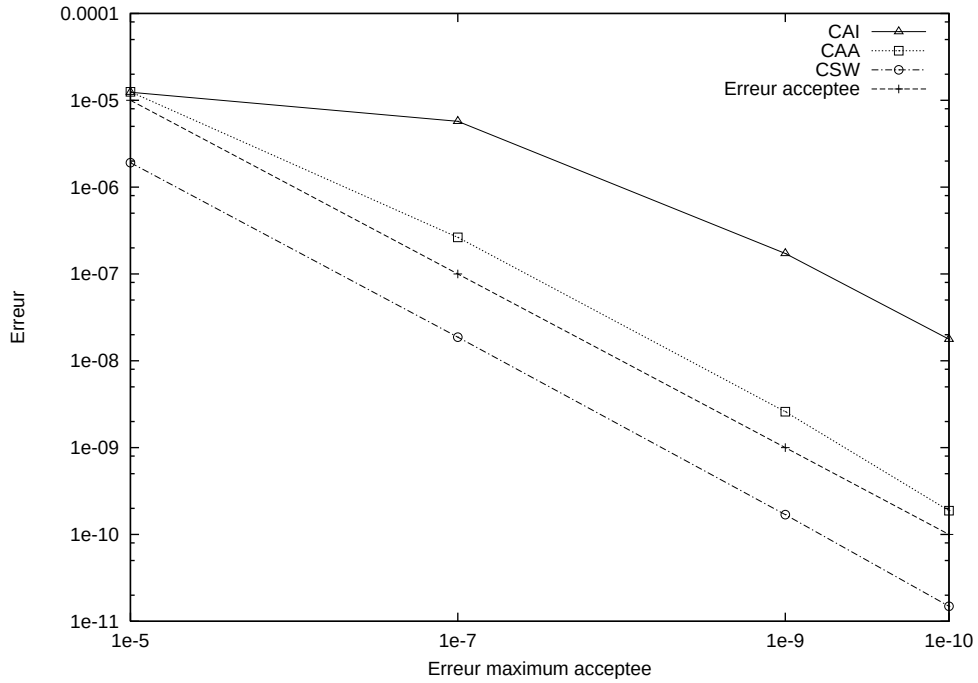


FIG. 5.7 – CFR- 5 noeuds

On a pu observer aussi que l'erreur calculée pour la méthode CSW est toujours plus petite que l'erreur maximum acceptée sur tous les modèles testés. Cependant, on a observé une mauvaise qualité des résultats obtenus pour une erreur maximum acceptée égale à $1e^{-5}$ sur les modèles FAS et WORKSTATION. Même si les résultats obtenus sont plus petits que l'erreur maximum acceptée, elle présente une erreur plus grande lorsqu'on les compare avec les méthodes CAI et CAA. Cette constatation peut être observée dans FIG. 5.8, mais elle apparaît sur plusieurs modèles WORKSTATION, comme on peut le voir dans l'annexe A dans la section A.6.

Ce comportement est souvent observé sur d'autres modèles WORKSTATION.

5.3 Analyse des paramètres

En plus d'une comparaison entre les méthodes de détection du régime stationnaire, on regarde aussi l'influence des différents paramètres, tels que la taille du modèle, l'ensemble d'états UP et l'erreur maximum acceptée, sur chaque méthode. Dans les sections suivantes, on présente nos constatations sur l'influence de ces paramètres sur les méthodes. Ces constatations sont basées sur une variété d'expérimentations réalisées sur plusieurs modèles. Les graphes

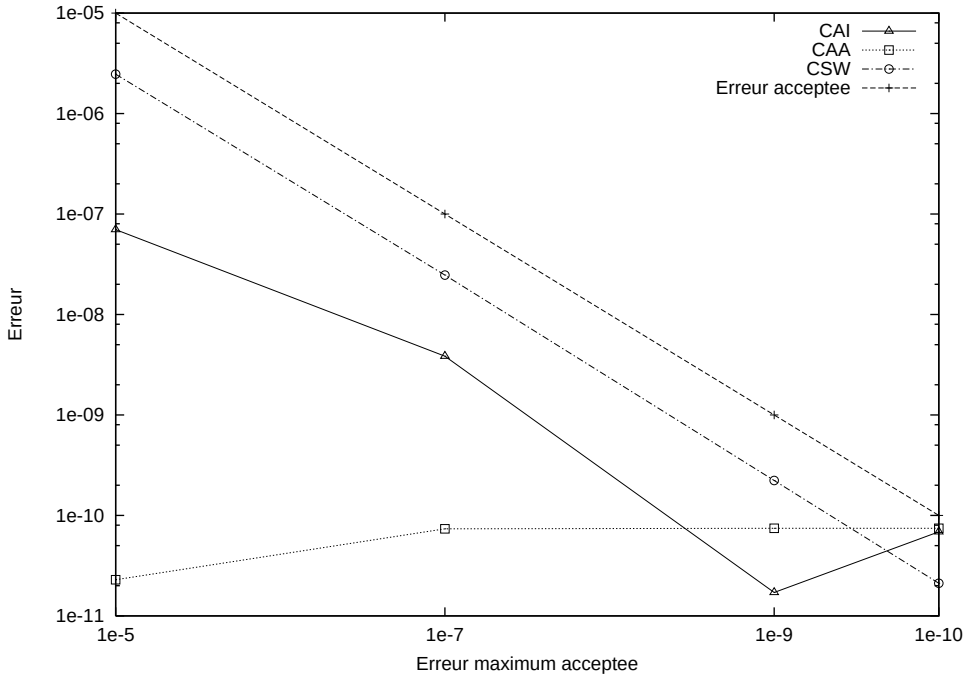


FIG. 5.8 – WORKSTATION- 32 serveurs et 50% des serveurs disponibles

présentés dans cette section sont ceux qui caractérisent le plus chaque cas. L'ensemble complet des mesures est présenté dans l'annexe A.

5.3.1 Taille des Modèles

Parmi les 6 différents modèles testés, 4 modèles ont une variation possible de la taille, soit par l'addition de nouveaux automates, soit par l'addition d'états aux automates déjà existants.

Sur les modèles FAS et CFR, l'augmentation de la taille du modèle se fait par l'inclusion de nouveaux automates, alors que sur les modèles MSA et WORKSTATION, la taille du modèle croit avec l'augmentation du nombre d'états dans les automates déjà existants.

L'inclusion d'automates dans le modèle entraîne un changement de la structure de la matrice, mais conserve le facteur entre la plus grande et la plus petite valeur de la matrice. Tandis que dans les modèles MSA et WORKSTATION, l'inclusion de nouveaux états sur les automates déjà en place fait augmenter le facteur entre la plus grande et la plus petite valeur de la matrice. L'ordre de grandeur de ce facteur peut jouer pour une convergence plus rapide ou plus lente [92].

Sur les modèles FAS et CFR, on observe une tendance de stabilisation et même à la baisse du nombre d'itérations, sur certains modèles, lorsque les modèles grandissent, comme on peut le voir sur le modèle CFR (FIG. 5.9).

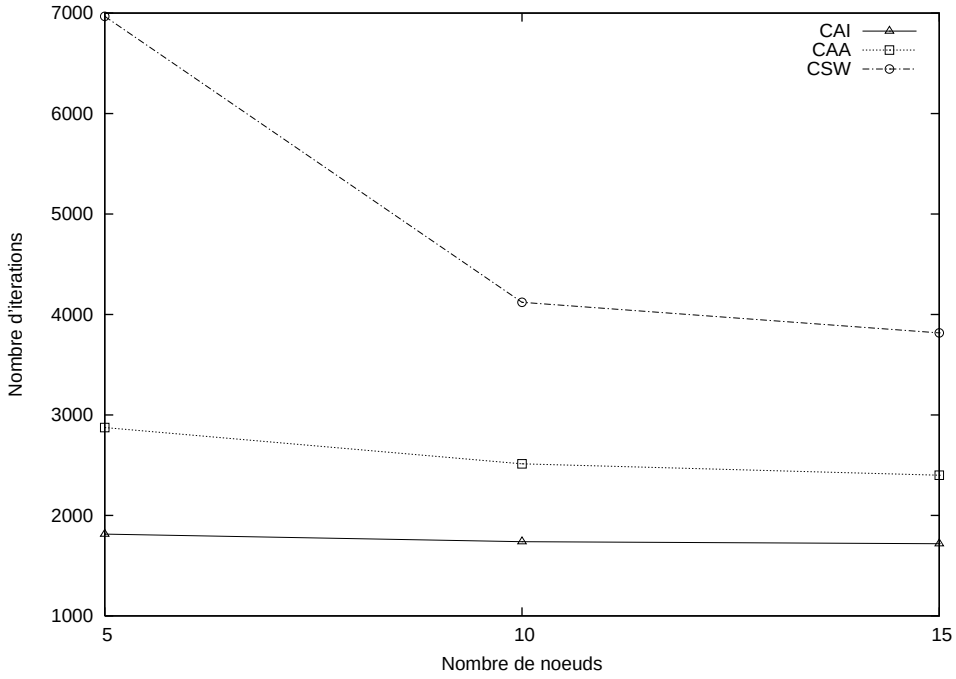


FIG. 5.9 – CFR

Sur les modèles MSA et WORKSTATION, on n'observe pas un comportement standard. FIG. 5.10 montre assez bien ce phénomène.

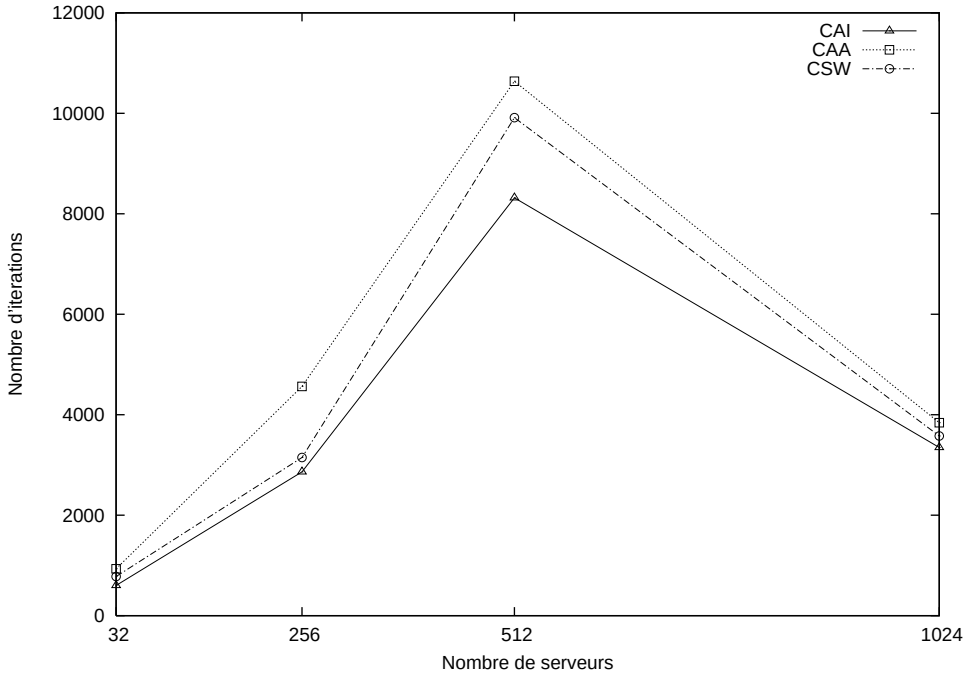


FIG. 5.10 – WORKSTATION

Sur la base des modèles testés, on peut constater que la taille du modèle n'a pas toujours la même influence sur le nombre d'itérations nécessaires pour atteindre le régime stationnaire. Nos hypothèses sur la relation entre facteur de la plus grande et la plus petite valeur de la matrice et la réduction ou l'augmentation du nombre d'itérations sur les modèles FAS et CFR sont basées sur nos connaissances des modèles testés, mais une analyse plus approfondie des structures des matrices P est nécessaire pour prouver ces relations.

5.3.2 Ensembles d'états UP

La relation entre l'état initial et le nombre d'itérations nécessaires pour le régime stationnaire est bien connue pour les méthodes qui utilisent la multiplication à gauche. Autrement dit, le nombre d'itérations pour atteindre le régime stationnaire dépend fortement de l'état initial du modèle. Lorsqu'on utilise la multiplication à droite, le nombre d'itérations pour atteindre le régime stationnaire ne dépend plus de l'état initial du modèle, mais du vecteur d'états UP.

Étant donné que, à chaque multiplication la suite des vecteurs w_n modifie les valeurs 0, initialement attribuée aux états $DOWN$, par des valeurs qui tendent vers la valeur de la disponibilité ponctuelle, on essaie d'établir ici une relation entre la taille de l'ensemble UP par rapport à la taille du modèle et la vitesse de convergence.

Deux modèles ont été utilisés pour tester différents ensembles d'états UP. Sur le modèle RWP, on a calculé la disponibilité ponctuelle pour 3 aires de couverture : 5, 8 et 11. Les 3 aires de couverture utilisées dans nos tests correspondent, respectivement à $25\ 625$, $70\ 625$ et $138\ 125$ états, soit 6.56%, 18.08% et 35.36% de l'espace d'état du modèle. Pour ce modèle, on observe une croissance constante du nombre d'itérations lorsque le nombre d'états dans l'ensemble UP augmente. Ce comportement est observé sur les trois méthodes comme on peut voir sur FIG. 5.11.

Sur les modèles WORKSTATION testés, on n'a pas changé uniquement l'ensemble d'états UP mais aussi la taille du modèle. On a testé 4 nombres de serveurs différents, 32, 256, 512 et 1024 serveurs. Pour chaque nombre de serveurs, on a calculé la disponibilité ponctuelle pour 5 niveaux de disponibilité. Le niveau le plus bas d'exigence a été établi à 10% des serveurs accessibles. Ce qui correspond à environ 64% de l'espace d'états. Pour les niveaux de disponibilités égaux à 30%, 50%, 70% et 90%, on a des ensembles d'états UP qui correspondent, respectivement à environ 38%, 6.3%, 2.3% 0.25% de l'espace d'états de chaque modèle.

Sur les deux plus petits modèles (32 et 256 serveurs), on observe une croissance du nombre d'itérations lorsque le pourcentage de l'ensemble d'états UP diminue. Cette constatation est illustrée par le modèle avec 32 serveurs (FIG. 5.12), où l'on a observé l'évolution du nombre d'itérations par rapport à différents ensemble d'états UP pour une erreur maximum acceptée de $1e^{-10}$.

Cependant, on n'observe pas cette relation sur les deux plus grands modèles, avec 512 et $1\ 024$ serveurs. Les mesures obtenues sur le modèle avec 512 serveurs nous montrent un nombre plus grand d'itérations pour un niveau de disponibilité de 50% (FIG. 5.13), pendant que sur le

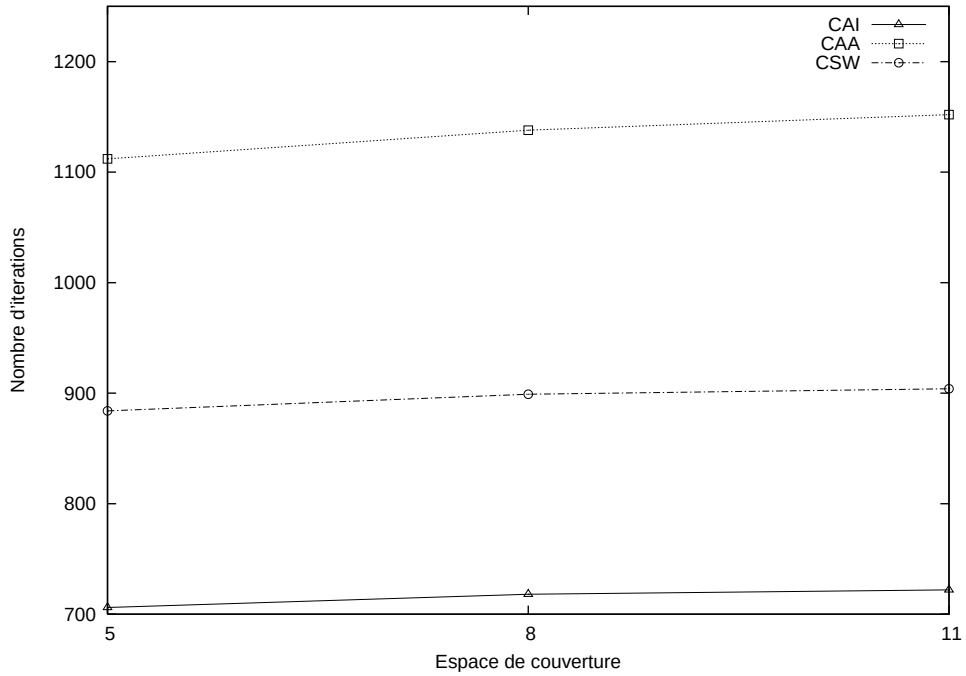


FIG. 5.11 – RWP

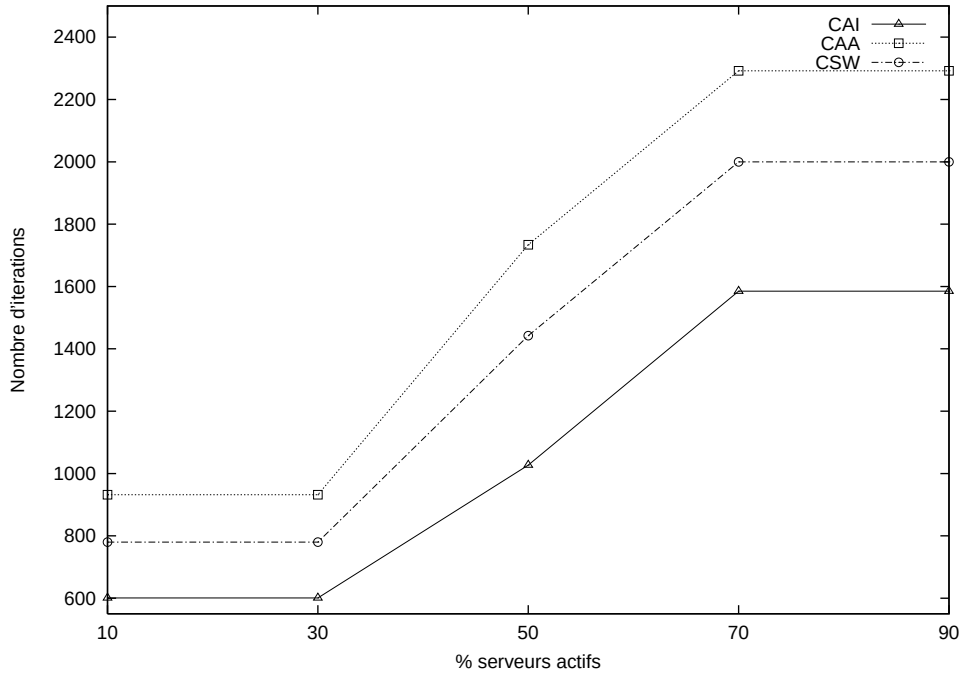


FIG. 5.12 – WORKSTATION

modèle avec $1\,024$ serveurs, on constate deux pics du nombre d'itérations pour les niveaux de disponibilité de 30% et 70% (FIG. 5.14). Même si on peut observer que les trois méthodes

suivent la même tendance, les méthodes CAA et CSW sont les plus touchées par ce phénomène.

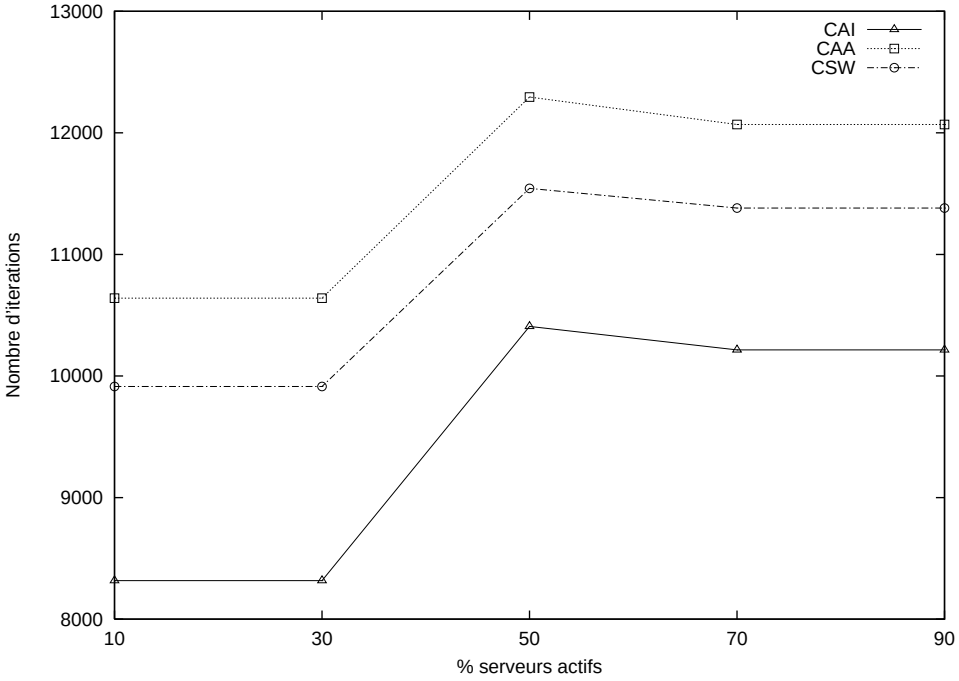


FIG. 5.13 – WORKSTATION

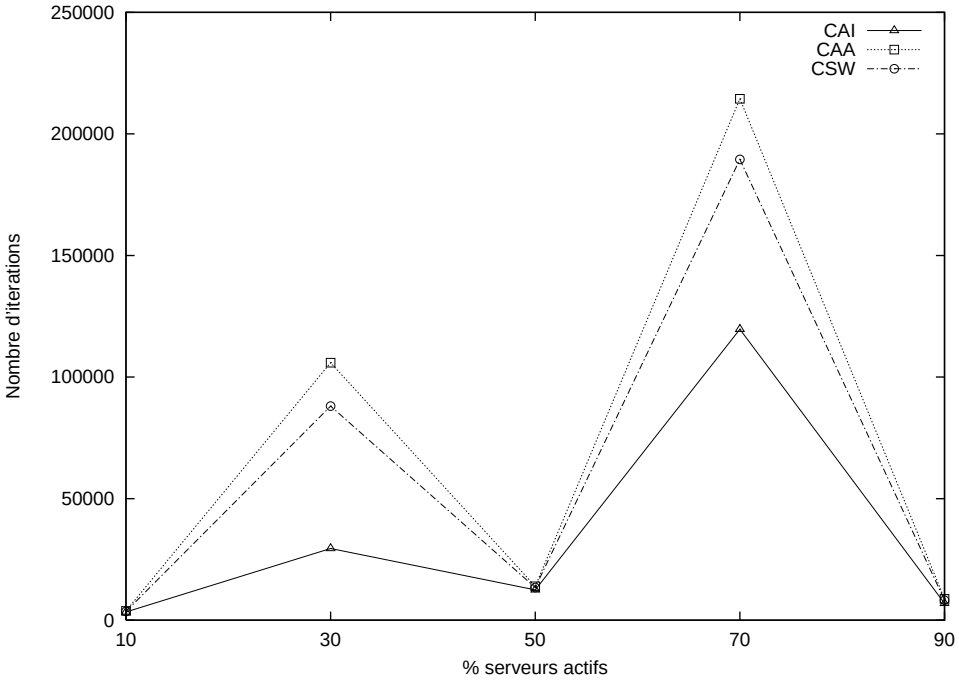


FIG. 5.14 – WORKSTATION

D'après les mesures relevées, on peut constater que, sur certains modèles, la taille de l'ensemble d'états UP influence le nombre d'itérations nécessaires pour atteindre le régime stationnaire. Cependant, malgré l'ensemble de tests réalisés, on n'est pas en mesure d'établir une relation directe entre la taille de l'ensemble d'états UP et la taille du modèle, ou sur quel type de modèles cette influence existe.

5.3.3 L'erreur maximum acceptée

On peut observer dans tous les modèles que le nombre d'itérations nécessaires pour détecter le régime stationnaire augmente au fur et à mesure qu'on réduit l'erreur maximum acceptée.

Cette augmentation est souvent du type linéaire. Ce comportement peut être observé sur la plus grande partie des modèles et sur les trois méthodes testées. Cependant, on peut observer des cas où les méthodes CAI et CAA ont une forte croissance du nombre d'itérations pour une erreur maximum acceptée petite. Pour la méthode CAA le cas le plus visible apparaît sur le modèle DDS, comme on peut le constater sur FIG. 5.15.

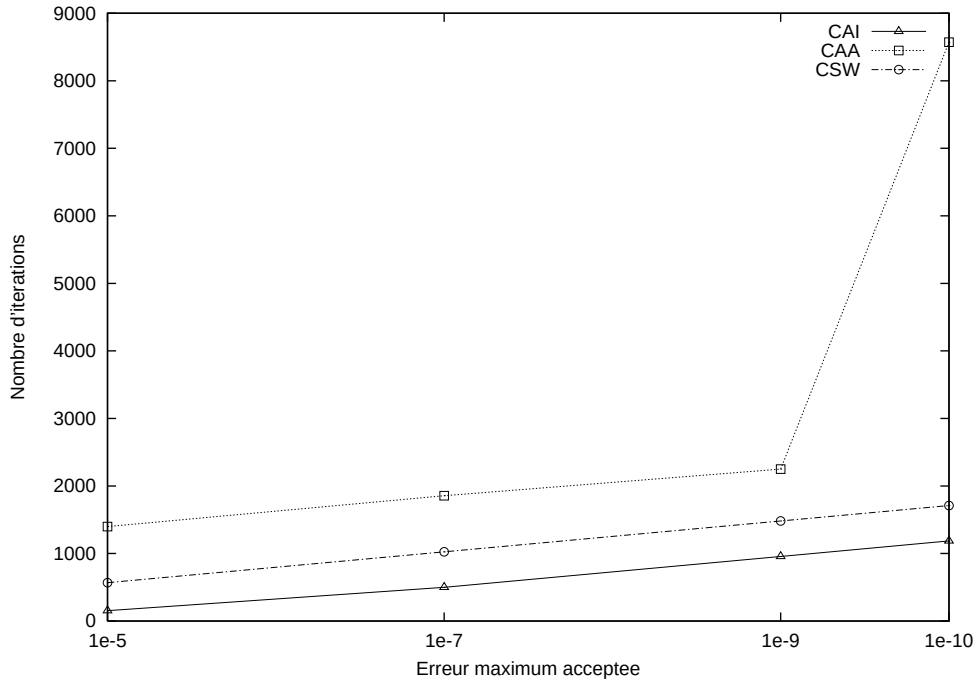


FIG. 5.15 – DDS

Ce comportement peut avoir une relation entre proximité de la plus petite valeur de la matrice et de l'erreur maximum acceptée. Cependant une analyse plus approfondie est nécessaire pour prouver ce relation.

5.4 Comparaison des différentes représentation du vecteur constant

On a testé l'efficacité de trois formats pour la représentation d'états initiaux (multiplication à droite) et pour la représentation d'états UP (multiplication à gauche) présentés dans la section 3.5 (Chapitre 3). Les mesures obtenues sont analysées par rapport à la vitesse de calcul mais aussi par rapport à l'espace de stockage.

5.4.1 Vitesse de calcul

TAB. 5.1 présente les mesures relevées pour les 3 implantations du vecteur constant, pour la multiplication à droite et aussi pour la multiplication à gauche.

Modèle	Vecteur plein		Vecteur creux		Fonction	
	Gauche	Droite	Gauche	Droite	Gauche	Droite
FAS- 20 serveurs	$2,20$s	$2,26$s	$2,15$s	$2,20$s	$2,16$s	$2,40$s
CFR- 15 noeuds	$23,21$s	$30,28$s	$23,17$s	$29,73$s	$24,94$s	$31,92$s
MSA- 256 serveurs	$0,01$s	$0,01$s	$\approx 0,0$s	$0,01$s	$\approx 0,0$s	$0,01$s
DDS	$7,32$s	$7,39$s	$7,26$s	$7,29$s	$7,91$s	$7,86$s
RWP- 11 espaces	$2,20$s	$2,19$s	$2,10$s	$2,13$s	$2,13$s	$2,14$s
WORKSTATION- 1024 serveurs	$8,66$s	$8,38$s	$7,39$s	$8,01$s	$7,66$s	$10,84$s

TAB. 5.1 – Vitesse de calcul

On rappelle que notre objectif est de comparer les 3 implantations des vecteurs constants pour la multiplication à droite et à gauche. Les différences de vitesse de multiplication observées entre la multiplication à droite et à gauche ne sont pas à comparer ici car l'ordre de multiplication a une forte influence sur la vitesse de multiplication [34]. En plus, sur les implantations avec des vecteurs creux et avec des fonctions, une comparaison entre l'implantation à gauche et à droite n'est pas sensée, étant donné que, pour l'implantation avec des vecteurs creux, les tailles des vecteurs sont assez différentes. Par exemple, sur le modèle WORKSTATION avec $1\ 024$ serveurs, la taille du vecteur constant pour la multiplication à droite est égal à 1 et pour la multiplication à gauche elle est égal à $7\ 027\ 848$. Du même pour l'implantation avec des fonctions, la complexité d'évaluation des fonctions qui définissent l'état initial et l'ensemble d'état UP n'est pas toujours la même.

Sur les 3 implantations proposées, on peut constater une claire supériorité de l'implantation avec des vecteurs creux sur les autres implantations et cela est le choix recommandé. Les mesures relevées pour l'implantation avec un vecteur plein ou par des fonctions ne permet pas de définir la meilleure performance, surtout pour la multiplication à droite. Dans ce cas, l'implantation par des fonctions est plus intéressante par le coût réduit en espace mémoire.

5.4.2 Espace de stockage

TAB. 5.2 présente l'espace mémoire nécessaire à le stockage du vecteur constant pour les 3 implantations proposées, pour la multiplication à droite et aussi pour la multiplication à gauche.

Modèle	Vecteur Plein	Vecteur Creux		Fonction
		Gauche	Droite	
FAS- 20 serveurs	$8,00$ Mo	$4,00$ Mo	$0,01$ Mo	$0,00$ Mo
CFR- 15 noeuds	$109,47$ Mo	$54,74$ Mo	$0,01$ Mo	$0,00$ Mo
MSA- 256 serveurs	$4,02$ Mo	$1,00$ Mo	$0,01$ Mo	$0,00$ Mo
DDS	$26,82$ Mo	$0,02$ Mo	$0,01$ Mo	$0,00$ Mo
RWP- 11 espaces	$2,98$ Mo	$0,10$ Mo	$0,01$ Mo	$0,00$ Mo
WORKSTATION- 1024 serveurs	$128,25$ Mo	$17,32$ Mo	$0,01$ Mo	$0,00$ Mo

TAB. 5.2 – Espace de stockage

On remarque d'abord que l'espace nécessaire à la représentation de la fonction est insignifiante dans les deux implantations. Cette méthode est surement la plus économique de toutes pour l'espace de stockage.

La représentation du vecteur constant par un vecteur plein est déconseillée pour la grande partie de cas. Le seul cas où l'implantation en plein est conseillée est pour la multiplication à droite lorsque le nombre de valeurs non-nulle dans le vecteur initial dépasse les 50% de la taille du vecteur.

Sur l'implantation du vecteur constant par un vecteur creux, l'espace de stockage dépend du nombre de valeurs différentes de zéro dans le vecteur d'états initial, pour la multiplication à droite et de la taille de l'ensemble d'états UP, pour la multiplication à gauche.

Notons que pour la multiplication à gauche, un seul vecteur d'entiers est stocké pour représenter l'ensemble des états UP. Ceci justifie la moitie de la taille de l'espace de stockage pour le modèle où la taille de l'ensemble UP est presque égal à la taille de l'espace d'états produit.

5.5 Conclusion

Dans ce chapitre, on a analysé l'efficacité de deux méthodes de détection du régime stationnaire sur des grands modèles.

La comparaison des méthodes présentée dans ce chapitre a été faite sur différents modèles et paramètres de façon à voir le comportement de chaque méthode face à différents critères, tels que la taille des modèles, l'erreur maximum acceptée et l'ensemble d'états UP.

Parmi toutes les méthodes testées, la performance de la méthode CAI, par rapport au nombre d'itérations, est nettement meilleure que les méthodes CAA et CSW.

Cependant, les problèmes de fausse détection du régime stationnaire entraîne des erreurs importantes sur la valeur de la disponibilité ponctuelle et compromet les bons résultats obtenus sur le critère nombre d'itérations et rend la méthode CAI pas fiable sur de grands modèles.

Même si la méthode CAA présente moins de problèmes de fausse détection du régime stationnaire que la méthode CAI, le nombre d'itérations nécessaires pour la détection du régime stationnaire ne rend pas cette méthode attractive dans la plus grande partie des modèles testés.

Finalement, la méthode CSW présente des résultats corrects avec un surcoût entre 20% et 500% par rapport à la méthode la plus performante (CAI). En outre, cette méthode permet un calcul fiable et rapide de la disponibilité au régime stationnaire, ce qui évite le problème d'instabilité de la méthode d'uniformisation classique pour des instants de temps t assez grands. Il est évident que pour les grands modèles, cette méthode est la méthode recommandée.

Un deuxième point étudié dans ce chapitre a été l'utilisation de différentes implantations pour le vecteur constant. Parmi les trois implantations testées : vecteur plein, vecteur creux et fonction, le choix du vecteur creux semble assez évidente, cars on a un bon compromis entre la vitesse de calcul et l'espace de stockage nécessaire. On peut envisager l'implantation d'une heuristique qui choisit automatiquement le meilleur format de représentation par rapport aux taux de remplissage du vecteur.

Deuxième partie

Réseaux d’automates stochastiques à temps discret

Chapitre 6

Formalisme SAN à temps discret

Depuis la proposition du formalisme des Réseaux d'Automates Stochastiques (*Stochastic Automata Networks* - SAN) par Plateau dans [72, 73], la plupart des travaux ont porté sur une échelle de temps continue. Le formalisme à temps discret a reçu beaucoup moins d'attention et cela s'explique par la difficulté de modéliser des systèmes complexes avec des distributions discrètes.

Dans ce chapitre[1], on va donner les définitions de base du formalisme SAN à temps discret (Section 6.2) qui nous permettrons de définir un algorithme pour la construction de la chaîne de Markov représentée (Section 6.3). Mais d'abord, pour mieux comprendre le problème, la Section 6.1 introduit les motivations et travaux précédents sur les SAN et aussi sur d'autres formalismes structurés.

6.1 Motivation et travaux précédents

Contrairement aux systèmes modélisés sur une échelle de temps continue, où un seul événement peut avoir lieu à chaque instant de temps, en temps discret, plusieurs événements peuvent avoir lieu dans une même unité de temps et donc, chaque combinaison d'événement possible doit être déterminée. Par exemple, pour deux événements e_1 et e_2, 4 combinaisons sont possibles : e_1 a lieu et e_2 n'a pas lieu ; e_1 n'a pas lieu et e_2 a lieu ; e_1 a lieu et e_2 a lieu aussi ; et e_1 n'a pas lieu et e_2 n'a pas lieu non plus.

Cependant, la grande difficulté de la modélisation de systèmes à temps discret n'est pas de déterminer toutes les combinaisons possibles, mais de résoudre les conflits, lorsque le tirage d'un événement amène à un état où l'autre n'est plus réalisable.

On trouve dans la littérature un certain nombre de propositions de formalismes pour mod-

[1] La présentation du formalisme SAN à temps discret donnée dans ce chapitre est commune dans cette thèse et dans [87].

éliser des systèmes à temps discret : Réseaux de Petri Stochastique [65, 19], Réseaux de Files d'Attente [46] et Réseaux d'Automates Stochastiques [74, 9].

Au niveau des Réseaux de Petri Stochastiques, différentes sémantiques ont été proposées pour traiter le problème des transitions concurrentes (deux transitions réalisables dans la même unité de temps) [65, 19, 102, 82]. Pour illustrer ceci, supposons que plusieurs transitions soient possibles au même instant de temps, par exemple les transitions t_1 et t_2 dans FIG. 6.1. Ces deux transitions sont dites *concurrentes*, car le déclenchement d'une transition empêche l'occurrence de l'autre.

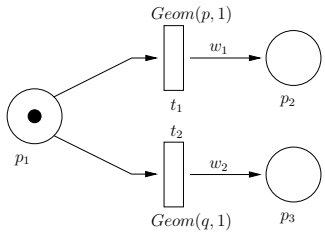


FIG. 6.1 – Exemple d'un réseau de Petri avec transitions concurrentes

Une des premières sémantiques pour traiter la concurrence a été proposée par Molloy dans [65]. Dans cette proposition, deux transitions concurrentes ne peuvent pas avoir lieu en même temps. Notons que, sur le graphe de marquage du réseau de Petri obtenu par cette sémantique, on insère des probabilités de choix, comme indiqué sur l'exemple de FIG. 6.2, où p est la probabilité d'occurrence de la transition t_1 et q la probabilité d'occurrence de t_2.

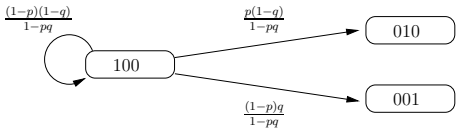


FIG. 6.2 – Graphe de marquage du modèle de FIG. 6.1 selon Molloy

Une autre sémantique a été proposée par Ciardo dans [19]. Dans cette sémantique, Ciardo propose d'associer un poids à chaque transition, ici w_1 et w_2 pour le modèle de FIG. 6.1, et lorsque les deux transitions sont potentiellement réalisables en même temps, la probabilité de la transition effectivement réalisé est pondérée par la probabilité de chaque transition. FIG. 6.3 montre le graphe de marquage obtenu par cette sémantique.

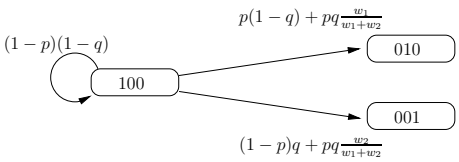


FIG. 6.3 – Graphe de marquage du modèle de FIG. 6.1 selon Ciardo

Dans ces deux approches, la simultanéité est écartée alors qu'elle peut avoir un sens physique dans certains systèmes.

Au niveau des Réseaux d'Automates Stochastiques (SAN), les travaux sont moins nombreux [74, 9]. La première proposition de SAN à temps discret a été faite par Plateau et Atif dans [74]. Dans cette approche, la notion d'*événements compatibles* a été introduite pour définir les ensembles d'événements qui peuvent se réaliser de façon simultanée. Cependant, la sémantique à adopter en cas de conflit n'est pas très explicite et le descripteur markovien obtenu devient souvent très complexe à cause du grand nombre de combinaisons possibles d'événements compatibles. Quelques méthodes ont été proposées pour simplifier cette approche, soit par l'agrégation/désagrégation des automates [48], soit par les propriétés structurelles du modèle [39], mais ces méthodes s'appliquent à un sous-ensemble très réduit de modèles.

Une nouvelle proposition a été faite par Benoit dans [9]. Dans cette proposition, Benoit introduit la notion d'*ordre de priorité* entre les événements du modèle. L'ordre de priorité permet de définir le comportement à suivre en cas de conflit et rend possible la construction d'une sémantique de chaîne de Markov.

La définition formelle présentée dans la section 6.2 utilise la sémantique des SAN à temps discret proposé par Benoit dans [9]. Dans la section 6.3 nous définissons les règles de construction de l'automate global d'un modèle SAN. La section 6.4 présente la procédure d'obtention de la chaîne de Markov représentée par le modèle SAN. Enfin, la section 6.5 présente les conclusions de ce chapitre.

6.2 Formalisme des Réseaux d'Automates Stochastiques

Dans cette section, on va présenter les notations et les définitions de base qui seront utilisées pour la construction du Descripteur Markovien d'un modèle SAN à temps discret.

On va considérer un modèle SAN comprenant N automates et E événements.

✍ **Notation**

$\mathcal{A}$	l'ensemble des automates[2] ($\mid \mathcal{A} \mid = N$).

6.2.1 Définitions et notations de base pour un automate dans un réseau

On note $\mathcal{A}^{(i)}$, où $i \in [1..N]$, le i-ème automate d'un modèle SAN.

Pour les notations suivantes, on adopte $[i..j]$ pour le sous-ensemble de $\mathbb{N}$ contenant toutes les valeurs de i jusqu'à j (ces valeurs incluses) et $[i, j]$ pour le sous-ensemble de $\mathbb{R}$ (des valeurs

[2] On adopte la notation $\mid \mathcal{X} \mid$ afin de définir la cardinalité d'un ensemble $\mathcal{X}$.

continues) contenant toutes les valeurs de i jusqu'à j.

✍ **Notation**

$\mathcal{S}^{(i)}$	l'ensemble d'états de l'automate $\mathcal{A}^{(i)}$;
$x^{(i)}$	l'état local de l'automate $\mathcal{A}^{(i)}$, où $x^{(i)} \in \mathcal{S}^{(i)}$.

Définition 6.2.1. *À tous les ensembles $\mathcal{S}^{(i)}$ on rajoute un état Φ, appelé état* **fantôme**.

Cet état sera utilisé comme état intermédiaire transitoire lorsque plusieurs événements ont lieu dans la même unité de temps. On verra l'utilité de l'état fantôme plus en détail sur l'automate $\mathcal{A}^{(1)}$ dans FIG. 6.4.

✍ **Notation**

$\breve{\mathcal{S}}^{(i)}$	l'ensemble d'états d'un automate $\mathcal{A}^{(i)}$ d'un modèle SAN, $\breve{\mathcal{S}}^{(i)} = \mathcal{S}^{(i)} \cup \Phi$.

Définition 6.2.2. *L'espace d'états produit $\mathcal{S}$ d'un modèle* SAN *est défini par le produit cartésien des espaces d'états $\mathcal{S}^{(i)}$, i.e., $\mathcal{S} = \prod_{i=1}^{N} \mathcal{S}^{(i)}$.*

✍ **Notation**

$\tilde{x}$	l'état global d'un modèle SAN défini par la combinaison des états locaux des N automates, $\tilde{x} = (x^{(1)}, \ldots, x^{(N)})$, où $\tilde{x} \in \mathcal{S}$;
ω	un ensemble d'indices d'automates, où $\omega \subseteq [1..N]$;
$\tilde{x}^{(\omega)}$	le vecteur des états locaux $x^{(i)}$ tel que $i \in \omega$.

On remarque que la définition d'un état local d'un automate ($x^{(i)}$) et la définition d'un état global ($\tilde{x}$) peuvent être vues comme des cas particuliers de $\tilde{x}^{(\omega)}$. Un état local $x^{(i)}$ est le cas où $\omega = \{i\}$, et l'état global $\tilde{x}$ est le cas où $\omega = \{1, 2, 3, \ldots, N\}$.

✍ **Notation**

$\mathcal{S}^{(\omega)}$	l'espace d'états produit de l'ensemble des états locaux des automates $\mathcal{A}^{(i)}$, où $i \in \omega$;
$f(\mathcal{S}^{(\omega)})$	une fonction de $\mathcal{S}^{(\omega)} \rightarrow \mathbb{R}^{+}$, où l'ensemble d'indices d'automates $\omega \subseteq [1..N]$;
$f(\tilde{x}^{(\omega)})$	la fonction $f(\mathcal{S}^{(\omega)})$ évaluée pour le vecteur d'états locaux $\tilde{x}^{(\omega)} \in \mathcal{S}^{(\omega)}$.

Notons que les états $x^{(i)}$, où $i \in \omega$, sont les paramètres d'évaluation pour la fonction $f(\mathcal{S}^{(\omega)})$, c'est-à-dire que l'espace d'états $\mathcal{S}^{(\omega)}$ est le domaine de définition de la fonction f.

✍ **Notation**

$\mathcal{E}$	l'ensemble d'événements du modèle SAN considéré ;

e	l'identificateur d'un événement, où $e \in \mathcal{E}$.

Définition 6.2.3. *Pour $e \in \mathcal{E}$, on appelle tuple d'événement, le triplet (e, ρ_e, ϱ_e) composé de :*

1. *e, l'identificateur de l'événement ;*
2. *ρ_e, la fonction définie de $\mathcal{S} \rightarrow [0, 1]$, la probabilité d'occurrence de l'événement e ;*
3. *L'occurrence de l'événement e est une variable aléatoire de loi de Bernoulli de paramètre ρ_e. Pour $e \neq e'$ les lois sont indépendantes ;*
4. *ϱ_e, la priorité de l'événement. La priorité est une valeur entière strictement positive, 1 est la priorité maximum et $+\infty$ est la priorité minimum. Deux événements différents ne peuvent pas avoir la même priorité.*

La définition 6.2.3 associe une probabilité d'occurrence ρ_e et une priorité ϱ_e à un événement e. La priorité détermine l'ordre d'occurrence lorsque plusieurs événements sont possibles dans la même unité de temps.

Définition 6.2.4. *Pour $x^{(i)}$ et $y^{(i)} \in \breve{\mathcal{S}}^{(i)}$, on appelle tuple de transition locale un couple $(e, \pi_e(x^{(i)}, y^{(i)}))$:*

1. *e, l'identificateur d'un événement de $\mathcal{E}$;*
2. *$\pi_e(x^{(i)}, y^{(i)})$, une fonction définie de $\mathcal{S} \rightarrow [0, 1]$, qui est appelée la* **probabilité de routage** *de ce tuple.*

✍ **Notation**

$\mathcal{T}$	l'ensemble des tuples de transition locale d'un modèle SAN ;
$2^{\mathcal{T}}$	l'ensemble de parties de $\mathcal{T}$. Une partie de $\mathcal{T}$ sera appelée **élément de transition**.

Définition 6.2.5. *$\mathcal{P}^{(i)}$ est la matrice de transition locale de $\mathcal{S}^{(i)} \times \breve{\mathcal{S}}^{(i)} \rightarrow 2^{\mathcal{T}}$, qui définit les éléments de transition de l'automate $\mathcal{A}^{(i)}$.*

✍ **Notation**

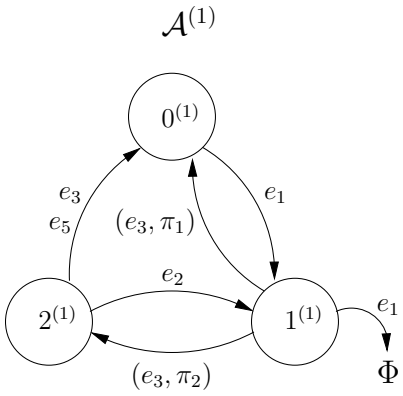
$\mathcal{P}^{(i)}(x^{(i)}, y^{(i)})$	l'élément de transition de l'état local $x^{(i)} \in \mathcal{S}^{(i)}$ vers l'état local $y^{(i)} \in \breve{\mathcal{S}}^{(i)}$, qui contient un sous-ensemble des tuples de transition locale dans $\mathcal{T}$;
$\pi_e(x^{(i)}, y^{(i)})(\tilde{x})$	la probabilité de routage associée au tuple de transition locale $(e, \pi_e(x^{(i)}, y^{(i)}))$ dans $\mathcal{P}^{(i)}(x^{(i)}, y^{(i)})$ évaluée pour l'état global $\tilde{x}$.

La matrice de transition locale $\mathcal{P}^{(i)}$ d'un automate $\mathcal{A}^{(i)}$ (Définition 6.2.5) indique la relation entre les états de l'automate $\mathcal{A}^{(i)}$ et les événements qui peuvent déclencher une transition d'un état à l'autre dans cet automate. La transition peut être déclenchée par n'importe quel événement qui figure dans un tuple de l'élément de transition associé. On remarque que Φ n'est jamais l'état de départ d'une transition.

Définition 6.2.6. *Un automate $\mathcal{A}^{(i)}$ est défini par :*

1. *un ensemble d'états $\breve{\mathcal{S}}^{(i)}$;*
2. *un ensemble d'événements $\mathcal{E}$;*
3. *une matrice de transition locale $\mathcal{P}^{(i)}$.*

Pour représenter graphiquement un automate stochastique, on associe un graphe à l'automate. Les noeuds du graphe représentent les états de l'automate, et les arcs représentent les transitions entre ces états. Les arcs sont étiquetés par un élément de transition où figurent les événements qui peuvent déclencher la transition.



Événement	Prob.	Priorité
e_1	ρ_1	3
e_2	ρ_2	1
e_3	ρ_3	2
e_5	ρ_5	5

FIG. 6.4 – Représentation graphique de l'automate $\mathcal{A}^{(1)}$

FIG. 6.4 décrit un automate avec 3 états plus l'état fantôme ($\breve{\mathcal{S}}^{(1)} = \{0^{(1)}; 1^{(1)}; 2^{(1)}; \Phi\}$). On remarque que lorsque la probabilité de routage est égale à 1.0, alors on remplace sur le dessin $(e_1, 1.0(0^{(1)}, 1^{(1)}))$ par e_1 afin d'alléger la présentation, une fois qu'on connaît l'état de départ et d'arrivée de la transition par le dessin.

Notons qu'il se peut qu'un même événement, en partant du même état, puisse mener dans plusieurs états différents. Pour définir la probabilité d'aller dans chacun des états d'arrivée, on utilise la probabilité de routage du tuple de transition locale associé.

À partir de cette représentation graphique, on peut déduire la matrice de transition locale $\mathcal{P}^{(1)}$ de l'automate $\mathcal{A}^{(1)}$:

$\mathcal{P}^{(1)}$	$0^{(1)}$	$1^{(1)}$	$2^{(1)}$	Φ
$0^{(1)}$	$\emptyset$	$\{(e_1, 1.0)\}$	$\emptyset$	$\emptyset$
$1^{(1)}$	$\{(e_3, \pi_1)\}$	$\emptyset$	$\{(e_3, \pi_2)\}$	$\{(e_1, 1.0)\}$
$2^{(1)}$	$\{(e_3, 1.0); (e_5, 1.0)\}$	$\{(e_2, 1.0)\}$	$\emptyset$	$\emptyset$

TAB. 6.1 – Matrice de transition locale $\mathcal{P}^{(1)}$ de l'automate $\mathcal{A}^{(1)}$

On remarque que les indices $x^{(i)}$ et $y^{(i)}$ associés aux probabilités de routages ont été enlevés de la matrices de transition locale, pour également, alléger la présentation.

Définition 6.2.7. *Soit un état local $x^{(i)} \in \mathcal{S}^{(i)}$ d'un automate $\mathcal{A}^{(i)}$ et un événement $e \in \mathcal{E}$, on définit $succ_e(x^{(i)})$ l'ensemble d'états successeurs $y^{(i)} \in \mathcal{S}^{(i)}$ de $x^{(i)}$. Ces états $y^{(i)}$ sont ceux tels que $\mathcal{P}^{(i)}(x^{(i)}, y^{(i)})$ possède au moins un tuple de transition locale avec l'identificateur e où : $\rho_e \neq 0$ et $\pi_e(x^{(i)}, y^{(i)}) \neq 0$.*

On remarque que l'ensemble d'états successeurs de $x^{(i)}$ grâce à l'événement e peut être vide (*i.e.*, $succ_e(x^{(i)}) = \emptyset$), cas où aucune transition ne peut avoir lieu dans $x^{(i)}$ par l'événement e.

Par exemple, l'ensemble d'états successeurs de chaque état de l'automate $\mathcal{A}^{(1)}$, pour chaque événement $e \in \mathcal{E}$ est :

Événement	Successeurs de $x^{(1)} = 0^{(1)}$	Successeurs de $x^{(1)} = 1^{(1)}$	Successeurs de $x^{(1)} = 2^{(1)}$
$succ_{e_1}(x^{(1)})$	$\{1^{(1)}\}$	$\emptyset$	$\emptyset$
$succ_{e_2}(x^{(1)})$	$\emptyset$	$\emptyset$	$\{1^{(1)}\}$
$succ_{e_3}(x^{(1)})$	$\emptyset$	$\{0^{(1)}, 2^{(1)}\}$	$\{0^{(1)}\}$
$succ_{e_5}(x^{(1)})$	$\emptyset$	$\emptyset$	$\{0^{(1)}\}$

TAB. 6.2 – Ensemble d'états successeurs de l'automate $\mathcal{A}^{(1)}$

Définition 6.2.8. *Soit un état local $x^{(i)} \in \mathcal{S}^{(i)}$ d'un automate $\mathcal{A}^{(i)}$, on défini $pot(x^{(i)})$ l'ensemble d'événements $e \in \mathcal{E}$ tel qu'il existe $y^{(i)} \in \check{\mathcal{S}}^{(i)}$, où $\mathcal{P}^{(i)}(x^{(i)}, y^{(i)})$ possède un tuple de transition locale $(e, \pi_e(x^{(i)}, y^{(i)}))$. On appelle $pot(x^{(i)})$ l'ensemble des événements possibles à partir de $x^{(i)}$.*

La définition 6.2.8 définit l'ensemble d'événements possibles à partir de l'état $x^{(i)}$. On remarque aussi que si $succ_e(x^{(i)})$ n'est pas vide, alors $e \in pot(x^{(i)})$.

TAB. 6.3 présente les ensembles d'événements possibles pour chaque état de l'automate $\mathcal{A}^{(1)}$.

État local	Événements possibles
$pot(0^{(1)})$	$\{e_1\}$
$pot(1^{(1)})$	$\{e_1; e_3\}$
$pot(2^{(1)})$	$\{e_2; e_3; e_5\}$

TAB. 6.3 – Événements possibles à partir de chaque état de l'automate $\mathcal{A}^{(1)}$

Étant donné un état $x^{(i)} \in \mathcal{S}^{(i)}$ et l'ensemble $pot(x^{(i)})$ (Définition 6.2.8), la sémantique probabiliste donnée à cet automate est la suivante :

1. l'événement le plus prioritaire e_1 de $pot(x^{(i)})$ est déclenché à partir de $x^{(i)}$ avec probabilité ρ_{e_1}, ce qui mène à l'état $x_1^{(i)}$;

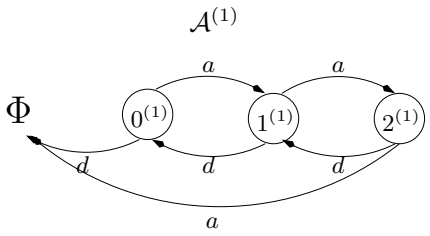
2. si l'événement suivant le plus prioritaire de $pot(x^{(i)})$ **est réalisable** dans $x_1^{(i)}$ (ce qui se voit car il doit être dans $pot(x_1^{(i)})$) alors son occurrence amène à $x_j^{(i)}$ et on répète les étapes 2 et 3 jusqu'à avoir examiné tous les événements de $pot(x^{(i)})$;
3. si l'événement suivant le plus prioritaire de $pot(x^{(i)})$ **n'est pas réalisable** dans $x_1^{(i)}$ (i.e. l'événement n'est pas dans $pot(x_1^{(i)})$) alors on l'ignore et on répète les étapes 2 et 3 jusqu'à avoir examiné tous les événements de $pot(x^{(i)})$.

On remarque que cette sémantique autorise la réalisation de plusieurs événements dans la même unité de temps. Par exemple, pour une file d'attente à temps discret, dans un état où le nombre de client est $n > 0$, une arrivée et un départ de la file peuvent survenir dans la même unité de temps.

Dans l'exemple précédent et pour l'état $1^{(1)}$, l'ensemble d'événements possibles inclut l'événement e_1 associée à la transition vers l'état fantôme Φ. Ceci signifie que e_1 est dans $pot(1^{(1)})$. Cependant, cet événement sera *réalisable* uniquement si l'occurrence de l'événement e_3, plus prioritaire, amène à l'état $0^{(1)}$, où e_1 est réalisable (donc dans $pot(0^{(1)})$) et amène vers un état "vrai" de l'automate (pas l'état fantôme Φ).

Prenons par exemple l'état $1^{(1)}$ et son ensemble $pot(1^{(1)}) = \{e_1; e_3\}$. Lorsque l'événement le plus prioritaire (e_3) a lieu, deux états sont possibles ($succ_{e_3}(1^{(1)})$) : $0^{(1)}$ avec probabilité π_1 et $2^{(1)}$ avec probabilité π_2. Pour l'état successeur $0^{(1)}$, l'événement suivant le plus prioritaire (e_1) est réalisable et mène à l'état $1^{(1)}$. Cette suite d'occurrence ($(e_3, \pi_1), (e_1, 1.0)$) représente une transition de la chaîne de Markov de l'état $1^{(1)}$ vers lui-même. Pour l'état successeur $2^{(1)}$, l'événement suivant le plus prioritaire (e_1) n'est pas réalisable ($e_1 \notin pot(2^{(1)})$), alors on l'ignore. Étant donné qu'il n'y a plus d'événement possible, la suite d'occurrence (formée uniquement par (e_3, π_2)) provoque une transition de la chaîne de Markov de l'état $1^{(1)}$ vers l'état $2^{(1)}$.

Pour une file d'attente, l'état fantôme Φ peut, par exemple, autoriser l'arrivée et le départ d'un client dans la même unité de temps, si $n = 0$. C'est la modélisation qui autorise cet enchaînement. L'exemple de file d'attente de FIG. 6.5 présente cette modélisation.



Événement	Prob.	Priorité
a	ρ_a	ϱ_a
d	ρ_d	ϱ_d

FIG. 6.5 – File d'attente avec arrivée et départ pour $n = 0$

Dans cet exemple, si $\varrho_a < \varrho_d$ (événement a plus prioritaire que l'événement d) à partir de l'état $0^{(1)}$ la réalisation simultanée des événements d (départ tardif) et a est possible. Si $\varrho_a > \varrho_d$ (événement d plus prioritaire que l'événement a) alors à partir de l'état $2^{(1)}$ la réalisation simultanée des événements a (arrivée tardive) et d est possible grâce à l'état fantôme Φ.

Automate complété

Dans la composition parallèle d'automates pour les SAN (voir dans la section suivante) on aura besoin d'exprimer le fait que, si l'événement e_1 de l'automate $\mathcal{A}^{(1)}$ et l'événement e_4 de l'automate $\mathcal{A}^{(2)}$ sont réalisables au même instant, alors 4 possibilités existent : e_1 a lieu et e_4 n'a pas lieu ; e_1 n'a pas lieu et e_4 a lieu ; e_1 a lieu et e_4 a lieu aussi ; et e_1 n'a pas lieu et e_4 n'a pas lieu non plus.

Pour ce faire, nous avons besoin de représenter de façon explicite sur l'automate le résultat de la *non occurrence* d'un événement. Pour cela, on va introduire des événements *factices*, appelés "*événements complémentaires*" pour chaque événement présent dans un automate.

Définition 6.2.9. *À tous les événements e d'un modèle* SAN*, où $e \in \mathcal{E}$ dont le triplet est (e, ρ_e, ϱ_e), on associe un événement complémentaire défini par le triplet $(\bar{e}, \rho_{\bar{e}}, \varrho_{\bar{e}})$ où :*

1. *$\bar{e}$, l'identificateur de l'événement complémentaire associé à l'événement e ;*
2. *$\rho_{\bar{e}}$, la fonction définie par $1 - \rho_e$, représentant la probabilité d'occurrence de l'événement complémentaire $\bar{e}$;*
3. *$\varrho_{\bar{e}}$, la priorité de l'événement complémentaire. La priorité d'un événement complémentaire est égale à la priorité de l'événement auquel l'événement complémentaire est associé (i.e., $\varrho_{\bar{e}} = \varrho_e$).*

✍ **Soit**

$\bar{\mathcal{E}}$	l'ensemble d'événements complémentaires de $\mathcal{E}$;
$\breve{\mathcal{E}}$	l'union des ensembles d'événements $\mathcal{E}$ et $\bar{\mathcal{E}}$ ($\breve{\mathcal{E}} = \mathcal{E} \cup \bar{\mathcal{E}}$).

Définition 6.2.10. *À tous les événements complémentaires $\bar{e}$, où $\bar{e} \in \bar{\mathcal{E}}$, on associe des tuples de transition locale défini par :*

1. $\forall x^{(i)} \in \mathcal{S}^{(i)}$ *tel que* $\exists (e, \pi_e(x^{(i)}, y^{(i)})) \in \mathcal{P}^{(i)}(x^{(i)}, y^{(i)})$ *tel que* $x^{(i)} \neq y^{(i)}$
   $(\bar{e}, \pi_{\bar{e}}(x^{(i)}, x^{(i)}))$
2. $\forall x^{(i)} \in \mathcal{S}^{(i)}$ *tel que* $\exists (e, \pi_e(x^{(i)}, \Phi)) \in \mathcal{P}^{(i)}(x^{(i)}, \Phi)$
   $(\bar{e}, \pi_{\bar{e}}(x^{(i)}, \Phi))$

Tous les tuples de transition locale des événements complémentaires ont comme probabilité de routage 1.0. L'occurrence d'un événement complémentaire amène toujours vers l'état de départ.

✍ **Notation**

$\bar{\mathcal{T}}$	l'ensemble des tuples de transition locale des événements complémentaires de l'ensemble $\bar{\mathcal{E}}$;
$\breve{\mathcal{T}}$	l'union des ensembles des tuples de transition locale $\mathcal{T}$ et $\bar{\mathcal{T}}$ ($\breve{\mathcal{T}} = \mathcal{T} \cup \bar{\mathcal{T}}$) ;
$2^{\breve{\mathcal{T}}}$	l'ensemble des parties de $\breve{\mathcal{T}}$.

Ces événements complémentaires sont utiles pour la composition parallèle dans le SAN. On appelle *automate complété*, l'automate où apparaissent les tuples des événements complémentaires.

Définition 6.2.11. *L'automate complété, noté $\breve{\mathcal{A}}^{(i)}$, d'un automate $\mathcal{A}^{(i)}(\breve{\mathcal{S}}^{(i)}, \mathcal{E}, \mathcal{P}^{(i)})$ est défini par :*

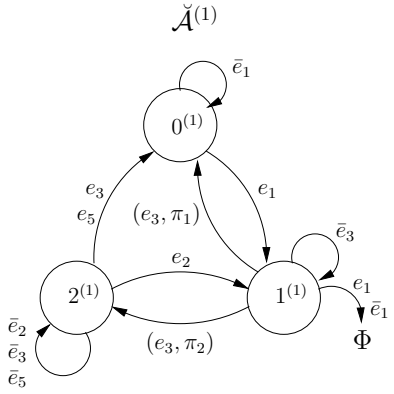
1. *l'ensemble d'états $\breve{\mathcal{S}}^{(i)}$;*
2. *un ensemble d'événements $\breve{\mathcal{E}}$;*
3. *une matrice de transition locale $\breve{\mathcal{P}}^{(i)} : \mathcal{S}^{(i)} \times \breve{\mathcal{S}}^{(i)} \rightarrow 2^{\breve{\mathcal{T}}}$:*

$$\forall x^{(i)} \in \mathcal{S}^{(i)}, \forall y^{(i)} \in \mathcal{S}^{(i)} \text{ tel que } x^{(i)} \neq y^{(i)}$$
$$\breve{\mathcal{P}}^{(i)}(x^{(i)}, y^{(i)}) = \mathcal{P}^{(i)}(x^{(i)}, y^{(i)})$$
$$\forall x^{(i)} \in \mathcal{S}^{(i)}$$
$$\breve{\mathcal{P}}^{(i)}(x^{(i)}, x^{(i)}) = \mathcal{P}^{(i)}(x^{(i)}, x^{(i)}) \bigcup_{\forall y^{(i)} \in \mathcal{S}^{(i)}, \forall e \in \mathcal{E} \text{ tel que } \exists\ (e, \pi_e(x^{(i)}, y^{(i)})) \in \mathcal{P}^{(i)}(x^{(i)}, y^{(i)})} \{(\bar{e}, \pi_{\bar{e}}(x^{(i)}, x^{(i)}))\}$$
$$\forall x^{(i)} \in \mathcal{S}^{(i)}$$
$$\breve{\mathcal{P}}^{(i)}(x^{(i)}, \Phi) = \mathcal{P}^{(i)}(x^{(i)}, \Phi) \bigcup_{\forall e \in \mathcal{E} \text{ tel que } \exists\ (e, \pi_e(x^{(i)}, \Phi)) \in \mathcal{P}^{(i)}(x^{(i)}, \Phi)} \{(\bar{e}, \pi_{\bar{e}}(x^{(i)}, \Phi))\}$$

L'automate complété $\breve{\mathcal{A}}^{(i)}$ est défini sur le même ensemble d'états $\breve{\mathcal{S}}^{(i)}$ de son automate $\mathcal{A}^{(i)}$. L'ensemble d'événements $\breve{\mathcal{E}}$ de l'automate complété contient les événements du modèle SAN et aussi les événements complémentaires ajoutés au modèle. La définition de la matrice de transition locale $\breve{\mathcal{P}}^{(i)}$ de l'automate complété est présentée dans la troisième règle de la définition 6.2.11. Aux éléments diagonaux sont ajoutés les tuples de transition locale des événements complémentaires. Un tuple de transition locale d'un événement complémentaire $\bar{e}$ est ajouté à $\breve{\mathcal{P}}^{(i)}(x^{(i)}, x^{(i)})$ uniquement s'il existe une transition d'un $x^{(i)}$ à un état $y^{(i)}$ ($y^{(i)}$ différent de l'état fantôme Φ) qui peut être déclenchée par un tuple de transition locale avec un identificateur de l'événement e. De façon similaire, un tuple de transition locale d'un événement complémentaire $\bar{e}$ est ajouté à $\breve{\mathcal{P}}^{(i)}(x^{(i)}, \Phi)$, pour chaque tuple de transition locale associé à $\mathcal{P}^{(i)}(x^{(i)}, \Phi)$. Les tuples de transition locale des événements complémentaires associés à des tuples de transition locale qui amène à l'état fantôme Φ sont associés à la même transition, c'est-à-dire, à la transition vers l'état fantôme Φ. Cette représentation rend l'événement complémentaire possible sans qui soit réalisable à partir de $x^{(i)}$.

Prenons par exemple l'automate $\mathcal{A}^{(1)}$ de FIG. 6.4. À partir de l'état $1^{(1)}$, les événements e_1 et e_3 sont possibles, cependant uniquement l'événement e_3 est réalisable à partir de cet état. L'événement e_1 sera réalisable uniquement si l'occurrence de l'événement e_3 a amené à l'état $0^{(1)}$. Pour garder cette sémantique, l'événement complémentaire $\bar{e}_1$ peut avoir lieu uniquement si l'événement e_3 (plus prioritaire) a eu lieu d'abord et a amené à l'état $0^{(1)}$. Par conséquence, l'événement complémentaire $\bar{e}_1$ doit être associé à la transition de l'état $1^{(1)}$ vers l'état fantôme Φ. De cette façon, l'événement complémentaire $\bar{e}_1$ est possible à partir de l'état $1^{(1)}$ mais n'est pas réalisable.

FIG. 6.6 montre la représentation graphique de l'automate complété $\breve{\mathcal{P}}^{(1)}$ de l'automate $\mathcal{P}^{(1)}$ présenté dans FIG. 6.4.



Événement	Prob.	Priorité
e_1	ρ_1	3
e_2	ρ_2	1
e_3	ρ_3	2
e_5	ρ_5	5

Événement	Prob.	Priorité
$\bar{e}_1$	$1-\rho_1$	3
$\bar{e}_2$	$1-\rho_2$	1
$\bar{e}_3$	$1-\rho_3$	2
$\bar{e}_5$	$1-\rho_5$	5

FIG. 6.6 – Représentation graphique de l'automate complété $\breve{\mathcal{A}}^{(1)}$ de l'automate $\mathcal{A}^{(1)}$

La matrice de transition locale $\breve{\mathcal{P}}^{(1)}$ de l'automate complété $\breve{\mathcal{A}}^{(1)}$ de FIG. 6.6 est :

$\breve{\mathcal{P}}^{(1)}$	$0^{(1)}$	$1^{(1)}$	$2^{(1)}$	Φ
$0^{(1)}$	$\{(\bar{e}_1, 1.0)\}$	$\{(e_1, 1.0)\}$	$\emptyset$	$\emptyset$
$1^{(1)}$	$\{(e_3, \pi_1)\}$	$\{(\bar{e}_3, 1.0)\}$	$\{(e_3, \pi_2)\}$	$\{(e_1, 1.0); (\bar{e}_1, 1.0)\}$
$2^{(1)}$	$\{(e_3, 1.0); (e_5, 1.0)\}$	$\{(e_2, 1.0)\}$	$\{(\bar{e}_2, 1.0); (\bar{e}_3, 1.0); (\bar{e}_5, 1.0)\}$	$\emptyset$

TAB. 6.4 – Matrice de transition locale de l'automate complété $\breve{\mathcal{A}}^{(1)}$

À partir de la matrice de transition locale de l'automate complété (Définition 6.2.11) et de l'ensemble d'événements possibles à partir de chaque état, on peut construire la chaîne de Markov représentée par l'automate complété.

Pour un seul automate ($i = 1$, réseau réduit à un automate), les états "vrai" de l'automate représentent les états de la chaîne de Markov[3] tandis que les probabilités de transition sont calculées par un calcul probabiliste des occurrences des événements possibles, sachant que l'hypothèse de base est que tous ces événements ont des lois de probabilités de Bernoulli représentant des variables aléatoires indépendantes (Définition 6.2.3). La définition de la chaîne de Markov pour un réseau d'automates est détaillée dans la section 6.3. Cependant, les principes de base présentés informellement ci-après, pour la construction de la chaîne de Markov d'un seul automate, sont les mêmes que pour un réseau d'automates.

Le calcul de la probabilité d'occurrence de chaque transition de la chaîne de Markov est fait de la façon suivante :

- Pour chaque état "vrai" de l'automate ($x^{(i)} \in \mathcal{S}^{(i)}$), on définit l'ensemble d'événements possibles ($pot(x^{(i)})$) ;
- Pour chaque événement e de $pot(x^{(i)})$, il existe dans $\mathcal{P}^{(i)}$ des tuples de transition locale. Ces tuples de transition locale peuvent être réécrits sous forme de *chaînons de transi-*

[3]Pour un réseau d'automates, les états de la chaîne de Markov est le produit cartésien des espaces d'états de chaque automate du modèle.

tion globale (voir définition formelle dans Définition 6.3.1) de la forme suivante : si $\mathcal{P}^{(i)}(x^{(i)}, y^{(i)})$ contient le tuple $(e, \pi_e(x^{(i)}, y^{(i)}))$ alors $(x^{(i)}, y^{(i)}, e, \pi_e(x^{(i)}, y^{(i)}))$ est appelé *chaînon* de $\mathcal{A}^{(i)}$.

– La sémantique que l'on donne au comportement markovien de l'automate est la suivante : pour que les événements $\{e_1, e_2, \ldots, e_C\} \in pot(x^{(i)})$ avec $\varrho_1 < \varrho_2 < \ldots < \varrho_C$ soient réalisables en **1 unité de temps** il faut qu'il existe une suite de chaînons tel que :
  – Chaque événement n'apparaît qu'une fois ;
  – L'état d'arrivée d'un chaînon est égal à l'état de départ du chaînon suivant. Il faut que les chaînons d'une liste de chaînons s'enchaînent de façon à forme une chaîne. Ces listes de chaînons de transition globale sont appelées *chaînes de transitions globales*. Les règles qui définissent une chaîne de transitions globales sont définies formellement dans Définition 6.3.2.

On va maintenant illustrer la construction de la chaîne de Markov de l'automate complété $\breve{\mathcal{A}}^{(1)}$.

Prenons par exemple l'état $0^{(1)}$. À partir de cet état, deux événements sont possibles ($pot(0^{(1)}) = \{e_1; \bar{e}_1\}$). Pour cet ensemble d'événements, on peut construire les chaînons de transition globale à partir de tuples de transition locale de l'automate. TAB. 6.5 présente les tuples de transition locale et les chaînons obtenus à partir de chaque tuple de transition locale.

Événement	Tuple de trans. locale	Chaînon de trans. globale	Événement	Tuple de trans. locale	Chaînon de trans. globale
e_1	$(e_1, 1.0)$	$(0^{(1)}, 1^{(1)}, e_1, 1.0)$	$\bar{e}_1$	$(\bar{e}_1, 1.0)$	$(0^{(1)}, 0^{(1)}, \bar{e}_1, 1.0)$

TAB. 6.5 – Ensemble de chaînons de transition globale pour l'ensemble d'événements de $pot(0^{(1)})$

À partir de ces chaînons, on peut construire deux chaînes de transitions globales : $\{(0^{(1)}, 1^{(1)}, e_1, 1.0)\}$ et $\{(0^{(1)}, 0^{(1)}, \bar{e}_1, 1.0)\}$. La chaîne de transitions globales $\{(0^{(1)}, 1^{(1)}, e_1, 1.0)\}$ contient un seul chaînon qui représente la transition de l'état $0^{(1)}$ vers l'état $1^{(1)}$ déclenchée par l'événement e_1. La probabilité d'occurrence de transition de l'état $0^{(1)}$ vers l'état $1^{(1)}$ de la chaîne de Markov est simplement la probabilité d'occurrence (ρ_1) de l'événement e_1. La chaîne de transitions globales $\{(0^{(1)}, 0^{(1)}, \bar{e}_1, 1.0)\}$ représente une transition de l'état $0^{(1)}$ vers lui-même. Cette chaîne aussi contient un seul chaînon et la probabilité de transition de la chaîne de Markov est égale à $1-\rho_1$, c'est-à-dire, la probabilité d'occurrence de l'événement $\bar{e}_1$. FIG. 6.7 présente les chemins de construction de ces deux chaînes.

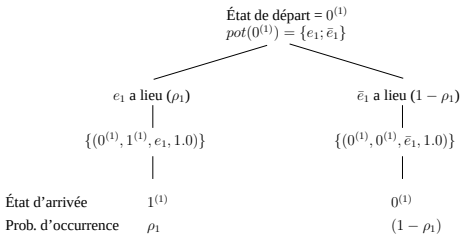


FIG. 6.7 – Construction des chaînes de transitions globales pour l'état $0^{(1)}$ de l'automate $\mathcal{A}^{(1)}$

Prenons maintenant l'état $1^{(1)}$ de l'automate complété $\breve{\mathcal{A}}^{(1)}$. À partir de cet état, quatre événements sont possibles ($pot(1^{(1)}) = \{e_3; \bar{e}_3; e_1; \bar{e}_1\}$). Pour cet ensemble d'événements, on a l'ensemble de chaînons suivant :

Événement	Tuple de trans. locale	Chaînon de trans. globale	Événement	Tuple de trans. locale	Chaînon de trans. globale
e_3	(e_3, π_1) (e_3, π_2) $(e_3, 1.0)$	$(1^{(1)}, 0^{(1)}, e_3, \pi_1)$ $(1^{(1)}, 2^{(1)}, e_3, \pi_2)$ $(2^{(1)}, 0^{(1)}, e_3, 1.0)$	$\bar{e}_3$	$(\bar{e}_3, 1.0)$ $(\bar{e}_3, 1.0)$	$(1^{(1)}, 1^{(1)}, \bar{e}_3, 1.0)$ $(2^{(1)}, 2^{(1)}, \bar{e}_3, 1.0)$
e_1	$(e_1, 1.0)$	$(0^{(1)}, 1^{(1)}, e_1, 1.0)$	$\bar{e}_1$	$(\bar{e}_1, 1.0)$	$(0^{(1)}, 0^{(1)}, \bar{e}_1, 1.0)$

TAB. 6.6 – Ensemble de chaînons de transition globale pour l'ensemble d'événements de $pot(1^{(1)})$

Pour cet ensemble de chaînons, quatre chaînes de transitions globales sont possibles : $\{(1^{(1)}, 2^{(1)}, e_3, \pi_2)\}$, $\{(1^{(1)}, 0^{(1)}, e_3, \pi_1); (0^{(1)}, 0^{(1)}, \bar{e}_1, 1.0)\}$, $\{(1^{(1)}, 0^{(1)}, e_3, \pi_1); (0^{(1)}, 1^{(1)}, e_1, 1.0)\}$ et $\{(1^{(1)}, 1^{(1)}, \bar{e}_3, 1.0)\}$. Ces chaînes de transitions globales sont des listes de chaînons de transition globale construites de façon à ce que chaque chaînon de transition globale soit enchaîné au chaînon suivant. FIG. 6.8 présente les chemins de construction de ces quatre chaînes de transitions globales.

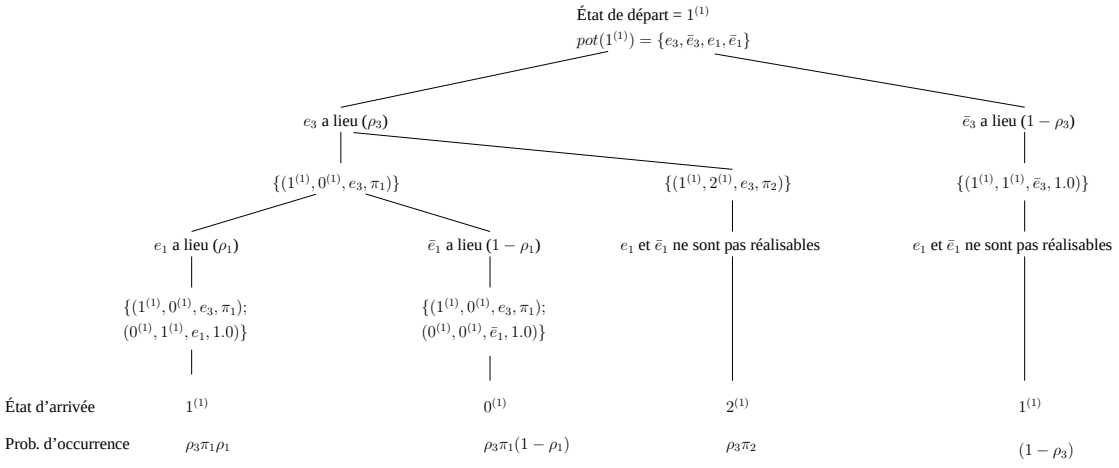


FIG. 6.8 – Construction des chaînes de transitions globales pour l'état $1^{(1)}$ de l'automate $\mathcal{A}^{(1)}$

On va regarder en détail chaque chaîne de transitions globales de cet état.

On va commencer par deuxième chaîne de FIG. 6.8 : $\{(1^{(1)}, 0^{(1)}, e_3, \pi_1); (0^{(1)}, 0^{(1)}, \bar{e}_1, 1.0)\}$. Cette chaîne de transitions globales va de l'état $1^{(1)}$ vers l'état $0^{(1)}$ de la chaîne de Markov. La probabilité d'occurrence calculée en multipliant la probabilité d'occurrence (ρ_3) de l'événement e_3 du premier chaînon de la chaîne par la probabilité de routage du chaînon (π_1) et par la probabilité d'occurrence ($1 - \rho_1$) de l'événement $\bar{e}_1$ du deuxième chaînon de la chaîne. La transition de l'état $1^{(1)}$ vers l'état $0^{(1)}$ de la chaîne de Markov a comme probabilité $\rho_3\pi_1(1-\rho_1)$. On remarque que pour cette chaîne de transitions globales, on a une séquence d'événements qui sont réalisables. La probabilité d'occurrence de la transition de la chaîne de Markov doit

prendre en compte la probabilité d'occurrence de tous les événements de la chaîne de transitions globales.

La chaîne de transitions globales $\{(1^{(1)}, 2^{(1)}, e_3, \pi_2)\}$, troisième chaîne dans FIG. 6.8 contient un seul chaînon pour une transition de l'état $1^{(1)}$ vers l'état $2^{(1)}$ de la chaîne de Markov. La probabilité d'occurrence de cette transition est calculée en multipliant la probabilité d'occurrence (ρ_3) de l'événement e_3 qui déclenche la transition par probabilité de routage π_2 du chaînon. La probabilité d'occurrence est égale à $\rho_3\pi_2$.

Prenons maintenant la première et la dernière chaîne de transitions globales. On remarque que ces deux chaînes arrivent au même état. Cela représente deux possibilités pour une seule transition, et donc, la probabilité d'occurrence de cette transition dans la chaîne de Markov est la somme des probabilités d'occurrence de chaque chaîne de transitions globales. Calculons maintenant la probabilité d'occurrence de chaque chaîne de transitions globales. La probabilité d'occurrence de la chaîne de transitions globales $\{(1^{(1)}, 0^{(1)}, e_3, \pi_1); (0^{(1)}, 1^{(1)}, e_1, 1.0)\}$ est calculée en multipliant la probabilité d'occurrence ρ_3 de l'événement e_3 par la probabilité de routage π_1 du premier chaînon et par la probabilité d'occurrence ρ_1 de l'événement e_1 du deuxième chaînon. La deuxième chaîne de transitions globales $\{(1^{(1)}, 1^{(1)}, \bar{e}_3, 1.0)\}$ est composée d'un seul chaînon, donc la probabilité de transition est simplement la probabilité d'occurrence $(1-\rho_3)$ de l'événement $\bar{e}_3$. La probabilité d'occurrence de l'état $1^{(1)}$ vers lui-même de la chaîne de Markov est $\rho_3\pi_1\rho_1 + (1-\rho_3)$.

À partir de l'état $2^{(1)}$, six événements sont possibles ($pot(2^{(1)}) = \{e_2; \bar{e}_2; e_3; \bar{e}_3; e_5; \bar{e}_5\}$). Pour cet ensemble d'événements, on peut construire les chaînons de transitions globales à partir de tuples de transition locale de l'automate. TAB. 6.7 présente les tuples de transition locale et les chaînons obtenus à partir de chaque tuple de transition locale.

Événement	Tuple de trans. locale	Chaînon de trans. globale	Événement	Tuple de trans. locale	Chaînon de trans. globale
e_2	$(e_2, 1.0)$	$(2^{(1)}, 1^{(1)}, e_2, 1.0)$	$\bar{e}_2$	$(\bar{e}_2, 1.0)$	$(2^{(1)}, 2^{(1)}, \bar{e}_2, 1.0)$
e_3	(e_3, π_1)	$(1^{(1)}, 0^{(1)}, e_3, \pi_1)$	$\bar{e}_3$	$(\bar{e}_3, 1.0)$	$(1^{(1)}, 1^{(1)}, \bar{e}_3, 1.0)$
	(e_3, π_2)	$(1^{(1)}, 2^{(1)}, e_3, \pi_2)$		$(\bar{e}_3, 1.0)$	$(2^{(1)}, 2^{(1)}, \bar{e}_3, 1.0)$
	$(e_3, 1.0)$	$(2^{(1)}, 0^{(1)}, e_3, 1.0)$			
e_5	$(e_5, 1.0)$	$(2^{(1)}, 0^{(1)}, e_5, 1.0)$	$\bar{e}_5$	$(\bar{e}_5, 1.0)$	$(2^{(1)}, 2^{(1)}, \bar{e}_5, 1.0)$

TAB. 6.7 – Ensemble de chaînons de transition globale pour l'ensemble d'événements de $pot(2^{(1)})$

À partir de cet ensemble de chaînons, on peut construire sept chaînes de transitions globales sortant de l'état $2^{(1)}$. La construction de ces chaînes de transitions globales est présentée dans FIG. 6.9.

Le calcul des probabilités de transition pour l'état $2^{(1)}$ suit la même procédure. La définition formelle de la chaîne de Markov représentée par le modèle SAN sera présentée dans la section 6.4.

FIG. 6.10 présente la chaîne de Markov représentée par l'automate complété $\breve{\mathcal{A}}^{(1)}$.

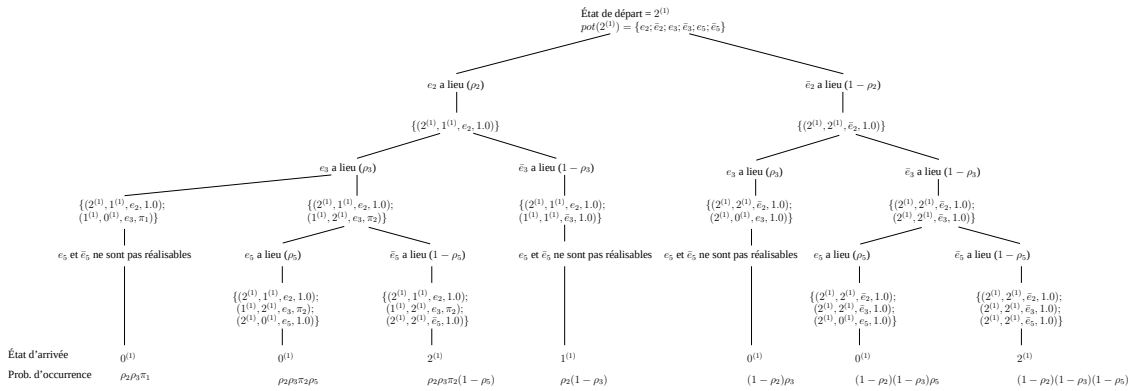


FIG. 6.9 – Construction des chaînes de transitions globales pour l'état $2^{(1)}$ de l'automate $\mathcal{A}^{(1)}$

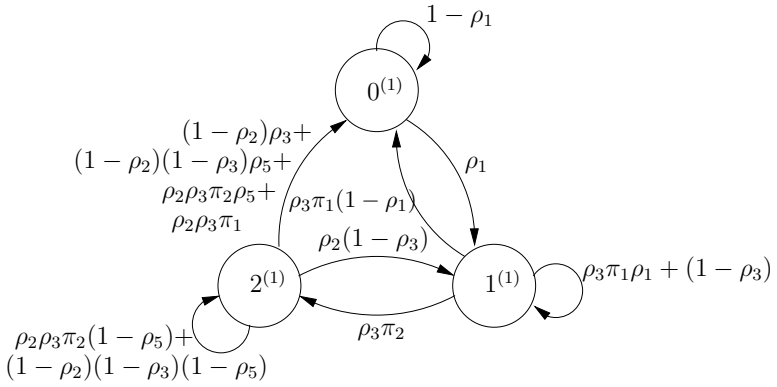
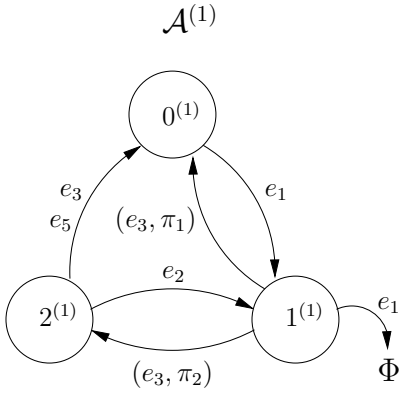
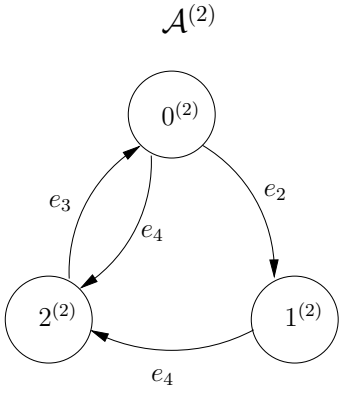


FIG. 6.10 – Chaîne de Markov représentée par l'automate $\mathcal{A}^{(1)}$

6.2.2 Définitions et notations de base pour un réseau d'automates

Soit un ensemble d'automates $\mathcal{A}^{(i)}$ où $i \in [1..N]$, selon la définition 6.2.6. On va utiliser le modèle SAN décrit par FIG. 6.11 pour illustrer les définitions qui suivent.





Événement	Prob.	Priorité
e_1	ρ_1	3
e_2	ρ_2	1
e_3	ρ_3	2
e_4	ρ_4	4
e_5	f	5

$f = (st\ \mathcal{A}^{(2)} == 2^{(2)}) * \rho_5$

FIG. 6.11 – Dessin d'un réseau d'automates

Définition 6.2.12. *Un réseau d'automates stochastiques (où modèle* SAN*) composé de N automates et E événements est défini par :*

1. *un ensemble d'événements $e \in \mathcal{E}$ commun à tous les automates, chaque événement avec son tuple d'événement (e, ρ_e, ϱ_e) et $|\mathcal{E}| = E$;*
2. *chacun des automates $\mathcal{A}^{(i)}$ ($i \in [1..N]$), son espace d'état $\breve{\mathcal{S}}$ et sa matrice de transition locale $\mathcal{P}^{(i)}$;*
3. *l'espace d'état produit $\mathcal{S} = \prod \mathcal{S}^{(i)}$;*
4. *la fonction d'atteignabilité $\mathcal{F}$, qui définit l'ensemble d'états atteignables du modèle* SAN *dans $\mathcal{S}$ (voir Définition 6.2.13).*

Définition 6.2.13. *La fonction d'atteignabilité $\mathcal{F}$ est une fonction définie de $\mathcal{S} \rightarrow 0, 1$. La fonction associe aux états globaux $\tilde{x} \in \mathcal{S}$ la valeur 1 s'ils sont* **atteignables** *et la valeur 0 s'ils sont* **non-atteignables**.

Définition 6.2.14. *L'espace d'états atteignables $\mathcal{R}$ est le sous-ensemble de $\mathcal{S}$ ($\mathcal{R} \subseteq \mathcal{S}$) composé de tous les états globaux $\tilde{x} \in \mathcal{S}$ tels que $\mathcal{F}(\tilde{x}) = 1$.*

La fonction d'atteignabilité $\mathcal{F}$ s'évalue pour tous les états globaux $\tilde{x} \in \mathcal{S}$ d'un modèle SAN et ainsi détermine quels sont les états atteignables de ce modèle.

✍ **Soit** : Étant donné $e \in \mathcal{E}$,

$\mathcal{O}_e$ l'ensemble d'indices i ($i \in [1..N]$) tel que la matrice de transition locale $\mathcal{P}^{(i)}$ contienne au moins un tuple de transition locale avec l'identificateur de l'événement e.

Définition 6.2.15. *Un événement $e \in \mathcal{E}$ est appelé :*

1. *événement local, si $\mid \mathcal{O}_e \mid = 1$;*
2. *événement synchronisant, si $\mid \mathcal{O}_e \mid > 1$.*

La définition 6.2.15 classe chaque événement qui peut être un *événement local* ou un *événement synchronisant*. Par définition un événement synchronisant est réalisable uniquement s'il est réalisable au même instant dans tous les automates concernés par l'événement. Le déclenchement d'un événement synchronisant change, de façon synchronisée, l'état local de tous les automates concernés par l'événement au même instant.

Pour le modèle SAN décrit par FIG. 6.11, on peut classer les événements ainsi :

Réseau d'automates complétés

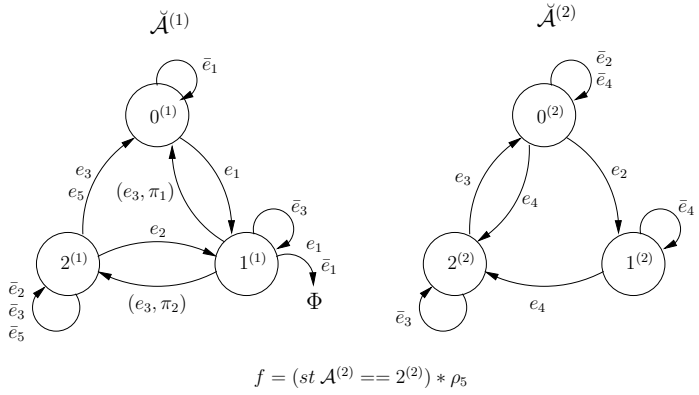
Définition 6.2.16. *Un réseau d'automates stochastiques complétés composé de N automates complétés et $2E$ événements est défini par :*

Événement	Type	$\mathcal{O}_e$
e_1	local	$\{1\}$
e_2	synchronisant	$\{1;2\}$
e_3	synchronisant	$\{1;2\}$
e_4	local	$\{2\}$
e_5	local	$\{1\}$

TAB. 6.8 – Classement des événements du modèle SAN de FIG. 6.11

1. *un ensemble d'événements $e \in \breve{\mathcal{E}}$ commun à tous les automates, chacun avec son tuple d'événement (e, ρ_e, ϱ_e) $|\breve{\mathcal{E}}| = 2E$;*
2. *chacun des automates complétés $\breve{\mathcal{A}}^{(i)}$ ($i \in [1..N]$), son espace d'état $\breve{\mathcal{S}}$ et sa matrice de transition locale $\breve{\mathcal{P}}^{(i)}$;*
3. *l'espace d'état produit $\mathcal{S} = \prod \mathcal{S}^{(i)}$;*
4. *la fonction d'atteignabilité $\mathcal{F}$, qui définit l'ensemble des états atteignables du* SAN *dans $\mathcal{S}$.*

FIG. 6.12 présente le réseau d'automates complétés déduit du réseau d'automates présenté dans FIG. 6.11.



Événement	Prob.	Priorité
e_1	ρ_1	3
e_2	ρ_2	1
e_3	ρ_3	2
e_4	ρ_4	4
e_5	f	5

Événement	Prob.	Priorité
$\bar{e}_1$	$1-\rho_1$	3
$\bar{e}_2$	$1-\rho_2$	1
$\bar{e}_3$	$1-\rho_3$	2
$\bar{e}_4$	$1-\rho_4$	4
$\bar{e}_5$	$1-f$	5

FIG. 6.12 – Représentation graphique d'un réseau d'automates complétés du modèle SAN de FIG. 6.11

6.2.3 SAN *bien définis*

Quelques restrictions doivent être faites pour assurer le cadre Markovien d'un modèle SAN. Les modèles SAN respectant ces restrictions sont appelés SAN *bien définis*.

Restriction 1 *Un automate $\mathcal{A}^{(i)}$ est bien défini, si et seulement si pour tout $\tilde{x} \in \mathcal{R}$, pour tout $x^{(i)} \in \mathcal{S}^{(i)}$, pour tout $e \in \mathcal{E}$ tel que $succ_e(x^{(i)})$ n'est pas vide et pour $y^{(i)} \in succ_e(x^{(i)})$:*

1.1. $$\left(\sum_{y^{(i)} \in succ_e(x^{(i)})} \pi_e(x^{(i)}, y^{(i)})(\tilde{x}) \right) = 1$$

La restriction 1.1 impose que la somme des probabilités de routage de toutes les transitions en sortant d'un même état doit être égale à 1.

Restriction 2 *Un événement $e \in \mathcal{E}$ est bien défini, si et seulement si :*

2.1. $\forall x^{(i)}, \forall y^{(i)} \in \mathcal{S}^{(i)}$ *tel que* $y^{(i)} \in succ_e(x^{(i)})$ *et* $(e, \pi_e(x^{(i)}, y^{(i)})) \in \mathcal{P}^{(i)}(x^{(i)}, y^{(i)})$ *si* $\exists (e_1, \pi_{e_1}(x^{(i)}, y^{(i)})) \in \mathcal{P}^{(i)}(x^{(i)}, y^{(i)})$ *alors* e_1 *est un événement différent de* e.

2.2. $\forall x^{(i)}, \forall y^{(i)} \in \mathcal{S}^{(i)}$ *tel que* $y^{(i)} \in succ_e(x^{(i)})$ *et* $(e, \pi_e(x^{(i)}, y^{(i)})) \in \mathcal{P}^{(i)}(x^{(i)}, y^{(i)})$ *si* $\exists (e_1, \pi_{e_1}(x^{(i)}, y^{(i)})) \in \mathcal{P}^{(i)}(x^{(i)}, \Phi)$ *alors* e_1 *est un événement différent de* e.

La restriction 2.1 imposée aux événements exige que l'identificateur d'un événement apparaisse *une seule fois* dans le sous-ensemble de tuples de transition locale d'un état donné vers un autre. La restriction 2.2 impose qu'il ne doit pas exister un tuple de transition locale $(e, \pi_e(x^{(i)}, y^{(i)}))$ à partir d'un état donné $x^{(i)} \in \mathcal{S}^{(i)}$ vers l'état fantôme Φ, s'il existe déjà un tuple de transition locale de $x^{(i)}$ vers un état $y^{(i)} \in \mathcal{S}^{(i)}$ avec le même événement e.

Restriction 3 *Un modèle* SAN *est bien défini, si et seulement si :*

3.1. tous ses automates sont bien définis ;

3.2. tous ses événements sont bien définis.

6.3 L'automate global

L'objectif d'un modèle SAN est de décrire de façon modulaire un automate unique (dit *automate global*) qui décrit le comportement de l'ensemble du réseau d'automates. Dans cette section nous procédons à cette définition.

Dans cette section, on va considérer un modèle SAN tel que défini dans la Définition 6.2.16.

6.3.1 Transition globale

Définition 6.3.1. *Étant donné un* SAN *complété* $(\breve{\mathcal{E}}, \breve{\mathcal{A}}^{(i)}, \mathcal{F})$*, où* $i \in [1..N]$*, on associe pour tout* $\tilde{x} \in \mathcal{S}$*, pour tout* $\tilde{y} \in \mathcal{S}$ *et pour tout* $e \in \breve{\mathcal{E}}$ *tel que pour tout* $i \in \mathcal{O}_e$ *il existe* $(e, \pi_e(x^{(i)}, y^{(i)})) \in \breve{\mathcal{P}}(x^{(i)}, y^{(i)})$*, un* **chaînon de transition globale** $(\tilde{x}, \tilde{y}, e, \Pi_e(\tilde{x}, \tilde{y}))$ *où :*

1. $\tilde{x}$*, l'état global de départ ;*
2. $\tilde{y}$*, l'état global d'arrivée ;*
3. e*, l'identificateur d'un événement qui déclenche la transition ;*

4. $\Pi_e(\tilde{x}, \tilde{y})$, *la probabilité de routage globale définie par :*

$$\Pi_e(\tilde{x}, \tilde{y}) = \prod_{i \in \mathcal{O}_e, \exists (e, \pi_e(x^{(i)}, y^{(i)})) \in \mathcal{P}(x^{(i)}, y^{(i)})} \pi_e(x^{(i)}, y^{(i)}).$$

Un chaînon de transition globale représente une transition de l'état global $\tilde{x}$ vers l'état global $\tilde{y}$ par le déclenchement uniquement de l'événement e avec une probabilité de routage globale $\Pi_e(\tilde{x}, \tilde{y})$ calculée par le produit des probabilités de routage de tous les automates concernés par l'événement e vers l'état d'arrivée $y^{(i)}$ quand $i \in \mathcal{O}_e$.

✍ **Notation**

$\tilde{\mathcal{T}}_e$	l'ensemble de chaînons de transition globale déclenchés par l'événement e ;
ϵ	un sous-ensemble d'événements de $\breve{\mathcal{E}}$;
$\tilde{\mathcal{T}}_\epsilon$	l'union des $\tilde{\mathcal{T}}_e$ où $e \in \epsilon$ ($\tilde{\mathcal{T}}_\epsilon = \bigcup_{e \in \epsilon} \tilde{\mathcal{T}}_e$).

6.3.2 Chaîne de transitions globales

Définition 6.3.2. *Soit un ensemble d'évènements ϵ, on appelle* **chaîne de transitions globales**, *une liste ordonnée $\{(\tilde{x}_1, \tilde{y}_1, e_1, \Pi_{e_1}(\tilde{x}_1, \tilde{y}_1)), \ldots, (\tilde{x}_C, \tilde{y}_C, e_C, \Pi_{e_C}(\tilde{x}_C, \tilde{y}_C))\}$ composée des C chaînons de transition globale de $\tilde{\mathcal{T}}_\epsilon$, qui respecte les règles suivantes :*

1. $\varrho_{e_1} < \varrho_{e_2} < \ldots < \varrho_{e_C}$
2. $\forall i \in [2..C]$
   $\tilde{x}_i = \tilde{y}_{i-1}$
3. $\nexists (\tilde{x}, \tilde{y}, e, \Pi_e(\tilde{x}, \tilde{y})) \in \tilde{\mathcal{T}}_\epsilon$ *tel que*
   $\varrho_e < \varrho_{e_1}$ *et* $\tilde{y} = \tilde{x}_1$
4. $\nexists (\tilde{x}, \tilde{y}, e, \Pi_e(\tilde{x}, \tilde{y})) \in \tilde{\mathcal{T}}_\epsilon$ *tel que*
   $\varrho_{e_C} < \varrho_e$ *et* $\tilde{x} = \tilde{y}_C$
5. $\forall i \in [1..C-1], \nexists (\tilde{x}, \tilde{y}, e, \Pi_e(\tilde{x}, \tilde{y})) \in \tilde{\mathcal{T}}_\epsilon$ *tel que*
   $\varrho_{e_i} < \varrho_e < \varrho_{e_{i+1}}$ *et* $\tilde{y}_i = \tilde{x}$

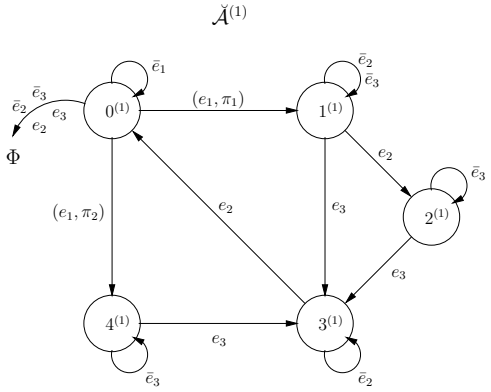
La définition 6.3.2 établit un ensemble de règles pour qu'une liste de chaînons de transition globale soit considérée une *chaîne de transitions globales*. La règle 1 définit l'ordre des chaînons de transition globale dans la chaîne, respectant l'ordre de priorité des événements dans la chaîne. La règle 2 garantit l'enchaînement de tous les chaînons. Cette règle assure que l'état d'arrivée $\tilde{y}_i$ d'un chaînon d'indice i sera l'état de départ $\tilde{x}_{i+1}$ du chaînon suivant (d'indice $i+1$). On se restreint uniquement aux chaînes les plus longues qui respectent les règles 1 et 2. Autrement dit, aucun autre chaînon de transition globale ne peut être enchaîné, soit au début, soit au mieux, soit à la fin de la chaîne. Les 3 dernières règles assurent cette condition. La règle 3 garantit qu'aucun chaînon de transition globale ne peut être enchaîné au début de la chaîne.

La règle 4 assure qu'aucun chaînon de transition globale ne peut être ajouté à la fin de la chaîne. La règle 5 garantit qu'aucun chaînon de transition globale ne peut être enchaîné entre deux chaînons qui se suivent.

✍ **Notation**

$\tilde{x}_{i_g}$	l'état de départ du i-ème chaînon de transition globale de la chaîne de transitions globales g ;
$\tilde{y}_{i_g}$	l'état d'arrivée du i-ème chaînon de transition globale de la chaîne de transitions globales g ;
$\mathcal{L}_\epsilon$	l'ensemble des chaînes de transitions globales de l'ensemble d'événements ϵ ;
$\mathcal{L}(\tilde{x}, \tilde{y})$	l'ensemble de chaînes de transitions globales $g \in \mathcal{L}_{pot(\tilde{x})}$ tel que $\tilde{x}_{1_g} = \tilde{x}$ et $\tilde{y}_{C_g} = \tilde{y}$ où C_g est le nombre de chaînons de transition globale de la chaîne de transitions globales g ;
$\mathcal{L}$	l'ensemble des chaînes de transitions globales du modèle SAN.

Pour illustrer les chaînes de transitions globales qui nous intéressent, on va considérer l'automate complété $\breve{\mathcal{A}}^{(1)}$ de FIG. 6.13 et un ensemble d'événements $\epsilon = \{e_1; \bar{e}_1; e_2; \bar{e}_2; e_3; \bar{e}_3\}$. Cet ensemble d'événements correspond au $\epsilon = pot(0^{(1)}) = \{e_1; \bar{e}_1; e_2; \bar{e}_2; e_3; \bar{e}_3\}$ (Définition 6.3.3).



Événement	Prob.	Priorité
e_1	ρ_1	1
e_2	ρ_2	2
e_3	ρ_3	3

Événement	Prob.	Priorité
$\bar{e}_1$	$1-\rho_1$	1
$\bar{e}_2$	$1-\rho_2$	2
$\bar{e}_3$	$1-\rho_3$	3

FIG. 6.13 – Exemple d'un automate complété $\breve{\mathcal{A}}^{(1)}$

Pour l'ensemble d'événement ϵ et l'automate complété $\breve{\mathcal{A}}^{(1)}$ de FIG. 6.13, TAB. 6.9 présente l'ensemble des chaînons de transition globale $\tilde{\mathcal{T}}_\epsilon$ du modèle. Pour simplifier la lecture, on va représenter, par exemple, un chaînon $(0^{(1)}, 1^{(1)}, e_1, \pi_1)$ par $(0, 1, e_1, \pi_1)$.

Étant donné le grand nombre de chaînes possibles à partir des chaînons de TAB. 6.9, on va présenter uniquement quelques chaînes qui expliquent les règles de la définition 6.3.2.

Étant donné $\epsilon = pot(0^{(1)}) = \{e_1; \bar{e}_1; e_2; \bar{e}_2; e_3; \bar{e}_3\}$. On considère la liste de chaînons suivante $\{(0, 1, e_1, \pi_1); (1, 2, e_2, 1); (2, 3, e_3, 1)\}$. Dans cette liste, on peut voir que les chaînons sont ordonnés selon ordre de priorité des événements et l'état d'arrivée 1 du chaînon $(0, 1, e_1, \pi_1)$

Événement	Chaînon de trans. globale	Événement	Chaînon de trans. globale
e_1	$(0,1,e_1,\pi_1)$ $(0,4,e_1,\pi_2)$	$\bar{e}_1$	$(0,0,\bar{e}_1,1)$
e_2	$(1,2,e_2,1)$ $(3,0,e_2,1)$	$\bar{e}_2$	$(1,1,\bar{e}_2,1)$ $(3,3,\bar{e}_2,1)$
e_3	$(1,3,e_3,1)$ $(2,3,e_3,1)$ $(4,3,e_3,1)$	$\bar{e}_3$	$(1,1,\bar{e}_3,1)$ $(2,2,\bar{e}_3,1)$ $(4,4,\bar{e}_3,1)$

TAB. 6.9 – Ensemble de chaînons de transition globale $\tilde{\mathcal{T}}_\epsilon$ de FIG. 6.13

est égal à l'état de départ du chaînon $(1,2,e_2,1)$ et son état d'arrivée est égal à l'état de départ du chaînon $(2,3,e_3,1)$ qui le suit. On voit aussi qu'aucun autre chaînon de transition globale ne peut être introduit à la liste, soit au début (il n'y a pas d'événement plus prioritaire que e_1, premier chaînon de la chaîne), soit au milieux (aucun chaînon de l'ensemble des chaînons possibles $\tilde{\mathcal{T}}_\epsilon$ (TAB. 6.9) ne peut être inséré entre deux chaînons de cette chaîne), soit à la fin (aucun événement est moins prioritaire que e_3, dernier chaînon de la chaîne). Dans ce cas, en respectant les règles de la définition 6.3.2, cette liste est une *chaîne de transitions globales*.

Considérons maintenant la liste de chaînons suivant $\{(0,1,e_1,\pi_1);(1,2,e_2,1)\}$. Cette liste est une sous-liste de la liste précédente, et, les chaînons sont ordonnés selon l'ordre de priorité des événements et les états de départ et d'arrivée des chaînons s'enchaînent. Cependant, il existe un chaînon $(2,3,e_3,1)$ dans $\tilde{\mathcal{T}}_\epsilon$ qui peut être encore ajouté à la fin de la liste, en effet, $\varrho_{e_2} < \varrho_{e_3}$ et l'état d'arrivée du dernier chaînon de la chaîne $(1,2,e_2,1)$ est égal à l'état de départ du chaînon $(2,3,e_3,1)$ de l'ensemble des chaînons $\tilde{\mathcal{T}}_\epsilon$ (TAB. 6.9). Cette liste n'est pas une chaîne de transitions globales, car elle ne respecte pas la règle 4 de la définition 6.3.2, qui définit qu'une liste des chaînons de transition globale est une chaîne de transitions globales uniquement s'il n'existe pas d'autres chaînons de transition globale qui puissent être ajouté à la fin de la liste.

On va considérer maintenant la liste de transitions globales $\{(0,1,e_1,\pi_1);(1,3,e_3,1)\}$. Cette liste respecte l'ordre de priorité des événements, l'enchaînement des chaînons et aucun autre chaînon de transition globale dans $\tilde{\mathcal{T}}_\epsilon$ (TAB. 6.9) ne peut être introduit au début (e_1 est l'événement le plus prioritaire) ou à la fin (e_3 est l'événement le moins prioritaire) de la liste. Cependant, il existe un chaînon $(1,1,\bar{e}_2,1) \in \tilde{\mathcal{T}}_\epsilon$, qui peut être introduit entre les deux chaînons de la liste : $\varrho_{e_1} < \varrho_{e_2} < \varrho_{e_3}$ et l'état d'arrivée du premier chaînon de la liste $(0,1,e_1,\pi_1)$ est égal à l'état de départ du chaînon $(1,1,\bar{e}_2,1)$ et l'état d'arrivée est égal à l'état de départ du dernier chaînon de la liste $(1,3,e_3,1)$. Autrement dit, cette liste ne respecte pas la règle 5 de la définition 6.3.2 et elle n'est pas une chaîne de transitions globales, car il existe une chaîne plus longue ($\{(0,1,e_1,\pi_1);(1,1,\bar{e}_2,1);(1,3,e_3,1)\}$) qui contient tous les chaînons de la liste $\{(0,1,e_1,\pi_1);(1,3,e_3,1)\}$.

Prenons un exemple de chaînes de transitions globales qui ne contiennent pas tous les événements possibles de l'état $0^{(1)}$ de l'automate complété $\breve{\mathcal{A}}^{(1)}$. Prenons la liste de transitions globales $\{(0,4,e_1,\pi_2);(4,3,e_3,1)\}$. Cette liste est une chaîne de transitions globales. Les chaînons

sont ordonnés par l'ordre de priorité de événements, l'état d'arrivée du chaînon $(0, 4, e_1, \pi_2)$ est égal à l'état de départ du chaînon $(4, 3, e_3, 1)$ et aucun autre chaînon ne peut être introduit, soit au début, soit au mieux, soit à la fin de la liste. Il faut remarquer que cette chaîne de transitions globales ne contient pas des chaînons avec l'événement e_2, ni son événement complémentaire $\bar{e}_2$, qui sont dans l'ensemble $\epsilon = \{e_1; \bar{e}_1; e_2; \bar{e}_2; e_3; \bar{e}_3\}$, c'est-à-dire, dans $pot(0^{(1)})$. Les événements e_2 et $\bar{e}_2$ ne sont pas présents dans cette chaîne car le premier chaînon de la chaîne $(0, 4, e_1, \pi_2)$ amène à l'état 4, où les événements e_2 et $\bar{e}_2$ ne sont pas réalisables.

La définition suivante nomme l'ensemble des *événements possibles* à partir d'un état global $\tilde{x}$. Pour que l'événement soit un événement possible globalement, il doit être réalisable dans tous les automates concernés par l'événement.

Définition 6.3.3. *On définit l'ensemble d'événements* $pot(\tilde{x})$ *comme :* $e \in pot(\tilde{x}) \Leftrightarrow \forall i \in \mathcal{O}_e, e \in pot(x^{(i)})$

6.3.3 Automate global

Définition 6.3.4. *Soit un* SAN *complété* $(\breve{\mathcal{E}}, \breve{\mathcal{A}}^{(i)}, \mathcal{S}, \mathcal{F})$*, où* $i \in [1..N]$*, l'automate global* $\mathcal{A}$ *est défini par :*

1. *l'ensemble d'états* $\mathcal{S}$ *;*
2. *l'ensemble d'événements* $\breve{\mathcal{E}}$ *;*
3. *la fonction d'atteignabilité* $\mathcal{F}$ *;*
4. *la matrice de transition globale* $\mathcal{G} : \mathcal{S} \times \mathcal{S} \to \mathcal{L}$ *:*

$$\forall \tilde{x} \in \mathcal{S}, \forall \tilde{y} \in \mathcal{S}$$
$$\mathcal{G}(\tilde{x}, \tilde{y}) = \bigcup_{\forall g \in \mathcal{L}(\tilde{x}, \tilde{y})} \{g\}$$

Contrairement au formalisme SAN en temps continu, ici un automate local n'est pas le même objet mathématique que l'automate global. En effet, les éléments de transition de la matrice de transition locale d'un automate contiennent un ensemble des tuples de transition locale (e, π) où l'occurrence de n'importe quel événement de l'élément de transition déclenche la transition. Dans l'automate global, les éléments de transition de la matrice de transition globale contiennent un ensemble de chaînes de transitions globales $\{(\tilde{x}_1, \tilde{y}_1, e_1, \Pi_{(\tilde{x}_1, \tilde{y}_1)}), \ldots, (\tilde{x}_C, \tilde{y}_C, e_C, \Pi_{(\tilde{x}_C, \tilde{y}_C)})\}$ où tous les événements associés aux chaînons de cette chaîne doivent avoir lieu pour que la transition se réalise.

6.4 La chaîne de Markov

La probabilité d'occurrence d'une chaîne de transitions globales est le produit de toutes les probabilités d'occurrence ρ_e des événements de la chaîne de transitions globales et des probabilités de routage globales $\Pi_e(\tilde{x}, \tilde{y})$.

✍ **Notation**

$\rho_e(\tilde{x})$	la probabilité d'occurrence de l'événement e évaluée pour l'état global $\tilde{x}$;
$\Pi_e(\tilde{x}_e, \tilde{y}_e)(\tilde{x})$	la probabilité de routage globale $\Pi_e(\tilde{x}_e, \tilde{y}_e)$ du chaînon de transition globale $(\tilde{x}_e, \tilde{y}_e, e, \Pi_e(\tilde{x}_e, \tilde{y}_e))$ évaluée pour l'état global $\tilde{x}$. On remarque de l'état global $\tilde{x}$ est égal à l'état de départ du chaînon.

Définition 6.4.1. *Étant donné une chaîne de transitions globales* $g = \{(\tilde{x}_1, \tilde{y}_1, e_1, \Pi_{e_1}(\tilde{x}_1, \tilde{y}_1)), \ldots, (\tilde{x}_C, \tilde{y}_C, e_C, \Pi_{e_C}(\tilde{x}_C, \tilde{y}_C))\}$*, la probabilité d'occurrence de cette chaîne est :*

$$\Pi(g) = (\rho_{e_1}(\tilde{x}_1) \times \Pi_{e_1}(\tilde{x}_1, \tilde{y}_1)(\tilde{x}_1)) \times \ldots \times (\rho_{e_C}(\tilde{x}_C) \times \Pi_{e_C}(\tilde{x}_C, \tilde{y}_C)(\tilde{x}_C))$$

où $\times$ *note le produit de réels.*

La chaîne de Markov représentée par un modèle SAN est définie sur les états atteignables $\mathcal{R}$ du modèle SAN. À partir de l'automate global, on peut aisément définir la chaîne de Markov représentée par le modèle SAN. La suite des états pris par l'automate global est un processus aléatoire. Dans la construction qu'on vient de faire, ce processus est tel que la probabilité d'atteindre l'état suivant ne dépend que de l'état courant et de la probabilité d'occurrence d'un ensemble d'événements simultanés qui sont ceux des chaînes qui sont associées aux transitions dans $\mathcal{G}$. On peut donc lui associer une matrice de transition de Markov de la façon suivante :

Définition 6.4.2. *Soit un* SAN *dont l'automate global* $\mathcal{A}\ (\mathcal{S}, \breve{\mathcal{E}}, \mathcal{F}, \mathcal{G})$*, la matrice de transition de la chaîne de Markov modélisé par ce* SAN *:*

1. $P : \mathcal{R} \times \mathcal{R} \to \mathbb{R}$ *:*

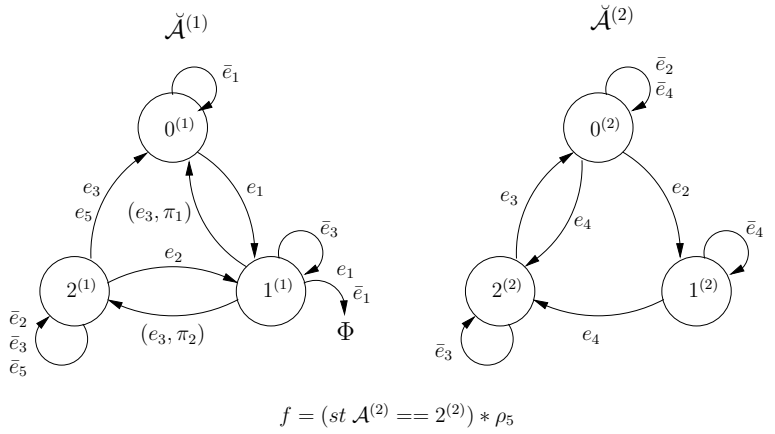
$$\forall \tilde{x} \in \mathcal{R}, \forall \tilde{y} \in \mathcal{R}$$
$$P(\tilde{x}, \tilde{y}) = \sum_{\forall g \in \mathcal{G}(\tilde{x}, \tilde{y})} \Pi(g)$$

La probabilité de transition de l'état $\tilde{x}$ vers l'état $\tilde{y}$ de la chaîne de Markov est la somme des probabilités d'occurrence de chaque chaîne de transitions globales qui amène de l'état $\tilde{x}$ vers l'état $\tilde{y}$.

6.4.1 Exemple de construction de la chaîne de Markov

Pour clarifier la construction de la chaîne de Markov, on va présenter pour un exemple et pour chaque état global atteignable $(\tilde{x} \in \mathcal{R})$, l'ensemble d'événements possibles ($pot(\tilde{x})$) ainsi que l'ensemble de chaînons de transition globale pour chaque $pot(\tilde{x})$ et les chaînes de transitions globales sortant de chaque état.

Le modèle considéré est celui présenté dans FIG. 6.12. On rappelle ce modèle.



Événement	Prob.	Priorité
e_1	ρ_1	3
e_2	ρ_2	1
e_3	ρ_3	2
e_4	ρ_4	4
e_5	f	5

Événement	Prob.	Priorité
$\bar{e}_1$	$1-\rho_1$	3
$\bar{e}_2$	$1-\rho_2$	1
$\bar{e}_3$	$1-\rho_3$	2
$\bar{e}_4$	$1-\rho_4$	4
$\bar{e}_5$	$1-f$	5

FIG. 6.14 – Réseau d'automates complétés

Pour l'état global $0^{(1)}0^{(2)}$ l'ensemble d'événements possibles est $pot(0^{(1)}0^{(2)}) = \{e_1, \bar{e}_1, e_4, \bar{e}_4\}$. L'ensemble de chaînons de transition globale pour cet ensemble $\tilde{\mathcal{T}}_{pot(0^{(1)}0^{(2)})}$ est présenté dans TAB. 6.10.

Événement	Chaînon de transition globale	Événement	Chaînon de transition globale
e_1	$(0^{(1)}0^{(2)}, 1^{(1)}0^{(2)}, e_1, 1)$	$\bar{e}_1$	$(0^{(1)}0^{(2)}, 0^{(1)}0^{(2)}, \bar{e}_1, 1)$
	$(0^{(1)}1^{(2)}, 1^{(1)}1^{(2)}, e_1, 1)$		$(0^{(1)}1^{(2)}, 0^{(1)}1^{(2)}, \bar{e}_1, 1)$
	$(0^{(1)}2^{(2)}, 1^{(1)}2^{(2)}, e_1, 1)$		$(0^{(1)}2^{(2)}, 0^{(1)}2^{(2)}, \bar{e}_1, 1)$
e_4	$(0^{(1)}0^{(2)}, 0^{(1)}2^{(2)}, e_4, 1)$	$\bar{e}_4$	$(0^{(1)}0^{(2)}, 0^{(1)}0^{(2)}, \bar{e}_4, 1)$
	$(0^{(1)}1^{(2)}, 0^{(1)}2^{(2)}, e_4, 1)$		$(0^{(1)}1^{(2)}, 0^{(1)}1^{(2)}, \bar{e}_4, 1)$
	$(1^{(1)}0^{(2)}, 1^{(1)}2^{(2)}, e_4, 1)$		$(1^{(1)}0^{(2)}, 1^{(1)}0^{(2)}, \bar{e}_4, 1)$
	$(1^{(1)}1^{(2)}, 1^{(1)}2^{(2)}, e_4, 1)$		$(1^{(1)}1^{(2)}, 1^{(1)}1^{(2)}, \bar{e}_4, 1)$
	$(2^{(1)}0^{(2)}, 2^{(1)}2^{(2)}, e_4, 1)$		$(2^{(1)}0^{(2)}, 2^{(1)}0^{(2)}, \bar{e}_4, 1)$
	$(2^{(1)}1^{(2)}, 2^{(1)}2^{(2)}, e_4, 1)$		$(2^{(1)}1^{(2)}, 2^{(1)}1^{(2)}, \bar{e}_4, 1)$

TAB. 6.10 – Ensemble de chaînons de transition globale $\tilde{\mathcal{T}}_{pot(0^{(1)}0^{(2)})}$ de FIG. 6.14

À partir de ces chaînons, on peut construire les chaînes de transitions globales (TAB. 6.11) pour l'état global $0^{(1)}0^{(2)}$ et calculer les probabilités d'occurrence des transitions.

État d'arrivée	Chaîne de transitions globales	Probabilité de transition
$0^{(1)}0^{(2)}$	$\{(0^{(1)}0^{(2)}, 0^{(1)}0^{(2)}, \bar{e}_1, 1); (0^{(1)}0^{(2)}, 0^{(1)}0^{(2)}, \bar{e}_4, 1)\}$	$(1-\rho_1)(1-\rho_4)$
$0^{(1)}2^{(2)}$	$\{(0^{(1)}0^{(2)}, 0^{(1)}0^{(2)}, \bar{e}_1, 1); (0^{(1)}0^{(2)}, 0^{(1)}2^{(2)}, e_4, 1)\}$	$(1-\rho_1)\rho_4$
$1^{(1)}0^{(2)}$	$\{(0^{(1)}0^{(2)}, 1^{(1)}0^{(2)}, e_1, 1); (1^{(1)}0^{(2)}, 1^{(1)}0^{(2)}, \bar{e}_4, 1)\}$	$\rho_1(1-\rho_4)$
$1^{(1)}2^{(2)}$	$\{(0^{(1)}0^{(2)}, 1^{(1)}0^{(2)}, e_1, 1); (1^{(1)}0^{(2)}, 1^{(1)}2^{(2)}, e_4, 1)\}$	$\rho_1\rho_4$

TAB. 6.11 – Ensemble des chaînes de transitions globales de l'état global $0^{(1)}0^{(2)}$ de FIG. 6.14

FIG. 6.15 présente les chemins de construction des chaînes de transitions globales pour l'état $0^{(1)}0^{(2)}$.

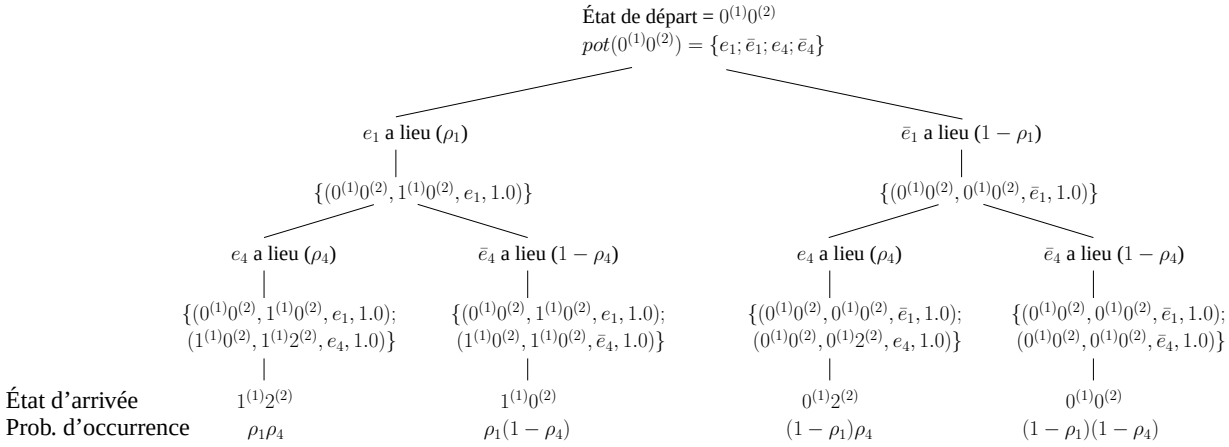


FIG. 6.15 – Construction des chaînes de transitions globales pour l'état $0^{(1)}0^{(2)}$

Pour l'état global $0^{(1)}1^{(2)}$ l'ensemble d'événements possibles est $pot(0^{(1)}1^{(2)}) = \{e_1, \bar{e}_1, e_4, \bar{e}_4\}$. L'ensemble de chaînons de transition globale pour cet ensemble $\tilde{\mathcal{T}}_{pot(0^{(1)}1^{(2)})}$ est présenté dans TAB. 6.12.

Événement	Chaînon de transition globale	Événement	Chaînon de transition globale
e_1	$(0^{(1)}0^{(2)}, 1^{(1)}0^{(2)}, e_1, 1)$	$\bar{e}_1$	$(0^{(1)}0^{(2)}, 0^{(1)}0^{(2)}, \bar{e}_1, 1)$
	$(0^{(1)}1^{(2)}, 1^{(1)}1^{(2)}, e_1, 1)$		$(0^{(1)}1^{(2)}, 0^{(1)}1^{(2)}, \bar{e}_1, 1)$
	$(0^{(1)}2^{(2)}, 1^{(1)}2^{(2)}, e_1, 1)$		$(0^{(1)}2^{(2)}, 0^{(1)}2^{(2)}, \bar{e}_1, 1)$
e_4	$(0^{(1)}0^{(2)}, 0^{(1)}2^{(2)}, e_4, 1)$	$\bar{e}_4$	$(0^{(1)}0^{(2)}, 0^{(1)}0^{(2)}, \bar{e}_4, 1)$
	$(0^{(1)}1^{(2)}, 0^{(1)}2^{(2)}, e_4, 1)$		$(0^{(1)}1^{(2)}, 0^{(1)}1^{(2)}, \bar{e}_4, 1)$
	$(1^{(1)}0^{(2)}, 1^{(1)}2^{(2)}, e_4, 1)$		$(1^{(1)}0^{(2)}, 1^{(1)}0^{(2)}, \bar{e}_4, 1)$
	$(1^{(1)}1^{(2)}, 1^{(1)}2^{(2)}, e_4, 1)$		$(1^{(1)}1^{(2)}, 1^{(1)}1^{(2)}, \bar{e}_4, 1)$
	$(2^{(1)}0^{(2)}, 2^{(1)}2^{(2)}, e_4, 1)$		$(2^{(1)}0^{(2)}, 2^{(1)}0^{(2)}, \bar{e}_4, 1)$
	$(2^{(1)}1^{(2)}, 2^{(1)}2^{(2)}, e_4, 1)$		$(2^{(1)}1^{(2)}, 2^{(1)}1^{(2)}, \bar{e}_4, 1)$

TAB. 6.12 – Ensemble de chaînons de transition globale $\tilde{\mathcal{T}}_{pot(0^{(1)}1^{(2)})}$ de FIG. 6.14

À partir de ces chaînons, on peut construire les chaînes de transitions globales (TAB. 6.13) pour l'état global $0^{(1)}1^{(2)}$ et calculer les probabilités d'occurrence des transitions.

État d'arrivée	Chaîne de transitions globales	Probabilité de transition
$0^{(1)}1^{(2)}$	$\{(0^{(1)}1^{(2)}, 0^{(1)}1^{(2)}, \bar{e}_1, 1); (0^{(1)}1^{(2)}, 0^{(1)}1^{(2)}, \bar{e}_4, 1)\}$	$(1-\rho_1)(1-\rho_4)$
$0^{(1)}2^{(2)}$	$\{(0^{(1)}1^{(2)}, 0^{(1)}1^{(2)}, \bar{e}_1, 1); (0^{(1)}1^{(2)}, 0^{(1)}2^{(2)}, e_4, 1)\}$	$(1-\rho_1)\rho_4$
$1^{(1)}1^{(2)}$	$\{(0^{(1)}1^{(2)}, 1^{(1)}1^{(2)}, e_1, 1); (1^{(1)}1^{(2)}, 1^{(1)}1^{(2)}, \bar{e}_4, 1)\}$	$\rho_1(1-\rho_4)$
$1^{(1)}2^{(2)}$	$\{(0^{(1)}1^{(2)}, 1^{(1)}1^{(2)}, e_1, 1); (1^{(1)}1^{(2)}, 1^{(1)}2^{(2)}, e_4, 1)\}$	$\rho_1\rho_4$

TAB. 6.13 – Ensemble des chaînes de transitions globales de l'état global $0^{(1)}1^{(2)}$ de FIG. 6.14

FIG. 6.16 présente les chemins de construction de ces chaînes de transitions globales.

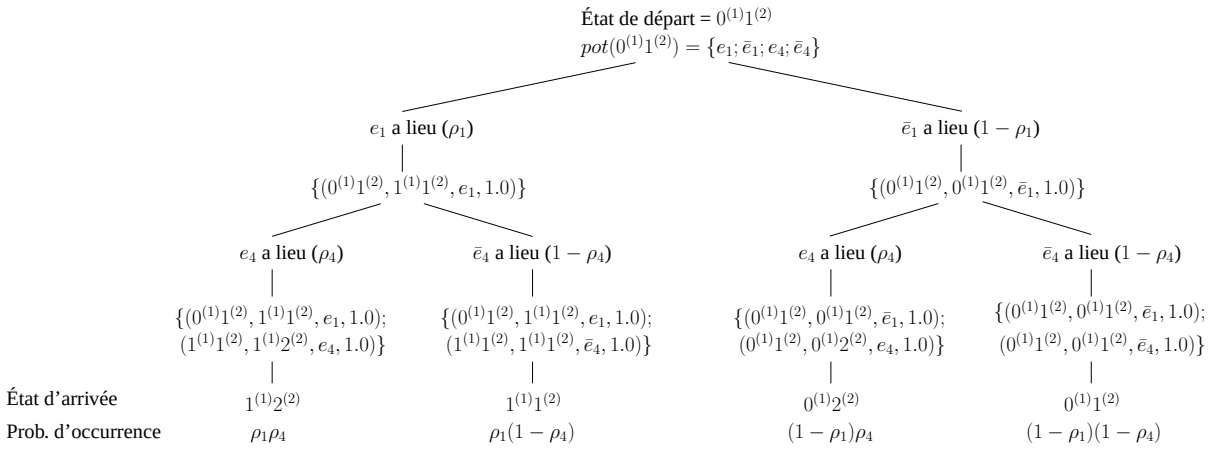


FIG. 6.16 – Construction des chaînes de transitions globales pour l'état $0^{(1)}1^{(2)}$

Pour l'état global $0^{(1)}2^{(2)}$ l'ensemble d'événements possibles est $pot(0^{(1)}2^{(2)}) = \{e_1, \bar{e}_1\}$. L'ensemble de chaînons de transition globale pour cet ensemble $\tilde{\mathcal{T}}_{pot(0^{(1)}0^{(2)})}$ est présenté dans TAB. 6.14.

Événement	Chaînon de transition globale	Événement	Chaînon de transition globale
e_1	$(0^{(1)}0^{(2)}, 1^{(1)}0^{(2)}, e_1, 1)$	$\bar{e}_1$	$(0^{(1)}0^{(2)}, 0^{(1)}0^{(2)}, \bar{e}_1, 1)$
	$(0^{(1)}1^{(2)}, 1^{(1)}1^{(2)}, e_1, 1)$		$(0^{(1)}1^{(2)}, 0^{(1)}1^{(2)}, \bar{e}_1, 1)$
	$(0^{(1)}2^{(2)}, 1^{(1)}2^{(2)}, e_1, 1)$		$(0^{(1)}2^{(2)}, 0^{(1)}2^{(2)}, \bar{e}_1, 1)$

TAB. 6.14 – Ensemble de chaînons de transition globale $\tilde{\mathcal{T}}_{pot(0^{(1)}2^{(2)})}$ de FIG. 6.14

À partir de ces chaînons, on peut construire les chaînes de transitions globales suivantes (TAB. 6.15) pour l'état global $0^{(1)}2^{(2)}$, ainsi que les probabilités d'occurrence de chaque transition.

État d'arrivée	Chaîne de transitions globales	Probabilité de transition
$0^{(1)}2^{(2)}$	$\{(0^{(1)}2^{(2)}, 0^{(1)}2^{(2)}, \bar{e}_1, 1)\}$	$(1-\rho_1)$
$1^{(1)}2^{(2)}$	$\{(0^{(1)}2^{(2)}, 1^{(1)}2^{(2)}, e_1, 1)\}$	ρ_1

TAB. 6.15 – Ensemble des chaînes de transitions globales de l'état global $0^{(1)}2^{(2)}$ de FIG. 6.14

FIG. 6.17 présente les chemins de construction des chaînes de transitions globales pour l'état global $0^{(1)}2^{(2)}$.

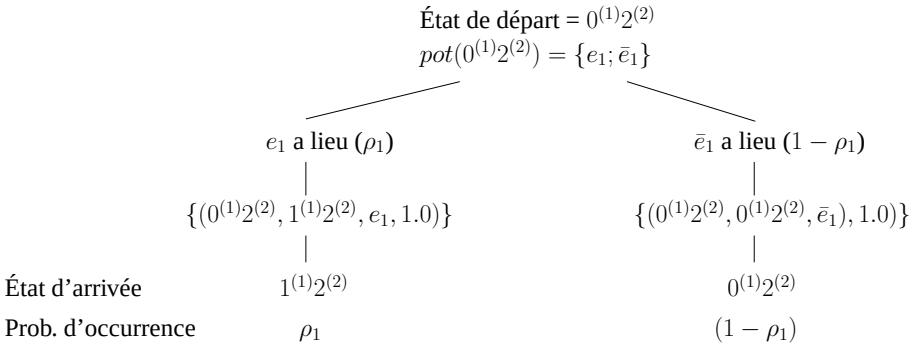


FIG. 6.17 – Construction des chaînes de transitions globales pour l'état $0^{(1)}2^{(2)}$

Pour l'état global $1^{(1)}0^{(2)}$ l'ensemble d'événements possibles est $pot(1^{(1)}0^{(2)}) = \{e_1, \bar{e}_1, e_4, \bar{e}_4\}$. L'ensemble de chaînons de transition globale pour cet ensemble $\tilde{\mathcal{T}}_{pot(1^{(1)}0^{(2)})}$ est présenté dans TAB. 6.16.

Événement	Chaînon de transition globale	Événement	Chaînon de transition globale
e_1	$(0^{(1)}0^{(2)}, 1^{(1)}0^{(2)}, e_1, 1)$	$\bar{e}_1$	$(0^{(1)}0^{(2)}, 0^{(1)}0^{(2)}, \bar{e}_1, 1)$
	$(0^{(1)}1^{(2)}, 1^{(1)}1^{(2)}, e_1, 1)$		$(0^{(1)}1^{(2)}, 0^{(1)}1^{(2)}, \bar{e}_1, 1)$
	$(0^{(1)}2^{(2)}, 1^{(1)}2^{(2)}, e_1, 1)$		$(0^{(1)}2^{(2)}, 0^{(1)}2^{(2)}, \bar{e}_1, 1)$
e_4	$(0^{(1)}0^{(2)}, 0^{(1)}2^{(2)}, e_4, 1)$	$\bar{e}_4$	$(0^{(1)}0^{(2)}, 0^{(1)}0^{(2)}, \bar{e}_4, 1)$
	$(0^{(1)}1^{(2)}, 0^{(1)}2^{(2)}, e_4, 1)$		$(0^{(1)}1^{(2)}, 0^{(1)}1^{(2)}, \bar{e}_4, 1)$
	$(1^{(1)}0^{(2)}, 1^{(1)}2^{(2)}, e_4, 1)$		$(1^{(1)}0^{(2)}, 1^{(1)}0^{(2)}, \bar{e}_4, 1)$
	$(1^{(1)}1^{(2)}, 1^{(1)}2^{(2)}, e_4, 1)$		$(1^{(1)}1^{(2)}, 1^{(1)}1^{(2)}, \bar{e}_4, 1)$
	$(2^{(1)}0^{(2)}, 2^{(1)}2^{(2)}, e_4, 1)$		$(2^{(1)}0^{(2)}, 2^{(1)}0^{(2)}, \bar{e}_4, 1)$
	$(2^{(1)}1^{(2)}, 2^{(1)}2^{(2)}, e_4, 1)$		$(2^{(1)}1^{(2)}, 2^{(1)}1^{(2)}, \bar{e}_4, 1)$

TAB. 6.16 – Ensemble de chaînons de transition globale $\tilde{\mathcal{T}}_{pot(1^{(1)}0^{(2)})}$ de FIG. 6.14

On remarque que les événements e_1 et $\bar{e}_1$ sont des événements possibles à partir de cet état, mais ils ne sont pas réalisables. En sachant que les événements e_1 et $\bar{e}_1$ sont les événements plus prioritaires de $pot(1^{(1)}0^{(2)})$ et donc, aucun autre événement peut avoir lieu avant eux, ils ne apparaissent dans aucune chaîne de transitions globales à partir de cet état.

À partir de ces chaînons, on peut construire les chaînes de transitions globales suivantes (TAB. 6.17) pour l'état global $1^{(1)}0^{(2)}$.

État d'arrivée	Chaîne de transitions globales	Probabilité de transition
$1^{(1)}0^{(2)}$	$\{(1^{(1)}0^{(2)}, 1^{(1)}0^{(2)}, \bar{e}_4, 1)\}$	$(1-\rho_4)$
$1^{(1)}2^{(2)}$	$\{(1^{(1)}0^{(2)}, 1^{(1)}2^{(2)}, e_4, 1)\}$	ρ_4

TAB. 6.17 – Ensemble des chaînes de transitions globales de l'état global $1^{(1)}0^{(2)}$ de FIG. 6.14

FIG. 6.18 présente les chemins de construction de ces chaînes de transitions globales.

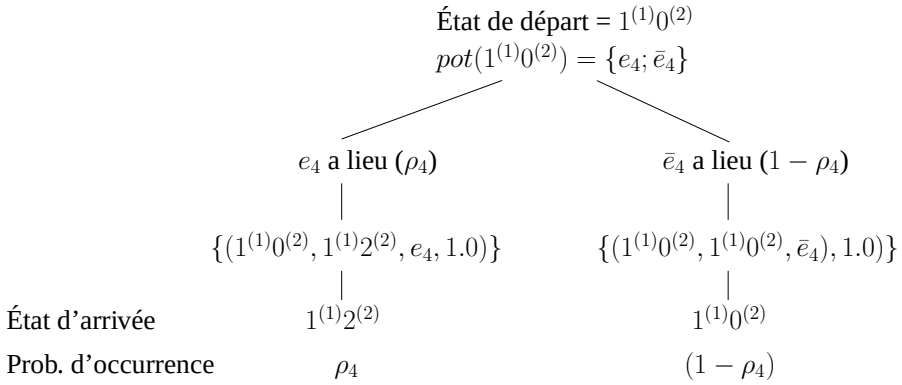


FIG. 6.18 – Construction des chaînes de transitions globales pour l'état $1^{(1)}0^{(2)}$

De façon similaire à l'état global précédente, pour l'état global $1^{(1)}1^{(2)}$ l'ensemble d'événements possibles est composé uniquement de l'événement e_4 et de son événement complémentaire $\bar{e}_4$ ($pot(1^{(1)}1^{(2)}) = \{e_1, \bar{e}_1, e_4, \bar{e}_4\}$). L'ensemble de chaînons de transition globale pour cet ensemble $\tilde{\mathcal{T}}_{pot(1^{(1)}1^{(2)})}$ est le même présenté dans TAB. 6.16.

Cependant, les chaînes de transitions globales sont différentes, car l'état de départ n'est pas le même. TAB. 6.18 présente les chaînes de transitions globales pour l'état global $1^{(1)}1^{(2)}$.

État d'arrivée	Chaîne de transitions globales	Probabilité de transition
$1^{(1)}1^{(2)}$	$\{(1^{(1)}1^{(2)}, 1^{(1)}1^{(2)}, \bar{e}_4, 1)\}$	$(1-\rho_4)$
$1^{(1)}2^{(2)}$	$\{(1^{(1)}1^{(2)}, 1^{(1)}2^{(2)}, e_4, 1)\}$	ρ_4

TAB. 6.18 – Ensemble des chaînes de transitions globales de l'état global $1^{(1)}1^{(2)}$ de FIG. 6.14

FIG. 6.19 présente les chemins de construction de ces chaînes de transitions globales.

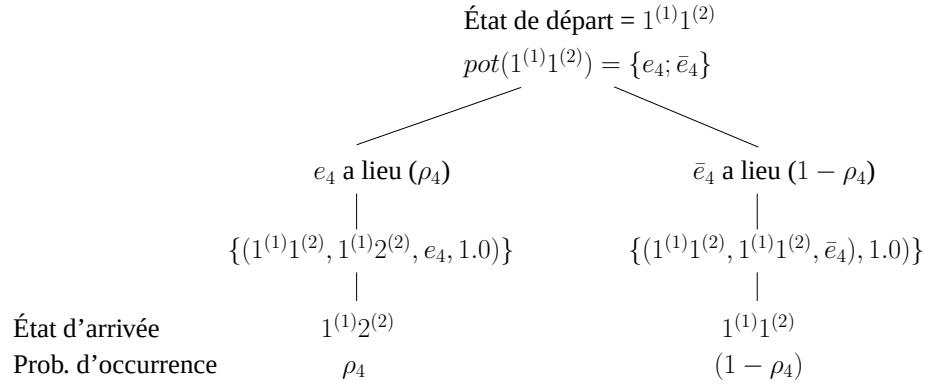


FIG. 6.19 – Construction des chaînes de transitions globales pour l'état $1^{(1)}1^{(2)}$

Pour l'état global $1^{(1)}2^{(2)}$ l'ensemble d'événements possibles est $pot(1^{(1)}2^{(2)}) = \{e_3, \bar{e}_3, e_1, \bar{e}_1\}$.

L'ensemble de chaînons de transition globale pour cet ensemble $\tilde{\mathcal{T}}_{pot(1^{(1)}2^{(2)})}$ est présenté dans TAB. 6.19.

Événement	Chaînon de transition globale	Événement	Chaînon de transition globale
e_3	$(1^{(1)}2^{(2)}, 0^{(1)}0^{(2)}, e_3, \pi_1)$ $(1^{(1)}2^{(2)}, 2^{(1)}0^{(2)}, e_3, \pi_2)$ $(2^{(1)}2^{(2)}, 0^{(1)}0^{(2)}, e_3, 1)$	$\bar{e}_3$	$(1^{(1)}2^{(2)}, 1^{(1)}2^{(2)}, \bar{e}_3, 1)$ $(2^{(1)}2^{(2)}, 2^{(1)}2^{(2)}, \bar{e}_3, 1)$
e_1	$(0^{(1)}0^{(2)}, 1^{(1)}0^{(2)}, e_1, 1)$ $(0^{(1)}1^{(2)}, 1^{(1)}1^{(2)}, e_1, 1)$ $(0^{(1)}2^{(2)}, 1^{(1)}2^{(2)}, e_1, 1)$	$\bar{e}_1$	$(0^{(1)}0^{(2)}, 0^{(1)}0^{(2)}, \bar{e}_1, 1)$ $(0^{(1)}1^{(2)}, 0^{(1)}1^{(2)}, \bar{e}_1, 1)$ $(0^{(1)}2^{(2)}, 0^{(1)}2^{(2)}, \bar{e}_1, 1)$

TAB. 6.19 – Ensemble de chaînons de transition globale $\tilde{\mathcal{T}}_{pot(1^{(1)}2^{(2)})}$ de FIG. 6.14

À partir de ces chaînons, on peut construire les chaînes de transitions globales suivantes (TAB. 6.20) pour l'état global $1^{(1)}2^{(2)}$.

État d'arrivée	Chaîne de transitions globales	Probabilité de transition
$0^{(1)}0^{(2)}$	$\{(1^{(1)}2^{(2)}, 0^{(1)}0^{(2)}, e_3, \pi_1); (0^{(1)}0^{(2)}, 0^{(1)}0^{(2)}, \bar{e}_1, 1)\}$	$\rho_3\pi_1(1-\rho_1)$
$1^{(1)}1^{(2)}$	$\{(1^{(1)}2^{(2)}, 0^{(1)}0^{(2)}, e_3, \pi_1); (0^{(1)}0^{(2)}, 1^{(1)}0^{(2)}, e_1, 1)\}$	$\rho_3\pi_1\rho_1$
$1^{(1)}2^{(2)}$	$\{(1^{(1)}2^{(2)}, 1^{(1)}2^{(2)}, \bar{e}_3, 1)\}$	$(1-\rho_3)$
$2^{(1)}0^{(2)}$	$\{(1^{(1)}2^{(2)}, 2^{(1)}0^{(2)}, e_3, \pi_2)\}$	$\rho_3\pi_2$

TAB. 6.20 – Ensemble des chaînes de transitions globales de l'état global $1^{(1)}2^{(2)}$ de FIG. 6.14

FIG. 6.20 présente les chemins de construction de ces chaînes de transitions globales.

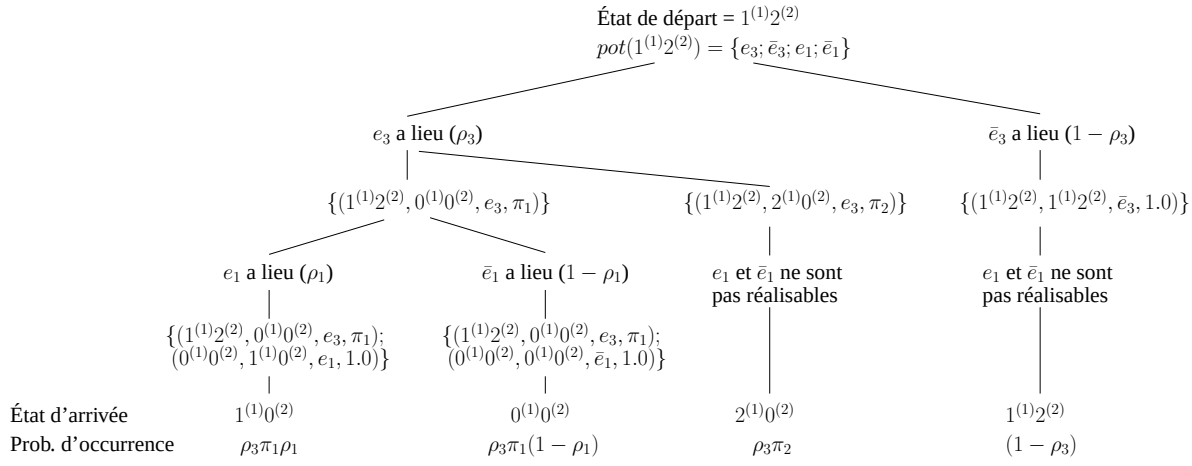


FIG. 6.20 – Construction des chaînes de transitions globales pour l'état $1^{(1)}2^{(2)}$

Pour l'état global $2^{(1)}0^{(2)}$ l'ensemble d'événements possibles est $pot(2^{(1)}0^{(2)}) = \{e_2, \bar{e}_2, e_4, \bar{e}_4, e_5, \bar{e}_5\}$. L'ensemble de chaînons de transition globale pour cet ensemble $\tilde{\mathcal{T}}_{pot(2^{(1)}0^{(2)})}$ est présenté dans TAB. 6.21.

Événement	Chaînon de transition globale	Événement	Chaînon de transition globale
e_2	$(2^{(1)}0^{(2)}, 1^{(1)}1^{(2)}, e_2, 1)$	$\bar{e}_2$	$(2^{(1)}0^{(2)}, 2^{(1)}0^{(2)}, \bar{e}_2, 1)$
e_4	$(0^{(1)}0^{(2)}, 0^{(1)}2^{(2)}, e_4, 1)$	$\bar{e}_4$	$(0^{(1)}0^{(2)}, 0^{(1)}0^{(2)}, \bar{e}_4, 1)$
	$(0^{(1)}1^{(2)}, 0^{(1)}2^{(2)}, e_4, 1)$		$(0^{(1)}1^{(2)}, 0^{(1)}1^{(2)}, \bar{e}_4, 1)$
	$(1^{(1)}0^{(2)}, 1^{(1)}2^{(2)}, e_4, 1)$		$(1^{(1)}0^{(2)}, 1^{(1)}0^{(2)}, \bar{e}_4, 1)$
	$(1^{(1)}1^{(2)}, 1^{(1)}2^{(2)}, e_4, 1)$		$(1^{(1)}1^{(2)}, 1^{(1)}1^{(2)}, \bar{e}_4, 1)$
	$(2^{(1)}0^{(2)}, 2^{(1)}2^{(2)}, e_4, 1)$		$(2^{(1)}0^{(2)}, 2^{(1)}0^{(2)}, \bar{e}_4, 1)$
	$(2^{(1)}1^{(2)}, 2^{(1)}2^{(2)}, e_4, 1)$		$(2^{(1)}1^{(2)}, 2^{(1)}1^{(2)}, \bar{e}_4, 1)$
e_5	$(2^{(1)}0^{(2)}, 0^{(1)}0^{(2)}, e_5, 1)$	$\bar{e}_5$	$(2^{(1)}0^{(2)}, 2^{(1)}0^{(2)}, \bar{e}_5, 1)$
	$(2^{(1)}1^{(2)}, 0^{(1)}1^{(2)}, e_5, 1)$		$(2^{(1)}1^{(2)}, 2^{(1)}1^{(2)}, \bar{e}_5, 1)$
	$(2^{(1)}2^{(2)}, 0^{(1)}2^{(2)}, e_5, 1)$		$(2^{(1)}2^{(2)}, 2^{(1)}2^{(2)}, \bar{e}_5, 1)$

TAB. 6.21 – Ensemble de chaînons de transition globale $\tilde{\mathcal{T}}_{pot(2^{(1)}0^{(2)})}$ de FIG. 6.14

À partir de ces chaînons, on peut construire les chaînes de transitions globales suivantes (TAB. 6.22) pour l'état global $2^{(1)}0^{(2)}$.

État d'arrivée	Chaîne de transitions globales	Probabilité de transition
$0^{(1)}0^{(2)}$	$\{(2^{(1)}0^{(2)}, 2^{(1)}0^{(2)}, \bar{e}_2, 1); (2^{(1)}0^{(2)}, 2^{(1)}0^{(2)}, \bar{e}_4, 1); (2^{(1)}0^{(2)}, 0^{(1)}0^{(2)}, e_5, 1)\}$	0
$0^{(1)}2^{(2)}$	$\{(2^{(1)}0^{(2)}, 2^{(1)}0^{(2)}, \bar{e}_2, 1); (2^{(1)}0^{(2)}, 2^{(1)}2^{(2)}, e_4, 1); (2^{(1)}2^{(2)}, 0^{(1)}2^{(2)}, e_5, 1)\}$	$(1-\rho_2)\rho_4\rho_5$
$1^{(1)}1^{(2)}$	$\{(2^{(1)}0^{(2)}, 1^{(1)}1^{(2)}, e_2, 1); (1^{(1)}1^{(2)}, 1^{(1)}1^{(2)}, \bar{e}_4, 1)\}$	$\rho_2(1-\rho_4)$
$1^{(1)}2^{(2)}$	$\{(2^{(1)}0^{(2)}, 1^{(1)}1^{(2)}, e_2, 1); (1^{(1)}1^{(2)}, 1^{(1)}2^{(2)}, e_4, 1)\}$	$\rho_2\rho_4$
$2^{(1)}0^{(2)}$	$\{(2^{(1)}0^{(2)}, 2^{(1)}0^{(2)}, \bar{e}_2, 1); (2^{(1)}0^{(2)}, 2^{(1)}0^{(2)}, \bar{e}_4, 1)\}$	$(1-\rho_2)(1-\rho_4)$
$2^{(1)}2^{(2)}$	$\{(2^{(1)}0^{(2)}, 2^{(1)}0^{(2)}, \bar{e}_2, 1); (2^{(1)}0^{(2)}, 2^{(1)}2^{(2)}, e_4, 1); (2^{(1)}2^{(2)}, 2^{(1)}2^{(2)}, \bar{e}_5, 1)\}$	$(1-\rho_2)\rho_4(1-\rho_5)$

TAB. 6.22 – Ensemble des chaînes de transitions globales de l'état global $2^{(1)}0^{(2)}$ de FIG. 6.14

FIG. 6.21 présente les chemins de construction de ces chaînes de transitions globales.

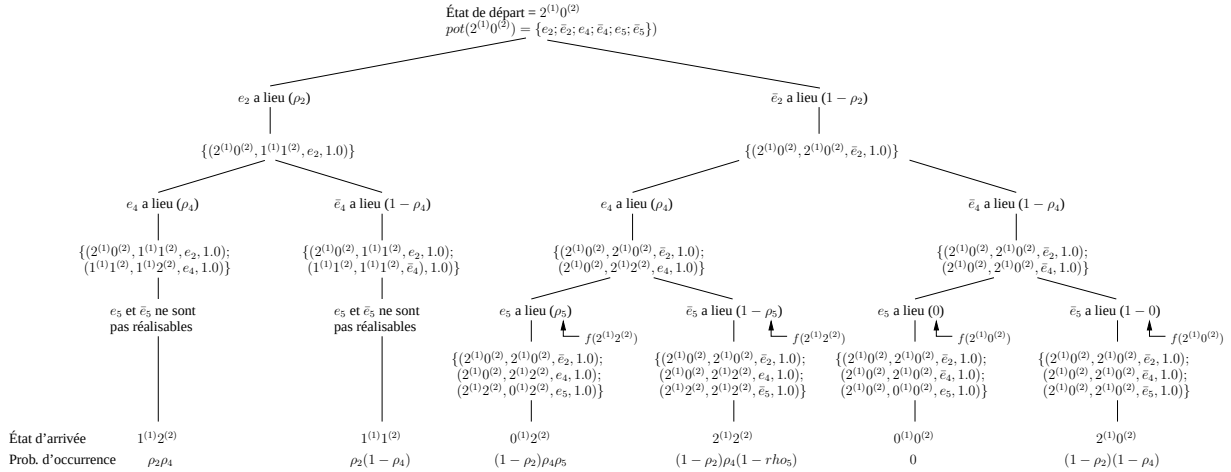


FIG. 6.21 – Construction des chaînes de transitions globales pour l'état $2^{(1)}0^{(2)}$

Pour l'état global $2^{(1)}1^{(2)}$ l'ensemble d'événements possibles est $pot(2^{(1)}1^{(2)}) = \{e_4, \bar{e}_4, e_5, \bar{e}_5\}$. L'ensemble de chaînons de transition globale pour cet ensemble $\tilde{\mathcal{T}}_{pot(2^{(1)}1^{(2)})}$ est présenté dans

TAB. 6.23.

Événement	Chaînon de transition globale	Événement	Chaînon de transition globale
e_4	$(0^{(1)}0^{(2)}, 0^{(1)}2^{(2)}, e_4, 1)$	$\bar{e}_4$	$(0^{(1)}0^{(2)}, 0^{(1)}0^{(2)}, \bar{e}_4, 1)$
	$(0^{(1)}1^{(2)}, 0^{(1)}2^{(2)}, e_4, 1)$		$(0^{(1)}1^{(2)}, 0^{(1)}1^{(2)}, \bar{e}_4, 1)$
	$(1^{(1)}0^{(2)}, 1^{(1)}2^{(2)}, e_4, 1)$		$(1^{(1)}0^{(2)}, 1^{(1)}0^{(2)}, \bar{e}_4, 1)$
	$(1^{(1)}1^{(2)}, 1^{(1)}2^{(2)}, e_4, 1)$		$(1^{(1)}1^{(2)}, 1^{(1)}1^{(2)}, \bar{e}_4, 1)$
	$(2^{(1)}0^{(2)}, 2^{(1)}2^{(2)}, e_4, 1)$		$(2^{(1)}0^{(2)}, 2^{(1)}0^{(2)}, \bar{e}_4, 1)$
	$(2^{(1)}1^{(2)}, 2^{(1)}2^{(2)}, e_4, 1)$		$(2^{(1)}1^{(2)}, 2^{(1)}1^{(2)}, \bar{e}_4, 1)$
e_5	$(2^{(1)}0^{(2)}, 0^{(1)}0^{(2)}, e_5, 1)$	$\bar{e}_5$	$(2^{(1)}0^{(2)}, 2^{(1)}0^{(2)}, \bar{e}_5, 1)$
	$(2^{(1)}1^{(2)}, 0^{(1)}1^{(2)}, e_5, 1)$		$(2^{(1)}1^{(2)}, 2^{(1)}1^{(2)}, \bar{e}_5, 1)$
	$(2^{(1)}2^{(2)}, 0^{(1)}2^{(2)}, e_5, 1)$		$(2^{(1)}2^{(2)}, 2^{(1)}2^{(2)}, \bar{e}_5, 1)$

TAB. 6.23 – Ensemble de chaînons de transition globale $\tilde{\mathcal{T}}_{pot(2^{(1)}1^{(2)})}$ de FIG. 6.14

À partir de ces chaînons, on peut construire les chaînes de transitions globales suivantes (TAB. 6.24) pour l'état global $2^{(1)}1^{(2)}$.

État d'arrivée	Chaîne de transitions globales	Probabilité de transition
$0^{(1)}1^{(2)}$	$\{(2^{(1)}1^{(2)}, 2^{(1)}1^{(2)}, \bar{e}_4, 1); (2^{(1)}1^{(2)}, 0^{(1)}1^{(2)}, e_5, 1)\}$	0
$0^{(1)}2^{(2)}$	$\{(2^{(1)}1^{(2)}, 2^{(1)}2^{(2)}, e_4, 1); (2^{(1)}2^{(2)}, 0^{(1)}2^{(2)}, e_5, 1)\}$	$\rho_4\rho_5$
$2^{(1)}1^{(2)}$	$\{(2^{(1)}1^{(2)}, 2^{(1)}1^{(2)}, \bar{e}_4, 1); (2^{(1)}1^{(2)}, 2^{(1)}1^{(2)}, \bar{e}_5, 1)\}$	$(1-\rho_4)$
$2^{(1)}2^{(2)}$	$\{(2^{(1)}1^{(2)}, 2^{(1)}2^{(2)}, e_4, 1); (2^{(1)}2^{(2)}, 2^{(1)}2^{(2)}, \bar{e}_5, 1)\}$	$\rho_4(1-\rho_5)$

TAB. 6.24 – Ensemble des chaînes de transitions globales de l'état global $2^{(1)}1^{(2)}$ de FIG. 6.14

FIG. 6.22 présente les chemins de construction de ces chaînes de transitions globales.

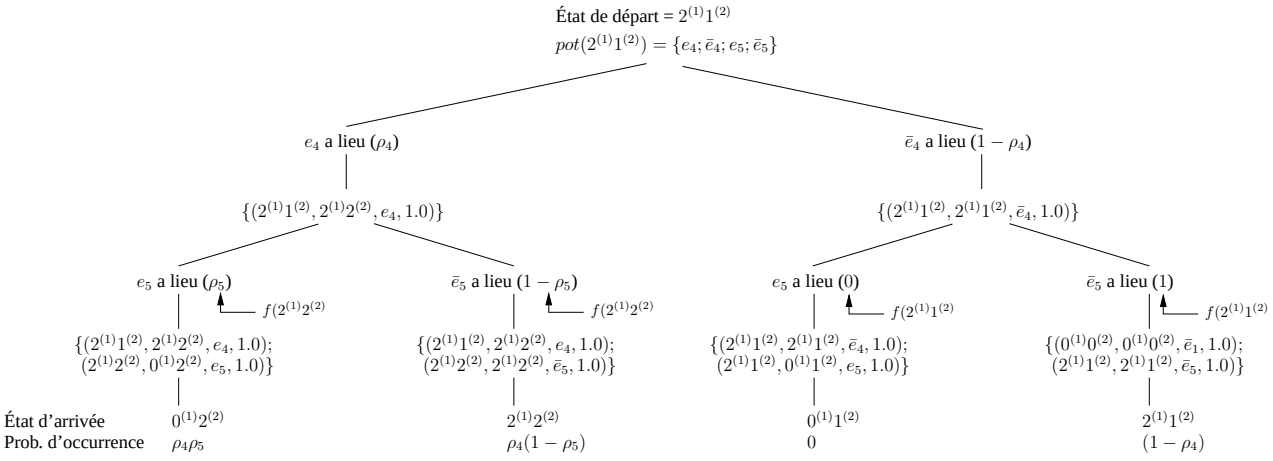


FIG. 6.22 – Construction des chaînes de transitions globales pour l'état $2^{(1)}1^{(2)}$

Pour l'état global $2^{(1)}2^{(2)}$ l'ensemble d'événements possibles est $pot(2^{(1)}2^{(2)}) = \{e_3, \bar{e}_3, e_5, \bar{e}_5\}$. L'ensemble de chaînons de transition globale pour cet ensemble $\tilde{\mathcal{T}}_{pot(2^{(1)}2^{(2)})}$ est présenté dans TAB. 6.25.

Événement	Chaînon de transition globale	Événement	Chaînon de transition globale
e_3	$(1^{(1)}2^{(2)}, 0^{(1)}0^{(2)}, e_3, \pi_1)$ $(1^{(1)}2^{(2)}, 2^{(1)}0^{(2)}, e_3, \pi_2)$ $(2^{(1)}2^{(2)}, 0^{(1)}0^{(2)}, e_3, 1)$	$\bar{e}_3$	$(1^{(1)}2^{(2)}, 1^{(1)}2^{(2)}, \bar{e}_3, 1)$ $(2^{(1)}2^{(2)}, 2^{(1)}2^{(2)}, \bar{e}_3, 1)$
e_5	$(2^{(1)}0^{(2)}, 0^{(1)}0^{(2)}, e_5, 1)$ $(2^{(1)}1^{(2)}, 0^{(1)}1^{(2)}, e_5, 1)$ $(2^{(1)}2^{(2)}, 0^{(1)}2^{(2)}, e_5, 1)$	$\bar{e}_5$	$(2^{(1)}0^{(2)}, 2^{(1)}0^{(2)}, \bar{e}_5, 1)$ $(2^{(1)}1^{(2)}, 2^{(1)}1^{(2)}, \bar{e}_5, 1)$ $(2^{(1)}2^{(2)}, 2^{(1)}2^{(2)}, \bar{e}_5, 1)$

TAB. 6.25 – Ensemble de chaînons de transition globale $\tilde{\mathcal{T}}_{pot(2^{(1)}2^{(2)})}$ de FIG. 6.14

À partir de ces chaînons, on peut construire les chaînes de transitions globales suivantes (TAB. 6.26) pour l'état global $2^{(1)}2^{(2)}$.

État d'arrivée	Chaîne de transitions globales	Probabilité de transition
$0^{(1)}0^{(2)}$	$\{(2^{(1)}2^{(2)}, 0^{(1)}0^{(2)}, e_3, 1)\}$	ρ_3
$0^{(1)}2^{(2)}$	$\{(2^{(1)}2^{(2)}, 2^{(1)}2^{(2)}, \bar{e}_3, 1); (2^{(1)}2^{(2)}, 0^{(1)}2^{(2)}, e_5, 1)\}$	$(1-\rho_3)\rho_5$
$2^{(1)}2^{(2)}$	$\{(2^{(1)}2^{(2)}, 2^{(1)}2^{(2)}, \bar{e}_3, 1); (2^{(1)}2^{(2)}, 2^{(1)}2^{(2)}, \bar{e}_5, 1)\}$	$(1-\rho_3)(1-\rho_5)$

TAB. 6.26 – Ensemble des chaînes de transitions globales de l'état global $2^{(1)}2^{(2)}$ de FIG. 6.14

FIG. 6.23 présente les chemins de construction de ces chaînes de transitions globales

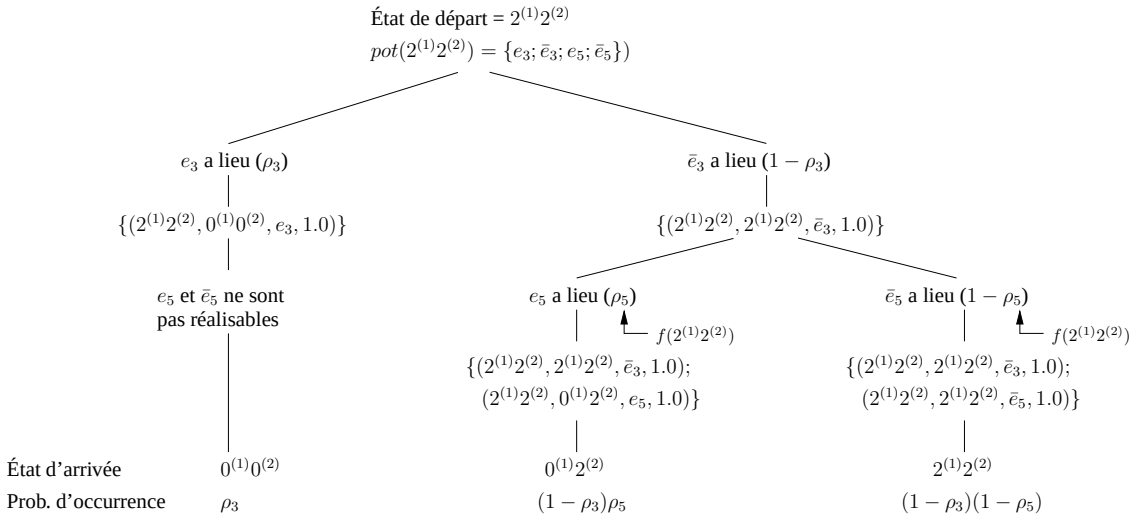


FIG. 6.23 – Construction des chaînes de transitions globales pour l'état $2^{(1)}2^{(2)}$

Avec les probabilités de transitions calculées à partir des chaînes de transitions globales on peut en déduire la chaîne de Markov représentée par le modèle SAN (FIG. 6.24).

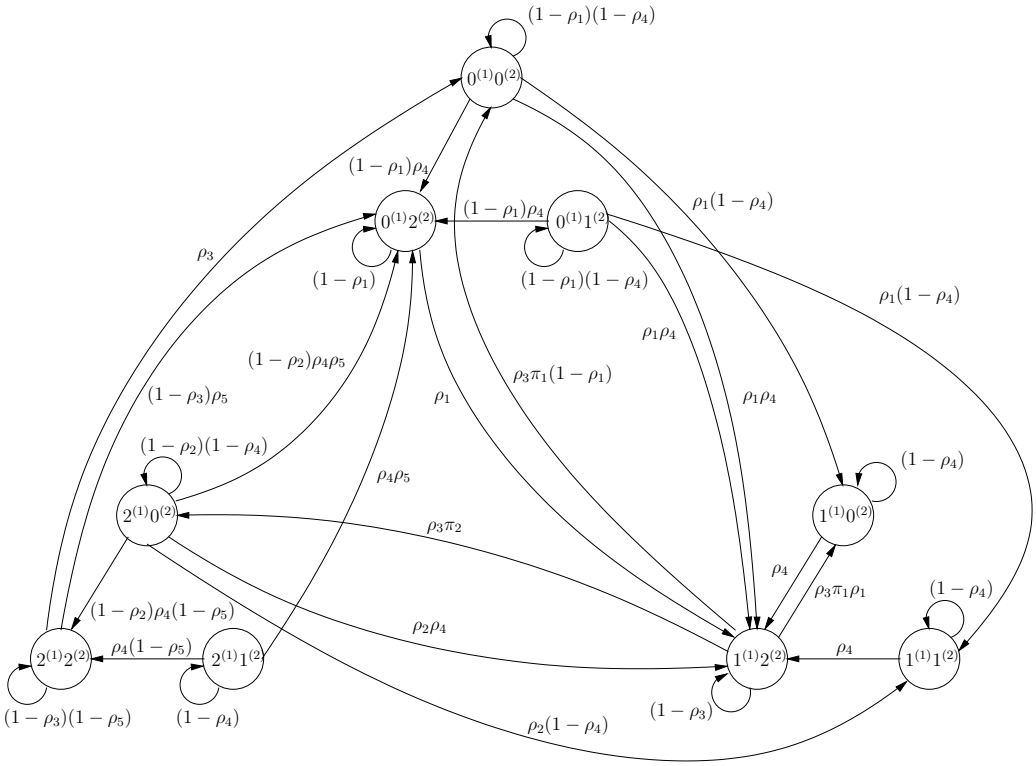


FIG. 6.24 – Chaîne de Markov équivalente au modèle de FIG. 6.11.

Et sa matrice de transition (les états étant dans l'ordre lexicographique).

$$P = \begin{pmatrix}
(1-\rho_1)(1-\rho_4) & 0 & (1-\rho_1)\rho_4 & \rho_1(1-\rho_4) & 0 & \rho_1\rho_4 & 0 & 0 & 0 \\
0 & (1-\rho_1)(1-\rho_4) & (1-\rho_1)\rho_4 & 0 & \rho_1(1-\rho_4) & \rho_1\rho_4 & 0 & 0 & 0 \\
0 & 0 & (1-\rho_1) & 0 & 0 & \rho_1 & 0 & 0 & 0 \\
0 & 0 & 0 & (1-\rho_4) & 0 & \rho_4 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & (1-\rho_4) & \rho_4 & 0 & 0 & 0 \\
\rho_3\pi_1(1-\rho_1) & 0 & 0 & \rho_3\pi_1\rho_1 & 0 & (1-\rho_3) & \rho_3\pi_2 & 0 & 0 \\
0 & 0 & (1-\rho_2)\rho_4\rho_5 & 0 & \rho_2(1-\rho_4) & \rho_2\rho_4 & (1-\rho_2)(1-\rho_4) & 0 & (1-\rho_2)\rho_4(1-\rho_5) \\
0 & 0 & \rho_4\rho_5 & 0 & 0 & 0 & 0 & (1-\rho_4) & \rho_4(1-\rho_5) \\
\rho_3 & 0 & (1-\rho_3)\rho_5 & 0 & 0 & 0 & 0 & 0 & (1-\rho_3)(1-\rho_5)
\end{pmatrix} \tag{6.1}$$

6.5 Conclusion

On a présenté dans ce chapitre une définition formelle pour les réseaux d'automates stochastiques à temps discret.

Tout d'abord, on a défini les notations de base d'un automate et d'un réseau d'automates. À partir de ces définitions de base et de la matrice de transition locale de chaque automate, on a introduit la définition d'automate complété et de réseau d'automates complétés. Un automate complété ou un réseau d'automates complétés inclut tous les transition définies pour un

automate et aussi les transitions qui représentent la *non-occurrence* de chaque événement. La *non-occurrence* d'un événement a été représentée par l'ajout d'un événement complémentaire.

À partir d'un réseau d'automates complétés, on a formalisé l'automate global représenté par le réseau d'automates. L'automate global nous a permis par la suite de définir la matrice de transition de la chaîne de Markov représentée par le modèle SAN. Ceci nous donne la sémantique, en terme de processus aléatoire de l'évolution d'un modèle SAN à temps discret. Ce chapitre clos la partie "définition" de la thèse.

La suite se concentre sur les aspects "représentation tensorielle" du formalisme SAN, plus précisément de la matrice de transition et de la matrice de transition du processus de Markov associé.

Chapitre 7

Algèbre tensorielle complexe (ATX)

Dans ce chapitre[1], on va introduire l'*Algèbre Tensorielle compleXe* (ATX). Notre objectif est de présenter les notations et propriétés de l'ATX afin de proposer une formule tensorielle (voir Chapitre 8) pour le descripteur d'un Réseau d'Automates Stochastiques (SAN) à temps discret.

L'Algèbre Tensorielle Complexe est définie par un opérateur matriciel nommé *produit tensoriel complexe*.

On remarque que les éléments et les couples d'éléments dans le contexte de ce chapitre sont inspirés des notions d'événements et tuples de transitions définis pour les Réseaux d'Automates Stochastiques (SAN) à temps discret (Chapitre 6). Par la suite nous ferons les identifications nécessaires.

Dans ce qui suit, on va donner les notations et propriétés des opérateurs relatifs aux éléments de matrice d'un produit tensoriel complexe (Section 7.1). Dans la section 7.2, on étend les définitions de ces opérateurs et, finalement, dans la section 7.3, on présente les définitions et propriétés du *produit tensoriel complexe* de matrices.

7.1 Opérateurs

Dans cette section, on présente les opérateurs relatifs aux éléments des matrices d'un produit tensoriel complexe. Typiquement, ces éléments sont les événements d'un modèle SAN, et dans la suite on les appellera aussi événements.

Définition 7.1.1. *Soit un ensemble $\breve{\mathcal{E}}$ composé de deux types d'événements :* ***actifs****, notés $e \in \mathcal{E}$, et à tout événement actif est associé un événement* ***complémentaire****, noté $\bar{e} \in \bar{\mathcal{E}}$, où $\breve{\mathcal{E}} = \mathcal{E} \cup \bar{\mathcal{E}}$.*

[1] Les définitions et extensions des opérateurs présentées dans ce chapitre sont communes dans cette thèse et dans [87].

On remarque que les événements *actifs* sont tous les événements e de l'ensemble d'*événements* $\mathcal{E}$ d'un modèle SAN et les événements *complémentaires* sont tous les *événements complémentaires* $\bar{e}$ de l'ensemble d'événements $\bar{\mathcal{E}}$.

On va introduire les définitions de deux événements particuliers de $\breve{\mathcal{E}}$.

Définition 7.1.2. *On définit un événement* ***nul****, noté* $O_{\breve{\mathcal{E}}}$.

Définition 7.1.3. *On définit un événement* ***neutre****, noté* $I_{\breve{\mathcal{E}}}$.

À tout événement *actif* et *complémentaire*, on donne une notion de priorité. Cette notion de priorité adoptée pour les événements actifs e complémentaires est la même que celle employée pour les événements d'un modèle SAN.

Définition 7.1.4. *Soit un événement* $e \in \mathcal{E}$*, on définit une priorité* $\varrho_e \in [1..+\infty[$ *associée à cet événement telle que : quelque soit* $e_1, e_2 \in \mathcal{E}$*,* $e_1 \neq e_2 \Rightarrow \varrho_{e_1} \neq \varrho_{e_2}$.

Définition 7.1.5. *Un événement* $\bar{e} \in \bar{\mathcal{E}}$ *possède la même priorité* ϱ_e *que son événement actif correspondant* $e \in \mathcal{E}$*, i.e., la priorité* $\varrho_{\bar{e}} = \varrho_e$.

Notation : Dans la suite on définit trois opérateurs "$\cdot$", "$+$" et "$*$" sur l'ensemble d'événements $\breve{\mathcal{E}}$. On note $\mathcal{E}^+$ l'ensemble obtenu par toutes les expressions légales de ces opérateurs. $\mathcal{E}^+$ est l'ensemble sur lequel opère ces trois opérateurs.

Dans les sections suivantes, on présente les notations et propriétés des opérateurs qui sont associées aux événements d'ensemble $\breve{\mathcal{E}}$.

7.1.1 Opérateur de simultanéité

Définition 7.1.6. *On définit sur l'ensemble* $\breve{\mathcal{E}}$ *d'événements un opérateur, noté "*$\cdot$*". On appelle cet opérateur de* ***produit de simultanéité*** *et on le définit par :*

1. *Soit* $e_1 \in \breve{\mathcal{E}}, e_2 \in \breve{\mathcal{E}}$*,* $e_1 \cdot e_2$ *est* irréductible *et n'a un sens que si* $\varrho_{e_1} < \varrho_{e_2}$*. avec cette définition,* $e_1 \cdot e_1$ *n'a pas de sens, et* $e_1 \cdot \bar{e}_1$ *n'a pas de sens non plus ;*
2. *Pour l'événement nul :*
   $$\forall e \in \breve{\mathcal{E}}, \qquad e \cdot O_{\breve{\mathcal{E}}} = O_{\breve{\mathcal{E}}}$$
3. *Pour l'événement neutre :*
   $$\forall e \in \breve{\mathcal{E}}, \qquad e \cdot I_{\breve{\mathcal{E}}} = e$$
4. *L'opérateur de simultanéité est un opérateur* **associatif**. *Quels que soient les événements* $e_1, e_2, e_3 \in \breve{\mathcal{E}}$ *tel que* $\varrho_{e_1} < \varrho_{e_2} < \varrho_{e_3}$*, alors*
   $$e_1 \cdot (e_2 \cdot e_3) = (e_1 \cdot e_2) \cdot e_3$$

Dans un modèle SAN, le produit de simultanéité est la représentation de l'occurrence simultanée de plusieurs événements dans une seule unité de temps. $e_1 \cdot e_1$ n'a pas de sens, car

l'événement e_1 ne peut pas avoir lieu simultanément avec lui même. De même, $e_1 \cdot \bar{e}_1$ n'a pas de sens, car l'événement e_1 et son événement complémentaire $\bar{e}_1$ ne peuvent pas avoir lieu simultanément non plus.

Définition 7.1.7. *Soient s événements distincts de $\breve{\mathcal{E}}$, on écrit la forme itérée de l'opérateur de simultanéité, appelé* ***combinaison simultanée*** *(cs), par la notation suivante :*

$$cs = \dot{\bigcup_{i=1}^{s}} e_i = e_1 \cdot e_2 \cdot \ldots \cdot e_s, \tag{7.1}$$

où les priorités sont telles que $\varrho_{e_1} < \varrho_{e_2} < \ldots < \varrho_{e_s}$. On note $\breve{\mathcal{E}}_{cs} \subseteq \breve{\mathcal{E}}$ l'ensemble de facteurs qui apparaissent dans la combinaison simultanée cs.

Définition 7.1.8. *Soit cs la combinaison simultanée $cs = \dot{\bigcup_{i=1}^{s}} e_i = e_1 \cdot e_2 \cdot \ldots \cdot e_s$, on définit cs^{-e_j} la combinaison simultanée cs où l'événement e_j est remplacé par l'événement neutre $I_{\breve{\mathcal{E}}}$, i.e., $cs^{-e_j} = e_1 \cdot \ldots \cdot e_{j-1} \cdot I_{\breve{\mathcal{E}}} \cdot e_{j+1} \cdot \ldots \cdot e_s$.*

7.1.2 Opérateur de choix

Définition 7.1.9. *On définit sur l'ensemble $\breve{\mathcal{E}}$ un opérateur, noté "$+$". On appelle cet opérateur de* ***somme de choix*** *et on le définit par :*

1. $\forall e_1, e_2, \qquad e_1 + e_2$ *a un sens et est irréductible ;*
2. *Pour l'événement nul :*
   $\forall e \in \breve{\mathcal{E}}, \qquad e + O_{\breve{\mathcal{E}}} = e$
3. *Pour l'événement neutre :*
   $\forall e \in \breve{\mathcal{E}}, \qquad e + I_{\breve{\mathcal{E}}}$ *est irréductible*
4. *L'opérateur de choix est un opérateur associatif et quels que soient les événements $e_1, e_2, e_3 \in \breve{\mathcal{E}}$, on a :*
   $e_1 + (e_2 + e_3) = (e_1 + e_2) + e_3$
5. *L'opérateur de choix est un opérateur commutatif et quels que soient les événements $e_1, e_2 \in \breve{\mathcal{E}}$, on a :*
   $e_1 + e_2 = e_2 + e_1$

Dans un modèle SAN, la somme de choix est la représentation de l'occurrence de plusieurs événements qui peuvent déclencher une même transition. Par exemple, soit une transition de l'état x vers l'état y dans un modèle SAN, transition qui peut être déclenchée par l'événement e_1 **ou** par l'événement e_2. Ceci en est représentée par $e_1 + e_2$.

Définition 7.1.10. *Soient c événements de $\breve{\mathcal{E}}$, on écrit la forme itérée de l'opérateur de choix avec la notation suivante :*

$$\dot{+}_{i=1}^{c} e_i = e_1 + e_2 + \ldots + e_c \tag{7.2}$$

7.1.3 Opérateur de concurrence

Définition 7.1.11. *On définit sur l'ensemble $\breve{\mathcal{E}}$ un opérateur, noté "$*$". On appelle cet opérateur de* ***produit de concurrence*** *et on le définit par :*

1. $\forall e_1, e_2 \in \breve{\mathcal{E}}, \quad e_1 * e_2$ *existe et si* $e_1 \neq e_2$*, alors* $e_1 * e_2$ *est* irréductible *;*
2. *Pour l'événement nul :*
   $\forall e \in \breve{\mathcal{E}}, \quad e * O_{\breve{\mathcal{E}}} = O_{\breve{\mathcal{E}}}$
3. *Pour l'événement neutre :*
   $\forall e \in \breve{\mathcal{E}}, \quad e * I_{\breve{\mathcal{E}}} = e$
4. *Soit un événement $e \in \breve{\mathcal{E}}$, on associe un degré d'idempotence entier, noté $\eta_e \in [1.. + \infty[$, tel que l'on ait les propriétés suivantes :*

$$\forall e \in \mathcal{E}, \forall i, 1 \leq i < \eta_e \quad \underbrace{e * e * \ldots * e}_{i \text{ fois}} = O_{\breve{\mathcal{E}}} \tag{7.3}$$

$$\forall e \in \bar{\mathcal{E}}, \forall i, 1 \leq i < \eta_e \quad \underbrace{e * e * \ldots * e}_{i \text{ fois}} = I_{\breve{\mathcal{E}}} \tag{7.4}$$

$$\forall e \in \breve{\mathcal{E}}, i = \eta_e \quad \underbrace{e * e * \ldots * e}_{i \text{ fois}} = e \tag{7.5}$$

$$\forall i, i > \eta_e \quad \underbrace{e * e * \ldots * e}_{i \text{ fois}} \quad \textit{n'a pas de sens} \tag{7.6}$$

5. *L'opérateur de concurrence est un opérateur associatif et quels que soient les événements $e_1, e_2, e_3 \in \breve{\mathcal{E}}$,*
   $e_1 * (e_2 * e_3) = (e_1 * e_2) * e_3$
6. *L'opérateur de concurrence est un opérateur commutatif et quels que soient les événements $e_1, e_2 \in \breve{\mathcal{E}}$,*
   $e_1 * e_2 = e_2 * e_1$
7. *L'opérateur de* concurrence *"$*$" est distributif sur l'opérateur de* choix *"$+$" et quels que soient les événements $e_1, e_2, e_3 \in \breve{\mathcal{E}}$,*
   $e_1 * (e_2 + e_3) = (e_1 * e_2) + (e_1 * e_3)$

Dans un modèle SAN, le produit de concurrence est la représentation de l'occurrence dans la même unité de temps d'événements dans différents automates. Par exemple, $e_1 * e_2$ représente l'occurrence de l'événement e_1 dans un automate **et** l'occurrence de l'événement e_2 dans un autre automate dans la même unité de temps.

Dans la propriété (4) de la définition 7.1.11 sont présentées les propriétés d'un produit de concurrence relatif au degré d'idempotence d'un événement. Le degré d'idempotence sera mis en relation avec le nombre d'automates concernés par un événement dans un modèle SAN. Pour un événement synchronisant concernant η_e automates, celui-ci ne pourra avoir lieu que si $\underbrace{e * e * \ldots * e}_{\eta_e \text{ fois}}$.

Définition 7.1.12. *Soient p événements de $\breve{\mathcal{E}}$, on écrit la forme itérée de l'opérateur de concurrence avec la notation suivante :*

$$\overset{p}{\underset{i=1}{*}} e_i = e_1 * e_2 * \ldots * e_p \tag{7.7}$$

Dans un modèle SAN à temps discret, où plusieurs événements peuvent avoir lieu simultanément dans un même automate, les facteurs d'un produit de concurrence peuvent être des combinaisons simultanées au lieu de "simples" événements de $\breve{\mathcal{E}}$.

Définition 7.1.13. *Soient p combinaisons simultanées $(cs_i)_{i=1}^{p}$, on appelle cc la* ***combinaison concurrente*** *des $(cs_i)_{i=1}^{p}$:*

$$cc = \overset{p}{\underset{i=1}{*}} cs_i = cs_1 * cs_2 * \ldots * cs_p \tag{7.8}$$

Les facteurs d'une combinaison concurrente sont des combinaisons simultanées qui représentent le produit de simultanéité d'événements dans un même automate. On remarque qu'un événement $e_1 \in \breve{\mathcal{E}}$ peut être interprété comme une combinaison simultanée cs d'un seul événement ($s = 1$), *i.e.*, $\breve{\mathcal{E}}_{cs} = \{e_1\}$.

Définition 7.1.14. *Soit un événement $e \in \breve{\mathcal{E}}$ et une combinaison concurrente cc, on définit $O_e(cc)$ le nombre de fois que l'événement e apparaît dans les combinaisons simultanées (cs) de cc.*

Dans le contexte des SAN, soit une combinaison concurrente et un événement, on cherche le nombre de fois que l'identificateur de cet événement apparaît dans les combinaisons simultanées de la combinaison concurrente. On remarque que e apparaît au plus une fois dans chaque cc.

Propriété 7.1.1. *Soit un événement $e \in \breve{\mathcal{E}}$ de degré d'idempotence η_e, une combinaison concurrente $cc = cs_1 * \ldots * cs_p$ avec p combinaisons simultanées et un $O_e(cc)$ donné, alors le produit de concurrence "$*$" a les propriétés suivantes :*

- $$cc = \underbrace{cs_1 * \ldots * cs_p}_{O_e(cc)=\eta_e} = e \cdot (cs'_1 * \ldots * cs'_p) \qquad cs'_i = \begin{cases} cs_i & \text{si } e \notin \breve{\mathcal{E}}_{cs_i} \\ cs_i^{-e} & \text{si } e \in \breve{\mathcal{E}}_{cs_i} \end{cases} \tag{7.9}$$
- $$cc = \underbrace{cs_1 * \ldots * cs_p}_{O_e(cc)<\eta_e} = cs'_1 * \ldots * cs'_p \qquad \text{si } e \in \bar{\mathcal{E}} \text{ et } cs'_i = \begin{cases} cs_i & \text{si } e \notin \breve{\mathcal{E}}_{cs_i} \\ cs_i^{-e} & \text{si } e \in \breve{\mathcal{E}}_{cs_i} \end{cases} \tag{7.10}$$
- $$cc = \underbrace{cs_1 * \ldots * cs_p}_{O_e(cc)<\eta_e} = O_{\breve{\mathcal{E}}} \qquad \text{si } e \in \mathcal{E} \tag{7.11}$$

La propriété 7.1.1 est une extension de la propriété (4) de la définition de l'opérateur "$*$" (Définition 7.1.11), où les facteurs d'un produit de concurrence sont des combinaisons simultanées au lieu de simples événements de $\breve{\mathcal{E}}$.

7.2 Extension de ces opérateurs

Dans cette section, on va étendre les définitions et propriétés des opérateurs présentés dans la section précédente.

Les opérateurs présentés dans la section 7.1 sont appliqués aux éléments de l'ensemble $\breve{\mathcal{E}}$. Dans cette section, au lieu d'utiliser les éléments de $\breve{\mathcal{E}}$, on présente l'extension des opérateurs pour les *couples d'éléments*. Les couples d'éléments sont typiquement les couples de transitions d'un modèle SAN, comprenant un événement et une probabilité de routage. Désormais, on va utiliser *tuples de transitions* pour faire référence aux *couples d'éléments*.

Définition 7.2.1. *Soit $\breve{\mathcal{T}}$ l'ensemble des tuples de transitions $(e, \pi_e(x, y))$, où $e \in \breve{\mathcal{E}}$ et $\pi_e(x, y) \in [0, 1]$ est une probabilité, fonction de x et y, où x et y appartiennent à un espace $\mathcal{S}$.*

Un tuple de transition $(e, \pi_e(x, y))$ représente typiquement une transition de l'état x vers l'état y déclenchée par l'événement e avec la probabilité de routage $\pi_e(x, y)$ dans un automate d'un modèle SAN, x et y appartenant à l'espace d'état $\mathcal{S}$ de l'automate.

Notons qu'un même événement $e \in \breve{\mathcal{E}}$ peut être associé à plusieurs tuples de transitions de $\breve{\mathcal{T}}$, *i.e.*, soit un événement $e \in \breve{\mathcal{E}}$, $(e, \pi_e(x, y)) \in \breve{\mathcal{T}}$ et $(e, \pi_e(z, w)) \in \breve{\mathcal{T}}$ sont deux tuples de transitions distincts de l'événement e si $x \neq z$ ou $y \neq w$.

Définition 7.2.2. *On définit un tuple de transition **nul** $(O_{\breve{\mathcal{E}}}, 0)$, noté $O_{\breve{\mathcal{T}}}$.*

Définition 7.2.3. *On définit un tuple de transition **neutre** $(I_{\breve{\mathcal{E}}}, 1)$, noté $I_{\breve{\mathcal{T}}}$.*

Notation : Dans la suite on définit trois opérateurs "$\cdot$", "$+$" et "$*$" sur l'ensemble de tuples de transitions $\breve{\mathcal{T}}$. On note $\mathcal{T}^+$ l'ensemble obtenu par toutes les expressions légales de ces opérateurs. $\mathcal{T}^+$ est l'ensemble sur lequel opère ces trois opérateurs.

Dans les sections suivantes, on va présenter les notations et propriétés des opérateurs "$\cdot$", "$+$" et "$*$" pour les tuples de transitions de l'ensemble $\breve{\mathcal{T}}$. Ces notations et propriétés sont des extensions des notations et propriétés présentées dans les sections précédentes.

7.2.1 Extension de l'opérateur de simultanéité

Définition 7.2.4. *On définit sur l'ensemble de tuples de transitions $\breve{\mathcal{T}}$ l'opérateur de **simultanéité**, noté "$\cdot$". Cet opérateur est défini par :*

1. *Soit $(e_1, \pi_{e_1}(x_1, y_1)) \in \breve{\mathcal{T}}, (e_2, \pi_{e_2}(x_2, y_2)) \in \breve{\mathcal{T}}$,*
   $(e_1, \pi_{e_1}(x_1, y_1)) \cdot (e_2, \pi_{e_2}(x_2, y_2))$ est irréductible et n'a un sens que si $\varrho_{e_1} < \varrho_{e_2}$ et $y_1 = x_2$. On remarque que $(e_1, \pi_{e_1}(x_1, y_1)) \cdot (e_1, \pi_{e_1}(x_1, y_1))$ n'a pas de sens, et $(e_1, \pi_{e_1}(x_1, y_1)) \cdot (\bar{e}_1, \pi_{\bar{e}_1}(x_1, y_1))$ n'a pas de sens non plus ;

2. *Pour le tuple de transition nul :*
$$\forall(e,\pi_e(x,y)) \in \breve{\mathcal{T}}, \qquad (e,\pi_e(x,y)) \cdot O_{\breve{\mathcal{T}}} = O_{\breve{\mathcal{T}}}$$
3. *Pour le tuple de transition neutre :*
$$\forall(e,\pi_e(x,y)) \in \breve{\mathcal{T}}, \qquad (e,\pi_e(x,y)) \cdot I_{\breve{\mathcal{T}}} = (e,\pi_e(x,y))$$
4. *L'opérateur de simultanéité est un opérateur* **associatif**. *Quels que soient les tuples de transitions* $(e_1,\pi_{e_1}(x_1,y_1)),(e_2,\pi_{e_2}(x_2,y_2)),(e_3,\pi_{e_3}(x_3,y_3)) \in \breve{\mathcal{T}}$ *tel que* $\varrho_{e_1} < \varrho_{e_2} < \varrho_{e_3}$ *et* $y_1 = x_2$ *et* $y_2 = x_3$*, alors*
$$(e_1,\pi_{e_1}(x_1,y_1)) \cdot \Big[(e_2,\pi_{e_2}(x_2,y_2)) \cdot (e_3,\pi_{e_3}(x_3,y_3))\Big] = \Big[(e_1,\pi_{e_1}(x_1,y_1)) \cdot (e_2,\pi_{e_2}(x_2,y_2))\Big] \cdot (e_3,\pi_{e_3}(x_3,y_3))$$

Définition 7.2.5. *Soient* s *tuples de transitions de* $\breve{\mathcal{T}}$*, on écrit la forme itérée de l'opérateur de simultanéité, appelé* ***combinaison simultanée de couples*** *(*csc*), avec la notation suivante :*

$$csc = \overset{s}{\underset{i=1}{\cdot}}(e_i,\pi_{e_i}(x_i,y_i)) = (e_1,\pi_{e_1}(x_1,y_1)) \cdot (e_2,\pi_{e_2}(x_2,y_2)) \cdot \ldots \cdot (e_s,\pi_{e_s}(x_s,y_s)), \quad (7.12)$$

où les priorités sont telle que $\varrho_{e_1} < \varrho_{e_2} < \ldots < \varrho_{e_s}$ *et* $\forall i \in [1..s-1], y_i = x_{i+1}$*. On note* $\breve{\mathcal{T}}_{csc} \subseteq \breve{\mathcal{T}}$ *l'ensemble des facteurs qui apparaissent dans* csc *et* $\breve{\mathcal{E}}_{csc} \subseteq \breve{\mathcal{E}}$ *l'ensemble des événements qui apparaissent dans* csc*.*

Définition 7.2.6. *Soit* csc *la combinaison simultanée de couples* $csc = \overset{s}{\underset{i=1}{\cdot}}(e_i,\pi_{e_i}(x_i,y_i)) = (e_1,\pi_{e_1}(x_1,y_1)) \cdot (e_2,\pi_{e_2}(x_2,y_2)) \cdot \ldots \cdot (e_s,\pi_{e_s}(x_s,y_s))$*, on définit* csc^{-e_j} *la combinaison simultanée de couples* csc *où le tuple de la transition* $(e_j,\pi_{e_j}(x_j,y_j))$ *est remplacé par le tuple de transition neutre* $I_{\breve{\mathcal{T}}}$*, i.e.,* $csc^{-e_j} = (e_1,\pi_{e_1}(x_1,y_1)) \cdot \ldots \cdot (e_{j-1},\pi_{e_{j-1}}(x_{j-1},y_{j-1})) \cdot I_{\breve{\mathcal{T}}} \cdot (e_{j+1},\pi_{e_{j+1}}(x_{j+1},y_{j+1})) \cdot \ldots \cdot (e_s,\pi_{e_s}(x_s,y_s))$*.*

7.2.2 Extension de l'opérateur de choix

Définition 7.2.7. *On définit sur l'ensemble de tuples de transitions* $\breve{\mathcal{T}}$ *l'opérateur de* ***choix****, noté "+". Cet opérateur est défini par :*

1. $\forall(e_1,\pi_{e_1}(x_1,y_1)),(e_2,\pi_{e_2}(x_2,y_2)) \in \breve{\mathcal{T}}$*,*
$(e_1,\pi_{e_1}(x_1,y_1)) + (e_2,\pi_{e_2}(x_2,y_2))$ *a un sens et est irréductible ;*
2. *Pour le tuple de transition nul :*
$$\forall(e,\pi_e(x,y)) \in \breve{\mathcal{T}}, \qquad (e,\pi_e(x,y)) + O_{\breve{\mathcal{T}}} = (e,\pi_e(x,y))$$
3. *Pour le tuple de transition neutre :*
$$\forall(e,\pi_e(x,y)) \in \breve{\mathcal{T}}, \qquad (e,\pi_e(x,y)) + I_{\breve{\mathcal{T}}} \text{ est irréductible}$$
4. *L'opérateur de choix est un opérateur* **associatif**. *Quels que soient les tuples de transitions* $(e_1,\pi_{e_1}(x_1,y_1)),(e_2,\pi_{e_2}(x_2,y_2)),(e_3,\pi_{e_3}(x_3,y_3)) \in \breve{\mathcal{T}}$*, alors*

$$(e_1, \pi_{e_1}(x_1, y_1)) + \Big[(e_2, \pi_{e_2}(x_2, y_2)) + (e_3, \pi_{e_3}(x_3, y_3))\Big] =$$

$$\Big[(e_1, \pi_{e_1}(x_1, y_1)) + (e_2, \pi_{e_2}(x_2, y_2))\Big] + (e_3, \pi_{e_3}(x_3, y_3))$$

5. *L'opérateur de choix est un opérateur* **commutatif**, *quels que soient les tuples de transitions* $(e_1, \pi_{e_1}(x_1, y_1)), (e_2, \pi_{e_2}(x_2, y_2)) \in \breve{\mathcal{T}}$,
$$(e_1, \pi_{e_1}(x_1, y_1)) + (e_2, \pi_{e_2}(x_2, y_2)) = (e_2, \pi_{e_2}(x_2, y_2)) + (e_1, \pi_{e_1}(x_1, y_1))$$

Définition 7.2.8. *Soient c tuples de transitions de $\breve{\mathcal{T}}$, on définit la forme itérée de l'opérateur de choix avec la notation suivante :*

$$\overset{c}{\underset{i=1}{+}}(e_i, \pi_{e_i}(x_i, y_i)) = (e_1, \pi_{e_1}(x_1, y_1)) + (e_2, \pi_{e_2}(x_2, y_2)) + \ldots + (e_c, \pi_{e_c}(x_c, y_c)) \tag{7.13}$$

7.2.3 Extension de l'opérateur de concurrence

Définition 7.2.9. *On définit sur l'ensemble de tuple de transitions $\breve{\mathcal{T}}$ l'opérateur de* ***concurrence****, noté "$*$". Cet opérateur est défini par :*

1. $\forall (e_1, \pi_{e_1}(x_1, y_1)), (e_2, \pi_{e_2}(x_2, y_2)) \in \breve{\mathcal{T}}, \quad (e_1, \pi_{e_1}(x_1, y_1)) * (e_2, \pi_{e_2}(x_2, y_2))$ *existe et si* $e_1 \neq e_2$, *alors* $(e_1, \pi_{e_1}(x_1, y_1)) * (e_2, \pi_{e_2}(x_2, y_2))$ *est* irréductible ;
2. *Pour le tuple de transition nul :*
$$\forall (e, \pi_e(x, y)) \in \breve{\mathcal{T}}, \quad (e, \pi_e(x, y)) * O_{\breve{\mathcal{T}}} = O_{\breve{\mathcal{T}}}$$
3. *Pour le tuple de transition neutre :*
$$\forall (e, \pi_e(x, y)) \in \breve{\mathcal{T}}, \quad (e, \pi_e(x, y)) * I_{\breve{\mathcal{T}}} = (e, \pi_e(x, y))$$
4. *Soit un événement $e \in \breve{\mathcal{E}}$ et n tuples de transitions $(e, \pi_e(x^{(j)}, y^{(j)})) \in \breve{\mathcal{T}}$ (où $j \in [1..n]$ et $x^{(j)}, y^{(j)}$ sont les paramètres de la probabilité du j-ème tuple de transition), on associe un degré d'idempotence entier, noté $\eta_e \in [1.. + \infty[$, tel que on ait les propriétés suivantes :*

$$\forall i, 1 \leq i < \eta_e \quad \underbrace{(e, \pi_e(x^{(1)}, y^{(1)})) * \ldots * (e, \pi_e(x^{(n)}, y^{(n)}))}_{i \text{ fois}} = O_{\breve{\mathcal{T}}} \text{ si } e \in \mathcal{E} \tag{7.14}$$

$$\forall i, 1 \leq i < \eta_e \quad \underbrace{(e, \pi_e(x^{(1)}, y^{(1)})) * \ldots * (e, \pi_e(x^{(n)}, y^{(n)}))}_{i \text{ fois}} = I_{\breve{\mathcal{T}}} \text{ si } e \in \bar{\mathcal{E}} \tag{7.15}$$

$$i = \eta_e \ \underbrace{(e, \pi_e(x^{(1)}, y^{(1)})) * \ldots * (e, \pi_e(x^{(n)}, y^{(n)}))}_{i \text{ fois}} = (e, \prod_{j=1}^{n} \pi_e(x^{(j)}, y^{(j)})) \tag{7.16}$$

$$\forall i > \eta_e \quad \underbrace{(e, \pi_(x^{(1)}, y^{(1)})) * \ldots * (e, \pi_e(x^{(n)}, y^{(n)}))}_{i \text{ fois}} \quad \text{n'a pas de sens} \tag{7.17}$$

5. *L'opérateur de concurrence est un opérateur* **associatif**. *Quels que soient les tuples de transitions $(e_1, \pi_{e_1}(x_1, y_1)), (e_2, \pi_{e_2}(x_2, y_2)), (e_3, \pi_{e_3}(x_3, y_3)) \in \breve{\mathcal{T}}$, alors*
$$(e_1, \pi_{e_1}(x_1, y_1)) * \Big[(e_2, \pi_{e_2}(x_2, y_2)) * (e_3, \pi_{e_3}(x_3, y_3))\Big] =$$

$$\Big[(e_1, \pi_{e_1}(x_1, y_1)) * (e_2, \pi_{e_2}(x_2, y_2))\Big] * (e_3, \pi_{e_3}(x_3, y_3))$$

6. *L'opérateur de concurrence est un opérateur commutatif, quels que soient les tuples de transitions* $(e_1, \pi_{e_1}(x_1, y_1)), (e_2, \pi_{e_2}(x_2, y_2)) \in \breve{\mathcal{T}}$,
$$(e_1, \pi_{e_1}(x_1, y_1)) * (e_2, \pi_{e_2}(x_2, y_2)) = (e_2, \pi_{e_2}(x_2, y_2)) * (e_1, \pi_{e_1}(x_1, y_1))$$
7. *L'opérateur de* concurrence *"$*$" est distributif sur l'opérateur de* choix *"$+$", i.e., soient les tuples de transitions* $(e_1, \pi_{e_1}(x_1, y_1)), (e_2, \pi_{e_2}(x_2, y_2)), (e_3, \pi_{e_3}(x_3, y_3)) \in \breve{\mathcal{T}}$,
$$(e_1, \pi_{e_1}(x_1, y_1)) * \Big[(e_2, \pi_{e_2}(x_2, y_2)) + (e_3, \pi_{e_3}(x_3, y_3))\Big] =$$
$$\Big[(e_1, \pi_{e_1}(x_1, y_1)) * (e_2, \pi_{e_2}(x_2, y_2))\Big] + \Big[(e_1, \pi_{e_1}(x_1, y_1)) * (e_3, \pi_{e_3}(x_3, y_3))\Big]$$

Définition 7.2.10. *Soient p tuples de transitions de $\breve{\mathcal{T}}$, on définit la forme itérée de l'opérateur de concurrence avec la notation suivante :*

$$\overset{p}{\underset{i=1}{*}} (e_i, \pi_{e_i}(x_i, y_i)) = (e_1, \pi_{e_1}(x_1, y_1)) * (e_2, \pi_{e_2}(x_2, y_2)) * \ldots * (e_c, \pi_{e_c}(x_c, y_c)) \tag{7.18}$$

Définition 7.2.11. *Soient p combinaisons simultanées de couples $(csc_i)_{i=1}^{p}$, on appelle ccc la* ***combinaison concurrente de couples*** *des $(csc_i)_{i=1}^{p}$:*

$$ccc = \overset{p}{\underset{i=1}{*}} csc_i = csc_1 * csc_2 * \ldots * csc_p \tag{7.19}$$

Définition 7.2.12. *Soit un événement $e \in \breve{\mathcal{E}}$ et une combinaison concurrente de couples ccc, on définit $O_e(ccc)$ le nombre de fois que l'événement e apparaît dans les combinaisons simultanées de couples (csc) de ccc.*

Propriété 7.2.1. *Soit un événement $e \in \breve{\mathcal{E}}$ de degré d'idempotence η_e, une combinaison concurrente de couples $ccc = csc_1 * \ldots * csc_p$ avec p combinaisons simultanées de couples et un $O_e(ccc)$ donné, alors le produit de concurrence "$*$" a les propriétés suivantes :*

1. $$ccc = \underbrace{csc_1 * \ldots * csc_p}_{O_e(ccc)=\eta_e} = (e, \prod_{i\in[1..p]/e\in\breve{\mathcal{E}}_{csc_i}} \pi_e(x^{(i)}, y^{(i)})) \cdot (csc'_1 * \ldots * csc'_p)$$
$$\text{si } e \in \breve{\mathcal{E}} \text{ et } csc'_i = \begin{cases} csc_i & \text{si } e \notin \breve{\mathcal{E}}_{csc_i} \\ csc_i^{-e} & \text{si } e \in \breve{\mathcal{E}}_{csc_i} \end{cases} \tag{7.20}$$

2. $$ccc = \underbrace{csc_1 * \ldots * csc_p}_{O_e(ccc)<\eta_e} = csc'_1 * \ldots * csc'_p$$
$$\text{si } e \in \bar{\mathcal{E}} \text{ et } csc'_i = \begin{cases} csc_i & \text{si } e \notin \breve{\mathcal{E}}_{csc_i} \\ csc_i^{-e} & \text{si } e \in \breve{\mathcal{E}}_{csc_i} \end{cases} \tag{7.21}$$

3. $$ccc = \underbrace{csc_1 * \ldots * csc_p}_{O_e(ccc)<\eta_e} = O_{\breve{\mathcal{T}}} \quad \text{si } e \in \mathcal{E} \tag{7.22}$$

7.3 Produit Tensoriel Complexe

Dans cette section, on va présenter les définitions et notations pour le *produit tensoriel complexe*. Le produit tensoriel complexe est un opérateur matriciel. Les éléments des matrices qui sont les facteurs d'un produit tensoriel complexe sont des tuples de transitions combinés sur un ensemble $\breve{\mathcal{T}}$.

Définition 7.3.1. *On appelle* **combinaison complexe** *sur $\breve{\mathcal{T}}$ une forme "multi-linéaire" du type :*

$$cx = \overset{n}{\underset{i=1}{+}} \overset{p_i}{\underset{j=1}{\cdot}} (e_{ij}, \pi_{e_{ij}}(x_{ij}, y_{ij}))$$

Par exemple, $(e_1, \pi_{e_1}(x_1, y_1)) \cdot (e_2, \pi_{e_2}(x_2, y_2)) + (e_3, \pi_{e_3}(x_3, y_3)) \cdot (e_4, \pi_{e_4}(x_4, y_4))$ est une combinaison complexe composée par les tuples de transition $(e_1, \pi_{e_1}(x_1, y_1)), (e_2, \pi_{e_2}(x_2, y_2)), (e_3, \pi_{e_3}(x_3, y_3))$ et $(e_4, \pi_{e_4}(x_4, y_4))$.

Définition 7.3.2. *On appelle matrice complexe sur $\breve{\mathcal{T}}$ une matrice dont les éléments sont des combinaisons complexes sur $\breve{\mathcal{T}}$.*

✍ **Soit**

A	une matrice complexe carrée[2] ;
n_A	la dimension de la matrice complexe A ;
a_{ij}	la combinaison complexe de la ligne i et la colonne j de la matrice complexe A ;
$a_{[ik],[jl]}$	la combinaison complexe de la ligne k du i-ème bloc horizontal et la colonne l du j-ème bloc vertical de la matrice complexe A.

Le produit de concurrence de deux matrices complexes A et B de dimensions ($n_A \times n_A$) et ($n_B \times n_B$) respectivement est une matrice de dimensions ($n_A \times n_B$).

Définition 7.3.3. *On appelle produit de concurrence de deux matrices complexes A et B la matrice complexe $C = A * B$ de dimensions $n_A \times n_B$ dont les éléments sont définis par :*

$$\begin{aligned} &i \in [1..n_A], l \in [1..n_B], \\ &c_{il} = \overset{n_A}{\underset{r=1}{+}} (a_{ir} * b_{rl}) \end{aligned} \tag{7.23}$$

Par exemple, soient deux matrices complexes A et B :

[2] On se restreint ici pour nos besoin à des matrices carrées, mais l'opérateur du produit tensoriel complexe peut être utilisé par des matrices non-carrées.

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

Le *produit de concurrence* défini par $C = A * B$ est égal à :

$$C = \left(\begin{array}{c|c} \begin{array}{c} a_{11} * b_{11} \\ + \\ a_{12} * b_{21} \end{array} & \begin{array}{c} a_{11} * b_{12} \\ + \\ a_{12} * b_{22} \end{array} \\ \hline \begin{array}{c} a_{21} * b_{11} \\ + \\ a_{22} * b_{21} \end{array} & \begin{array}{c} a_{21} * b_{12} \\ + \\ a_{22} * b_{22} \end{array} \end{array}\right)$$

Le produit tensoriel complexe de deux matrices complexes A et B de dimensions $(n_A \times n_A)$ et $(n_B \times n_B)$ respectivement est une matrice de dimensions $(n_A n_B \times n_A n_B)$. Cette matrice peut être vue comme une matrice avec $n_A \times n_A$ blocs, chacun de dimension $n_B \times n_B$. La définition de chacun des éléments de la matrice résultante est faite en précisant à quel bloc l'élément appartient et sa position interne dans le bloc.

Définition 7.3.4. *On appelle produit tensoriel complexe de deux matrices complexes A et B la matrice complexe $M = A \circledast B$ de dimension $n_M = n_A \times n_B$ dont les éléments sont définis par :*

$$m_{[ik],[jl]} = a_{ij} * b_{kl} \quad \textit{avec } i \in [1..n_A],\ j \in [1..n_A],\ k \in [1..n_B] \textit{ et } l \in [1..n_B] \tag{7.24}$$

Il est intéressant de remarquer que les éléments de la matrice complexe $M = A \circledast B$ sont des *produits de concurrence* (Définition 7.2.9) des combinaisons complexes des matrices A et B.

Par exemple, soient deux matrices complexes A et B :

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

Le *produit tensoriel complexe* défini par $C = A \circledast B$ est égal à :

$$C = \begin{pmatrix} a_{11} * B & a_{12} * B \\ a_{21} * B & a_{22} * B \end{pmatrix}$$

$$C = \left(\begin{array}{ccc|ccc} a_{11} * b_{11} & a_{11} * b_{12} & a_{11} * b_{13} & a_{12} * b_{11} & a_{12} * b_{12} & a_{12} * b_{13} \\ a_{11} * b_{21} & a_{11} * b_{22} & a_{11} * b_{23} & a_{12} * b_{21} & a_{12} * b_{22} & a_{12} * b_{23} \\ a_{11} * b_{31} & a_{11} * b_{32} & a_{11} * b_{33} & a_{12} * b_{31} & a_{12} * b_{32} & a_{12} * b_{33} \\ \hline a_{21} * b_{11} & a_{21} * b_{12} & a_{21} * b_{13} & a_{22} * b_{11} & a_{22} * b_{12} & a_{22} * b_{13} \\ a_{21} * b_{21} & a_{21} * b_{22} & a_{21} * b_{23} & a_{22} * b_{21} & a_{22} * b_{22} & a_{22} * b_{23} \\ a_{21} * b_{31} & a_{21} * b_{32} & a_{21} * b_{33} & a_{22} * b_{31} & a_{22} * b_{32} & a_{22} * b_{33} \end{array}\right)$$

Dans cet exemple, l'élément c_{53} (*i.e.*, l'élément $a_{21} * b_{23}$) est dans le bloc $(2, 1)$ et sa position interne dans ce bloc est $(2, 3)$.

Définition 7.3.5. *On appelle I_n la matrice identité complexe de dimension n telle que l'élément δ_{ij} (i.e., l'élément dans la ligne i et la colonne j de cette matrice) est égal à l'élément neutre $I_{\check{\mathcal{I}}}$ (si $i = j$) ou égal à l'élément nul $O_{\check{\mathcal{I}}}$ (si $i \neq j$), où $i \in [1..n]$ et $j \in [1..n]$.*

Un cas spécifique de produit tensoriel complexe est le produit tensoriel complexe d'une matrice complexe carrée par une matrice identité complexe. On appelle ce produit de *facteur normal*. Par exemple, soit la matrice complexe A et une matrice identité complexe I_3 :

$$A = \left(\begin{array}{cc} a_{11} & a_{12} \\ a_{21} & a_{22} \end{array}\right) \qquad I_3 = \left(\begin{array}{ccc} I_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} \\ O_{\check{\mathcal{I}}} & I_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} \\ O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & I_{\check{\mathcal{I}}} \end{array}\right)$$

Le facteur normal $A \circledast I_3$ est égal à :

$$\left(\begin{array}{ccc|ccc} a_{11} * I_{\check{\mathcal{I}}} & a_{11} * O_{\check{\mathcal{I}}} & a_{11} * O_{\check{\mathcal{I}}} & a_{12} * I_{\check{\mathcal{I}}} & a_{12} * O_{\check{\mathcal{I}}} & a_{12} * O_{\check{\mathcal{I}}} \\ a_{11} * O_{\check{\mathcal{I}}} & a_{11} * I_{\check{\mathcal{I}}} & a_{11} * O_{\check{\mathcal{I}}} & a_{12} * O_{\check{\mathcal{I}}} & a_{12} * I_{\check{\mathcal{I}}} & a_{12} * O_{\check{\mathcal{I}}} \\ a_{11} * O_{\check{\mathcal{I}}} & a_{11} * O_{\check{\mathcal{I}}} & a_{11} * I_{\check{\mathcal{I}}} & a_{12} * O_{\check{\mathcal{I}}} & a_{12} * O_{\check{\mathcal{I}}} & a_{12} * I_{\check{\mathcal{I}}} \\ \hline a_{21} * I_{\check{\mathcal{I}}} & a_{21} * O_{\check{\mathcal{I}}} & a_{21} * O_{\check{\mathcal{I}}} & a_{22} * I_{\check{\mathcal{I}}} & a_{22} * O_{\check{\mathcal{I}}} & a_{22} * O_{\check{\mathcal{I}}} \\ a_{21} * O_{\check{\mathcal{I}}} & a_{21} * I_{\check{\mathcal{I}}} & a_{21} * O_{\check{\mathcal{I}}} & a_{22} * O_{\check{\mathcal{I}}} & a_{22} * I_{\check{\mathcal{I}}} & a_{22} * O_{\check{\mathcal{I}}} \\ a_{21} * O_{\check{\mathcal{I}}} & a_{21} * O_{\check{\mathcal{I}}} & a_{21} * I_{\check{\mathcal{I}}} & a_{22} * O_{\check{\mathcal{I}}} & a_{22} * O_{\check{\mathcal{I}}} & a_{22} * I_{\check{\mathcal{I}}} \end{array}\right) = \left(\begin{array}{ccc|ccc} a_{11} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & a_{12} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} \\ O_{\check{\mathcal{I}}} & a_{11} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & a_{12} & O_{\check{\mathcal{I}}} \\ O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & a_{11} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & a_{12} \\ \hline a_{21} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & a_{22} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} \\ O_{\check{\mathcal{I}}} & a_{21} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & a_{22} & O_{\check{\mathcal{I}}} \\ O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & a_{21} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & a_{22} \end{array}\right)$$

Le facteur normal $I_3 \circledast A$ est égal à :

$$\left(\begin{array}{ccc|ccc} I_{\check{\mathcal{I}}} * a_{11} & I_{\check{\mathcal{I}}} * a_{12} & O_{\check{\mathcal{I}}} * a_{11} & O_{\check{\mathcal{I}}} * a_{12} & O_{\check{\mathcal{I}}} * a_{11} & O_{\check{\mathcal{I}}} * a_{12} \\ I_{\check{\mathcal{I}}} * a_{21} & I_{\check{\mathcal{I}}} * a_{22} & O_{\check{\mathcal{I}}} * a_{21} & O_{\check{\mathcal{I}}} * a_{22} & O_{\check{\mathcal{I}}} * a_{21} & O_{\check{\mathcal{I}}} * a_{22} \\ \hline O_{\check{\mathcal{I}}} * a_{11} & O_{\check{\mathcal{I}}} * a_{12} & I_{\check{\mathcal{I}}} * a_{11} & I_{\check{\mathcal{I}}} * a_{12} & O_{\check{\mathcal{I}}} * a_{11} & O_{\check{\mathcal{I}}} * a_{12} \\ O_{\check{\mathcal{I}}} * a_{21} & O_{\check{\mathcal{I}}} * a_{22} & I_{\check{\mathcal{I}}} * a_{21} & I_{\check{\mathcal{I}}} * a_{22} & O_{\check{\mathcal{I}}} * a_{21} & O_{\check{\mathcal{I}}} * a_{22} \\ \hline O_{\check{\mathcal{I}}} * a_{11} & O_{\check{\mathcal{I}}} * a_{12} & O_{\check{\mathcal{I}}} * a_{11} & O_{\check{\mathcal{I}}} * a_{12} & I_{\check{\mathcal{I}}} * a_{11} & I_{\check{\mathcal{I}}} * a_{12} \\ O_{\check{\mathcal{I}}} * a_{21} & O_{\check{\mathcal{I}}} * a_{22} & O_{\check{\mathcal{I}}} * a_{21} & O_{\check{\mathcal{I}}} * a_{22} & I_{\check{\mathcal{I}}} * a_{21} & I_{\check{\mathcal{I}}} * a_{22} \end{array}\right) = \left(\begin{array}{ccc|ccc} a_{11} & a_{12} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} \\ a_{21} & a_{22} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} \\ \hline O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & a_{11} & a_{12} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} \\ O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & a_{21} & a_{22} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} \\ \hline O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & a_{11} & a_{12} \\ O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & O_{\check{\mathcal{I}}} & a_{21} & a_{22} \end{array}\right)$$

7.4 Conclusion

Dans ce chapitre, on a présenté l'algèbre tensorielle complexe (ATX) et ses propriétés. Ces propriétés seront utilisées dans le chapitre 8 pour la définition d'une formule tensorielle pour le formalisme des Réseaux d'Automates Stochastiques (SAN) à temps discret.

-

Chapitre 8

Descripteur discret

La modélisation de systèmes complexes sur une échelle de temps discret est telle que plusieurs événements peuvent avoir lieu dans une même unité de temps. Une des grandes difficultés de la modélisation de ces systèmes n'est pas seulement de déterminer toutes les combinaisons possibles de tels événements (dans la même unité de temps), mais aussi de résoudre les cas de conflits lorsque le tirage d'un événement amène à un état où un autre n'est plus réalisable (dans la même unité de temps).

Dans ce chapitre[1], on va proposer une façon compacte de représenter les transitions d'un modèle SAN à temps discret, utilisant une représentation tensorielle complexe dénommée *descripteur discret*. La représentation tensorielle complexe du descripteur discret est basée sur l'algèbre tensorielle complexe définie dans le chapitre 7.

Dans la section 8.1, on va reprendre la définition de l'automate global d'un modèle SAN à temps discret afin de réécrire sa matrice de transition globale $\mathcal{G}$ de façon à préparer la preuve d'égalité entre le descripteur discret et cette matrice. On présente dans la section 8.2 les matrices utilisées pour le descripteur discret. Ensuite, dans la section 8.3, on présente le descripteur discret d'un modèle SAN et la preuve d'égalité avec la matrice de transition de l'automate global de ce modèle. Enfin, dans la section 8.4, on présente un exemple d'un modèle SAN afin d'illustrer l'obtention du descripteur discret du modèle.

8.1 Automate global

Un modèle SAN est une façon modulaire de représenter l'automate global qui décrit le comportement de l'ensemble du réseau d'automates. On considère un modèle SAN tel que défini dans la définition 6.2.16 (page 102).

✍ **Rappelons**

[1] Les définitions et notations nécessaires à définir le descripteur discret sont communes dans cette thèse et dans [87].

$\mathcal{L}(\tilde{x}, \tilde{y})$ l'ensemble de chaînes de transitions globales $g \in \mathcal{L}_{pot(\tilde{x})}$ tel que $\tilde{x}_{1_g} = \tilde{x}$ et $\tilde{y}_{C_g} = \tilde{y}$, où C_g est le nombre de chaînons de transition globale de la chaîne de transitions globales g.

Comme on l'a présenté dans le chapitre 6 (Définition 6.3.4, page 108), soit un SAN complété $(\breve{\mathcal{E}}, \breve{\mathcal{A}}^{(i)}, \mathcal{S}, \mathcal{F})$, où $i \in [1..N]$. Alors, l'automate global $\mathcal{A}$ est défini par :

1. l'ensemble d'états $\mathcal{S} = \prod_{i=1}^{N} \mathcal{S}^{(i)}$;
2. l'ensemble d'événements $\breve{\mathcal{E}}$;
3. la fonction d'atteignabilité $\mathcal{F}$;
4. la matrice de transition globale $\mathcal{G} : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{L}$, telle que :
   $\forall \tilde{x} \in \mathcal{S}, \forall \tilde{y} \in \mathcal{S}$
   $$\mathcal{G}(\tilde{x}, \tilde{y}) = \bigcup_{\forall g \in \mathcal{L}(\tilde{x}, \tilde{y})} \{g\}$$

On rappelle que l'objet mathématique qui décrit l'automate global n'est pas le même que celui d'un automate local d'un modèle SAN. Dans un automate local, les éléments de transition de la matrice de transition contiennent un ensemble des *tuples de transitions locales* $(e, \pi(x, y))$, où l'occurrence de n'importe quel événement de l'élément de transition déclenche la transition. Dans l'automate global, les éléments de transition de la matrice de transition contiennent un ensemble de *chaînes de transitions globales* $\{(\tilde{x}_1, \tilde{y}_1, e_1, \Pi_{e_1}(\tilde{x}_1, \tilde{y}_1)), \ldots, (\tilde{x}_C, \tilde{y}_C, e_C, \Pi_{e_C}(\tilde{x}_C, \tilde{y}_C))\}$, où tous les événements associés aux chaînons de cette chaîne doivent avoir lieu pour que la transition se réalise.

✍ **Notation**

$(e, \Pi_e(\tilde{x}, \tilde{y}))$ le tuple de transition globale de l'état global $\tilde{x}$ vers l'état global $\tilde{y}$ correspondant au chaînon de transition globale $(\tilde{x}, \tilde{y}, e, \Pi_e(\tilde{x}, \tilde{y}))$.

Définition 8.1.1. *Soit une chaîne de transitions globales* $g = \{(\tilde{x}_1, \tilde{y}_1, e_1, \Pi_{e_1}(\tilde{x}_1, \tilde{y}_1)), \ldots, (\tilde{x}_{C_g}, \tilde{y}_{C_g}, e_{C_g}, \Pi_{e_{C_g}}(\tilde{x}_{C_g}, \tilde{y}_{C_g}))\}$ *composée de* C_g *chaînons de transitions globales, on peut représenter cette chaîne par un produit de simultanéité de tuples de transition globale, où un tuple de transition globale est typiquement un couple*[2] *composé par l'identificateur de l'événement du chaînon et de la probabilité de routage globale de ce chaînon. Par extension, on pourra écrire* g *de deux façon différentes, sous forme de chaîne de transitions globales ou sous forme de produit de simultanéité :*

$$g = \{(\tilde{x}_1, \tilde{y}_1, e_1, \Pi_{e_1}(\tilde{x}_1, \tilde{y}_1)), \ldots, (\tilde{x}_{C_g}, \tilde{y}_{C_g}, e_{C_g}, \Pi_{e_{C_g}}(\tilde{x}_{C_g}, \tilde{y}_{C_g}))\} \tag{8.1}$$

où bien

$$g = \overset{C_g}{\underset{i=1}{\cdot}} \big(e_i, \Pi_{e_i}(\tilde{x}_i, \tilde{y}_i)\big) \tag{8.2}$$

[2]On remarque que la représentation d'un chaînon de transition globale par un tuple de transition globale mantient tout les informations nécessaires pour l'enchaînement puisque on conserve les connaissances des paramètres $\tilde{x}_i$ et $\tilde{y}_i$ de $\Pi_{e_i}(\tilde{x}_i, \tilde{y}_i)$.

Pour la démonstration de la preuve d'égalité entre la matrice de transition de l'automate global et le descripteur discret d'un modèle SAN, on va réécrire la matrice de transition $\mathcal{G}$ de l'automate global en utilisant l'algèbre tensorielle complexe comme le permet la définition précédente. Dans cette réécriture, l'*union* des chaînes de transitions globales sera une *somme de choix* de l'algèbre tensorielle complexe et une *chaîne de transitions globales* sera une *combinaison simultanée* de transitions globales, *i.e.* :

$$\begin{aligned} &\forall \tilde{x} \in \mathcal{S}, \forall \tilde{y} \in \mathcal{S} \\ &\mathcal{G}(\tilde{x}, \tilde{y}) = \underset{\forall g \in \mathcal{L}(\tilde{x},\tilde{y})}{+} \overset{C_g}{\underset{i=1}{\cdot}} \Big(e_{i_g}, \Pi_{e_{i_g}}(\tilde{x}_{i_g}, \tilde{y}_{i_g}) \Big) \end{aligned} \tag{8.3}$$

Cette nouvelle vision en utilisant l'algèbre tensorielle complexe sur les éléments de la matrice de transition d'un modèle SAN prépare la démonstration d'égalité dans la section 8.3 entre la matrice de transition de l'automate global et le descripteur discret de ce modèle.

Dans la section suivante, on va donner les définitions et notations des matrices d'événements utilisées pour le descripteur discret.

8.2 Matrices d'événements

Comme on l'a dit précédemment, plusieurs événements peuvent avoir lieu dans une même unité de temps dans un même automate. Les matrices d'événements représentent les possibilités d'occurrence de plusieurs événements à partir d'un état local du modèle. L'occurrence de plusieurs événements dans un même automate et dans une même unité de temps est représenté par une *chaîne de transitions locales*. On reprend maintenant quelques notations utilisées dans le chapitre 6 afin de présenter la définition des chaînes de transitions locales.

✍ **Notation**

$\mathcal{T}^{(i)}$	l'ensemble des tuples de transition de l'automate complété $\breve{\mathcal{A}}^{(i)}$, où $i \in [1..N]$ et $\mathcal{T}^{(i)} \subseteq \mathcal{T}$;
$\mathcal{T}_e^{(i)}$	l'ensemble des tuples de transition de l'automate complété $\breve{\mathcal{A}}^{(i)}$ déclenchés par l'événement e, où $i \in [1..N]$ et $\mathcal{T}_e^{(i)} \subseteq \mathcal{T}^{(i)}$;
$(e, \pi_e(x^{(i)}, y^{(i)}))$	le tuple de transition de $\mathcal{T}_e^{(i)}$ qui représente la transition de l'état local $x^{(i)} \in \mathcal{S}^{(i)}$ vers l'état local $y^{(i)} \in \mathcal{S}^{(i)}$ de l'automate complété $\breve{\mathcal{A}}^{(i)}$ déclenchée par l'événement e avec la probabilité de routage $\pi_e(x^{(i)}, y^{(i)})$;
ϵ	un sous-ensemble d'événements de $\breve{\mathcal{E}}$;

$\mathcal{T}_\epsilon^{(i)}$ l'union des $\mathcal{T}_e^{(i)}$ où $e \in \epsilon$; C'est-à-dire $\mathcal{T}_\epsilon^{(i)} = \underset{e\in\epsilon}{\cup} \mathcal{T}_e^{(i)}$.

Définition 8.2.1. *Soit un ensemble d'évènements ϵ et un automate complété $\breve{\mathcal{A}}^{(i)}$, on appelle* **chaîne de transitions locales** *sur ϵ, une liste ordonnée $\{(e_1, \pi_{e_1}(x_1^{(i)}, y_1^{(i)})), \ldots, (e_C, \pi_{e_C}(x_C^{(i)}, y_C^{(i)}))\}$ composée de C tuples de transition locale de $\mathcal{T}_\epsilon^{(i)}$, qui respectent les règles suivantes :*

1. $\varrho_{e_1} < \varrho_{e_2} < \ldots < \varrho_{e_C}$
2. $\forall j \in [2..C]$
   $x_j^{(i)} = y_{j-1}^{(i)}$
3. $\nexists (e, \pi_e(x^{(i)}, y^{(i)})) \in \mathcal{T}_\epsilon^{(i)}$ *tel que*
   $\varrho_e < \varrho_{e_1}$ *et* $y^{(i)} = x_1^{(i)}$
4. $\nexists (e, \pi_e(x^{(i)}, y^{(i)})) \in \mathcal{T}_\epsilon^{(i)}$ *tel que*
   $\varrho_{e_C} < \varrho_e$ *et* $x^{(i)} = y_C^{(i)}$
5. $\forall j \in [1..C-1], \nexists (e, \pi_e(x^{(i)}, y^{(i)})) \in \mathcal{T}_\epsilon^{(i)}$ *tel que*
   $\varrho_{e_j} < \varrho_e < \varrho_{e_{j+1}}$ *et* $y_j^{(i)} = x^{(i)}$

Les chaînes de transitions locales d'intérêt sont les chaînes qui prennent leurs événements dans un ensemble $\epsilon = pot(x^{(i)})$ et qui réalisent un enchaînement maximal de tuples de transitions locales des événements potentiellement réalisables à partir d'un état $x^{(i)}$.

De façon similaire aux chaînes de transitions globales, on peut représenter une chaîne de transitions locales par un produit de simultanéité de ses tuples de transitions locales.

Définition 8.2.2. *Soit une chaîne de transitions locales $l = \{(e_1, \pi_{e_1}(x_1^{(i)}, y_1^{(i)})), \ldots, (e_{C_l}, \pi_{e_C}(x_{C_l}^{(i)}, y_{C_l}^{(i)}))\}$ composée de C_l tuples de transitions locales de l'automate complété $\breve{\mathcal{A}}^{(i)}$, on peut représenter cette chaîne par un produit de simultanéité de ses tuples de transition. Alors, la chaîne de transitions locales l peut être exprimée par une chaîne de transitions locales ou par un produit de simultanéité :*

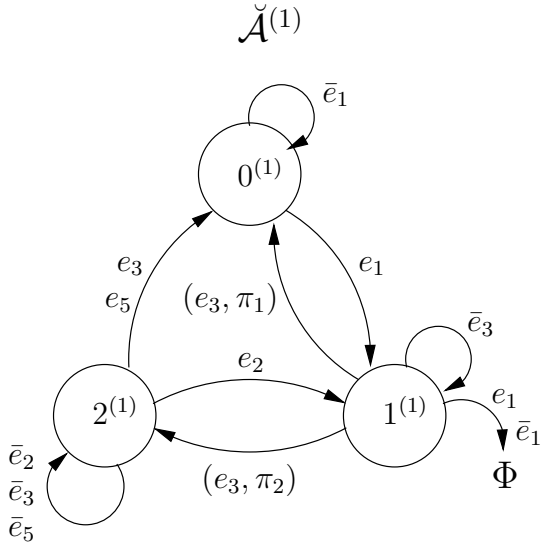
$$l = \{(e_1, \pi_{e_1}(x_1^{(i)}, y_1^{(i)})), \ldots, (e_{C_l}, \pi_{e_C}(x_{C_l}^{(i)}, y_{C_l}^{(i)}))\} \tag{8.4}$$

ou bien,

$$l = \overset{C_l}{\underset{j=1}{\cdot}} (e_j, \pi_{e_j}(x_j^{(i)}, y_j^{(i)})) \tag{8.5}$$

Dans FIG. 8.1, on présente un exemple d'un automate complété $\breve{\mathcal{A}}^{(1)}$. Cet automate possède trois états, quatre événements (e_1, e_2, e_3, et e_5) et quatre événements complémentaires ($\bar{e}_1$, $\bar{e}_2$, $\bar{e}_3$, et $\bar{e}_5$).

Pour l'automate complété $\breve{\mathcal{A}}^{(1)}$ présenté dans FIG. 8.1, pour chaque état local $x^{(i)} \in \mathcal{S}^{(i)}$ de l'automate, on a l'ensemble d'événements $\epsilon = pot(x^{(i)})$ (voir Définition 6.2.8). À partir de



Événement	Prob.	Priorité
e_1	ρ_1	3
e_2	ρ_2	1
e_3	ρ_3	2
e_5	ρ_5	5

Événement	Prob.	Priorité
$\bar{e}_1$	$1-\rho_1$	3
$\bar{e}_2$	$1-\rho_2$	1
$\bar{e}_3$	$1-\rho_3$	2
$\bar{e}_5$	$1-\rho_5$	5

FIG. 8.1 – Exemple d'un automate complété

l'ensemble d'événements ϵ relatif à chaque état de l'automate complété $\breve{\mathcal{A}}^{(1)}$, on peut obtenir pour cet automate les chaînes de transitions locales présentées dans TAB. 8.1.

Utilisant les chaînes de transitions locales, pour chaque automate complété $\breve{\mathcal{A}}^{(i)}$ d'un modèle SAN, on peut obtenir une *matrice d'événements* $P^{(i)}$. **Les éléments d'une matrice d'événements $P^{(i)}$ sont des produits de simultanéité des tuples de transition des chaînes de transitions locales**.

✍ **Notation**

$\mathcal{L}_{\epsilon}^{(i)}$ l'ensemble des chaînes de transitions locales de l'ensemble d'événements ϵ de l'automate complété $\breve{\mathcal{A}}^{(i)}$. On s'intéressera typiquement à un ϵ de type $pot(x^{(i)})$;

$\mathcal{L}^{(i)}(x^{(i)}, y^{(i)})$ l'ensemble de chaînes de transitions locales $l \in \mathcal{L}^{(i)}_{pot(x^{(i)})}$ tel que $x^{(i)}_{1_l} = x^{(i)}$ et $y^{(i)}_{C_l} = y^{(i)}$, où C_l est le nombre de tuples de transition locale de la chaîne de transitions locales l ;

e_{j_l} l'événement du j-ème tuple de transition de la chaîne de transitions locales l ;

$\pi_{e_{j_l}}(x^{(i)}_{j_l}, y^{(i)}_{j_l})$ la probabilité de routage pour la transition de l'état local $x^{(i)}_{j_l} \in \mathcal{S}^{(i)}$ vers l'état local $y^{(i)}_{j_l} \in \mathcal{S}^{(i)}$ de l'automate complété $\breve{\mathcal{A}}^{(i)}$ déclenchée par l'événement e_{j_l} du j-ème tuple de transition de la chaîne de transitions locales l.

Définition 8.2.3. *Les éléments de la matrice locale d'événements $P^{(i)}$ de l'automate complété*

État de départ $0^{(1)}$; $\epsilon = pot(0^{(1)}) = \{e_1, \bar{e}_1\}$	
État d'arrivée	Chaînes de transitions locales
$0^{(1)}$	$\{(\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)}))\}$
$1^{(1)}$	$\{(e_1, \pi_{e_1}(0^{(1)}, 1^{(1)}))\}$

État de départ $1^{(1)}$; $\epsilon = pot(1^{(1)}) = \{e_3, \bar{e}_3, e_1, \bar{e}_1\}$	
État d'arrivée	Chaînes de transitions locales
$0^{(1)}$	$\{(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})); (\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)}))\}$
$1^{(1)}$	$\{(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})); (e_1, \pi_{e_1}(0^{(1)}, 1^{(1)}))\}$
$1^{(1)}$	$\{(\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)}))\}$
$2^{(1)}$	$\{(e_3, \pi_{e_3}(1^{(1)}, 2^{(1)}))\}$

État de départ $2^{(1)}$; $\epsilon = pot(2^{(1)}) = \{e_2, \bar{e}_2, e_3, \bar{e}_3, e_5, \bar{e}_5\}$	
État d'arrivée	Chaînes de transitions locales
$0^{(1)}$	$\{(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})); (e_3, \pi_{e_3}(1^{(1)}, 0^{(1)}))\}$
$0^{(1)}$	$\{(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})); (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})); (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)}))\}$
$0^{(1)}$	$\{(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})); (e_3, \pi_{e_3}(2^{(1)}, 0^{(1)}))\}$
$0^{(1)}$	$\{(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})); (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})); (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)}))\}$
$1^{(1)}$	$\{(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})); (\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)}))\}$
$2^{(1)}$	$\{(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})); (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})); (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)}))\}$
$2^{(1)}$	$\{(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})); (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})); (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)}))\}$

TAB. 8.1 – Les chaînes de transitions locales de l'automate complété $\breve{\mathcal{A}}^{(1)}$ de FIG. 8.1

$\breve{\mathcal{A}}^{(i)}$ sont définis par :

$$\forall x^{(i)} \in \mathcal{S}^{(i)}, \forall y^{(i)} \in \mathcal{S}^{(i)},$$

$$P^{(i)}(x^{(i)}, y^{(i)}) = \underset{\forall l \in \mathcal{L}^{(i)}(x^{(i)}, y^{(i)})}{+} \overset{C_l}{\underset{j=1}{\cdot}} \left(e_{j_l}, \pi_{e_{j_l}}(x^{(i)}_{j_l}, y^{(i)}_{j_l}) \right)$$

Cette définition exprime que la transition de $x^{(i)}$ vers $y^{(i)}$, peut être réalisée par l'une quelconque des chaînes de $\mathcal{L}^{(i)}_{pot(x^{(i)})}$ qui amènent à $y^{(i)}$, c'est-à-dire de $\mathcal{L}^{(i)}(x^{(i)}, y^{(i)})$. Une chaîne de transitions locales est représentée par un *produit de simultanéité* ("$\cdot$") des C tuples de transition de cette chaîne. L'élément de la ligne $x^{(i)}$ et colonne $y^{(i)}$ de la matrice d'événements $P^{(i)}$ est une *somme de choix* ("$+$") de toutes les chaînes de transitions locales dont l'état de départ du premier tuple de transition locale ($x^{(i)}_1$) est égal à l'état $x^{(i)}$ et l'état d'arrivée du dernier tuple de transition locale ($y^{(i)}_C$) est égal à l'état $y^{(i)}$.

Pour l'automate complété $\breve{\mathcal{A}}^{(1)}$ présenté dans FIG. 8.1, à partir des chaînes de transitions locales de TAB. 8.1, on peut obtenir la matrice d'événements $P^{(1)}$:

$$P^{(1)} = \left(\begin{array}{c|c|c} (\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) & (e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) & O_{\tilde{T}} \\ \hline (e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) & \begin{array}{c}(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) \\ + \\ (\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)}))\end{array} & (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \\ \hline \begin{array}{c}(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \\ + \\ (e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) \\ + \\ (\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (e_3, \pi_{e_3}(2^{(1)}, 0^{(1)})) \\ + \\ (\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)}))\end{array} & (e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) & \begin{array}{c}(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) \\ + \\ (e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)}))\end{array} \end{array}\right)$$

8.2.1 Relations entre les chaînes de transitions globales et les chaînes de transitions locales

Nous allons maintenant définir les relations entre les chaînes de transitions globales et les chaînes de transitions locales. Pour exprimer ces relations, nous proposons deux lemmes. Le lemme 8.2.1 montre qu'à partir des chaînes de transitions locales $l^{(1)}, \ldots, l^{(N)}$ on peut obtenir une chaîne de transitions globales g par le produit de concurrence de ces chaînes de transitions locales. De façon analogue, le lemme 8.2.2 montre qu'une chaîne de transitions globales g peut être factorisée en un produit de concurrence de chaînes de transitions locales $l^{(1)}, \ldots, l^{(N)}$.

✍ **Rappelons**

$csc_{l^{(i)}}$ le produit de simultanéité de couples, appelé *combinaison simultanée de couples* (csc) (Définition 7.2.5, page 127), qui représente la chaîne de transitions locales $l^{(i)}$;

$\breve{\mathcal{E}}_{csc_{l^{(i)}}}$ l'ensemble des événements qui apparaissent dans $csc_{l^{(i)}}$ (Définition 7.2.5, 127) ;

$csc_{l^{(i)}}^{-e}$ la combinaison simultanée de couples $csc_{l^{(i)}}$ où le tuple de transition de l'événement e a été remplacé par le tuple de transition neutre (Définition 7.2.6, page 127) ;

$ccc_{[l^{(1)}..l^{(N)}]}$ le produit de concurrence de $csc_{l^{(i)}}$, où $i \in [1..N]$, appelé *combinaison concurrente de couples* (ccc) (Définition 7.2.11, page 129) ;

$\mathcal{L}^{(i)}(x^{(i)}, y^{(i)})$ l'ensemble de chaînes de transitions locales $l \in \mathcal{L}^{(i)}_{pot(x^{(i)})}$ tel que $x^{(i)}_{1_l} = x^{(i)}$ et $y^{(i)}_{C_l} = y^{(i)}$, où C_l est le nombre de tuples de transition locale de la chaîne de transitions locales l ;

$\mathcal{L}(\tilde{x}, \tilde{y})$ l'ensemble de chaînes de transitions globales $g \in \mathcal{L}_{pot(\tilde{x})}$ tel que $\tilde{x}_{1_g} = \tilde{x}$

et $\tilde{y}_{C_g} = \tilde{y}$, où C_g est le nombre de chaînons de transition globale de la chaîne de transitions globales g.

Le lemme suivant montre que, étant donné un "jeu" de chaînes de transitions locales $[l^{(1)}, \ldots, l^{(N)}]$ obtenues à partir des états locaux de $(x^{(1)}, \ldots, x^{(N)})$ et $(y^{(1)}, \ldots, y^{(N)})$, si le produit de concurrence de ces chaînes de transitions locales n'est pas nul, alors on a une chaîne de transitions globales g obtenue à partir de ce produit et $g \in \mathcal{L}(\tilde{x}, \tilde{y})$.

Lemme 8.2.1. *Soient les états locaux $(x^{(1)}, \ldots, x^{(N)})$ et $(y^{(1)}, \ldots, y^{(N)})$ d'un* SAN *complété $(\check{\mathcal{E}}, \check{\mathcal{A}}^{(i)}, \mathcal{S}, \mathcal{F})$, où $i \in [1..N]$, et des chaînes de transitions locales $l^{(1)} \in \mathcal{L}^{(1)}(x^{(1)}, y^{(1)})$, ..., $l^{(N)} \in \mathcal{L}^{(N)}(x^{(N)}, y^{(N)})$, si $\underset{i=1}{\overset{N}{*}} l^{(i)} \neq O_{\check{\mathcal{T}}}$ alors $g = \underset{i=1}{\overset{N}{*}} l^{(i)}$ est une chaîne de transitions globales telle que $g \in \mathcal{L}(\tilde{x}, \tilde{y})$.*

Démonstration. Selon la définition 8.2.2 (page 138), on peut réécrire le produit de concurrence des chaînes de transitions locales $l^{(i)}$ de la façon suivante :

$$\underset{i=1}{\overset{N}{*}} l^{(i)} = \underbrace{\overbrace{\underset{i=1}{\overset{N}{*}} \left(\overset{C_{l^{(i)}}}{\underset{j=1}{\cdot}} \left(e_{j_{l^{(i)}}}, \pi_{e_{j_{l^{(i)}}}} (x^{(i)}_{j_{l^{(i)}}}, y^{(i)}_{j_{l^{(i)}}}) \right) \right)}^{csc_{l^{(i)}}}}_{ccc_{[l^{(1)}..l^{(N)}]}} \tag{8.6}$$

Selon la propriété d'idempotence (Propriété 7.2.1, page 129), si $ccc_{[l^{(1)}..l^{(N)}]} \neq O_{\check{\mathcal{T}}}$ (*i.e.*, la combinaison concurrente de couples ccc composée des $csc_{l^{(i)}}$ est différente de $O_{\check{\mathcal{T}}}$ - item 3 de la propriété d'idempotence), alors, pour chaque événement qui apparaît dans $ccc_{[l^{(1)}..l^{(N)}]}$, on a deux cas possibles :

- (item 1 de la propriété d'idempotence) - les tuples de transition locale d'un événement e dans les $csc_{l^{(i)}}$, où $i \in [1..N]$, respectent le degré d'idempotence de cet événement et ils sont factorisés en un seul tuple de transition globale $(e, \Pi_e(\tilde{x}, \tilde{y}))$, où la probabilité de routage $\Pi_e(\tilde{x}, \tilde{y})$ est obtenue par le produit des probabilités de routage de chaque tuple de transition locale qui apparaît dans les $csc_{l^{(i)}}$ de $ccc_{[l^{(1)}..l^{(N)}]}$, *i.e.*, $\Pi_e(\tilde{x}, \tilde{y}) = \prod_{i \in [1..N] / i \in \check{\mathcal{E}}_{csc_{l^{(i)}}}} \pi_e(x^{(i)}, y^{(i)})$;
- (item 2 de la propriété d'idempotence) - les tuples de transition locale d'un événement e dans les $csc_{l^{(i)}}$, où $i \in [1..N]$, ne respectent pas le degré d'idempotence de cet événement. Cet événement est alors un événement complémentaire et ses tuples de transition locale sont remplacés par le *tuple de transition neutre* (Définition 7.2.3, page 126). Par la propriété 3 de la définition de l'opérateur de simultanéité (Définition 7.2.4, page 126), ce tuple de transition neutre va donc disparaître dans le produit de simultanéité.

Pour la construction des chaînes de transitions globales, on s'intéresse uniquement aux événements qui ont leur degré d'idempotence respecté.

On va maintenant factoriser les événements des $csc_{l^{(i)}}$ qui composent la $ccc_{[l^{(1)}..l^{(N)}]}$ de

l'équation (8.6). On note par $\breve{\mathcal{E}}_{ccc_{[l^{(1)}..l^{(N)}]}}$ l'ensemble d'événements qui apparaissent dans toutes les $csc_{l^{(i)}}$ qui composent la $ccc_{[l^{(1)}..l^{(N)}]}$, c'est-à-dire, $\breve{\mathcal{E}}_{ccc_{[l^{(1)}..l^{(N)}]}} = \breve{\mathcal{E}}_{csc_{l^{(1)}}} \cup \ldots \cup \breve{\mathcal{E}}_{csc_{l^{(N)}}}$.

En prenant l'événement e, le plus prioritaire de l'ensemble d'événements $\breve{\mathcal{E}}_{ccc_{[l^{(1)}..l^{(N)}]}}$ tel que $O_e(ccc_{[l^{(1)}..l^{(N)}]}) = \eta_e$, et en le factorisant, on peut réécrire l'équation (8.6) par :

$$\underbrace{\overset{N}{\underset{i=1}{*}} \overbrace{\left(\overset{C_{l^{(i)}}}{\underset{j=1}{\cdot}} \left(e_{j_{l^{(i)}}}, \pi_{e_{j_{l^{(i)}}}}(x^{(i)}_{j_{l^{(i)}}}, y^{(i)}_{j_{l^{(i)}}})\right)\right)}^{csc_{l^{(i)}}}}_{ccc_{[l^{(1)}..l^{(N)}]}} = \left(e, \prod_{k\in[1..N]/e\in\breve{\mathcal{E}}_{csc_k}} \pi_e(x^{(k)}_e, y^{(k)}_e)\right) \cdot \left(\overset{N}{\underset{i=1}{*}} csc'_{l^{(i)}}\right), \tag{8.7}$$

où $csc'_{l^{(i)}} = csc_{l^{(i)}}$ si $e \notin \breve{\mathcal{E}}_{csc_{l^{(i)}}}$ ou $csc'_{l^{(i)}} = csc^{-e}_{l^{(i)}}$ si $e \in \breve{\mathcal{E}}_{csc_{l^{(i)}}}$.

On remarque que, par la définition de $csc^{-e}_{l^{(i)}}$ (Définition 7.2.6, page 127), le tuple de transition locale de l'événement e est remplacé par un tuple de transition neutre ($I_{\breve{\mathcal{T}}}$).

Notons aussi que $\prod_{k\in[1..N]/e\in\breve{\mathcal{E}}_{csc_k}} \pi_e(x^{(k)}_e, y^{(k)}_e)$ est égal à la probabilité de routage globale $\Pi_e(\tilde{x}_e, \tilde{y}_e)$ du chaînon de transition globale $(\tilde{x}, \tilde{y}, e, \Pi_e(\tilde{x}_e, \tilde{y}_e))$ (Définition 6.3.1, page 104). De cette façon, on peut réécrire l'équation (8.7) :

$$\underbrace{\overset{N}{\underset{i=1}{*}} \overbrace{\left(\overset{C_{l^{(i)}}}{\underset{j=1}{\cdot}} \left(e_{j_{l^{(i)}}}, \pi_{e_{j_{l^{(i)}}}}(x^{(i)}_{j_{l^{(i)}}}, y^{(i)}_{j_{l^{(i)}}})\right)\right)}^{csc_{l^{(i)}}}}_{ccc_{[l^{(1)}..l^{(N)}]}} = (e, \Pi_e(\tilde{x}_e, \tilde{y}_e)) \cdot \left(\overset{N}{\underset{i=1}{*}} csc'_{l^{(i)}}\right) \tag{8.8}$$

Prenons maintenant l'événement e_2, le deuxième événement le plus prioritaire, de l'ensemble d'événements $\breve{\mathcal{E}}_{ccc_{[l^{(1)}..l^{(N)}]}}$ tel que $O_{e_2}(ccc_{[l^{(1)}..l^{(N)}]}) = \eta_{e_2}$. En factorisant cet événement, on peut réécrire l'équation (8.8) :

$$\underbrace{\overset{N}{\underset{i=1}{*}} \overbrace{\left(\overset{C_{l^{(i)}}}{\underset{j=1}{\cdot}} \left(e_{j_{l^{(i)}}}, \pi_{e_{j_{l^{(i)}}}}(x^{(i)}_{j_{l^{(i)}}}, y^{(i)}_{j_{l^{(i)}}})\right)\right)}^{csc_{l^{(i)}}}}_{ccc_{[l^{(1)}..l^{(N)}]}} = (e, \Pi_e(\tilde{x}_e, \tilde{y}_e)) \cdot (e_2, \Pi_{e_2}(\tilde{x}_{e_2}, \tilde{y}_{e_2})) \cdot \left(\overset{N}{\underset{i=1}{*}} csc''_{l^{(i)}}\right) \tag{8.9}$$

où $csc''_{l^{(i)}} = csc'_{l^{(i)}}$ si $e \notin \breve{\mathcal{E}}_{csc'_{l^{(i)}}}$ ou $csc''_{l^{(i)}} = csc'^{-e}_{l^{(i)}}$ si $e \in \breve{\mathcal{E}}_{csc'_{l^{(i)}}}$.

En répétant cette factorisation pour tout événement e de l'ensemble d'événements $\breve{\mathcal{E}}_{ccc_{[l^{(1)}..l^{(N)}]}}$ tel que $O_e(ccc_{[l^{(1)}..l^{(N)}]}) = \eta_e$, en respectant l'ordre de priorité des événements, on peut réécrire

l'équation (8.9) :

$$\underbrace{\overset{N}{\underset{i=1}{*}}\overbrace{\left(\overset{C_{l^{(i)}}}{\underset{j=1}{\cdot}}\left(e_{j_{l^{(i)}}},\pi_{e_{j_{l^{(i)}}}}(x^{(i)}_{j_{l^{(i)}}},y^{(i)}_{j_{l^{(i)}}})\right)\right)}^{csc_{l^{(i)}}}}_{ccc_{[l^{(1)}..l^{(N)}]}}=\underset{\substack{\forall e\in\breve{\mathcal{E}}_{ccc_{[l^{(1)}..l^{(N)}]}}/\\ O_e(ccc_{[l^{(1)}..l^{(N)}]})=\eta_e}}{\cdot}(e,\Pi_e(\tilde{x}_e,\tilde{y}_e))\cdot\left(\overset{N}{\underset{i=1}{*}}I_{\breve{\mathcal{T}}}\right) \tag{8.10}$$

On remarque que, en factorisant tous les événements de $\breve{\mathcal{E}}_{ccc_{[l^{(1)}..l^{(N)}]}}$, tous les tuples de transition locale de ces événements dans les $csc_{l^{(i)}}$ sont remplacés par $I_{\breve{\mathcal{T}}}$, ce qui donne $\overset{N}{\underset{i=1}{*}}I_{\breve{\mathcal{T}}}$. Par la propriété 3 des définitions de l'opérateur de simultanéité (Définition 7.2.4, page 126) et de l'opérateur de concurrence (Définition 7.2.9, page 128), ces tuples de transition neutre vont disparaître et on peut réécrire l'équation (8.10) :

$$\underbrace{\overset{N}{\underset{i=1}{*}}\overbrace{\left(\overset{C_{l^{(i)}}}{\underset{j=1}{\cdot}}\left(e_{j_{l^{(i)}}},\pi_{e_{j_{l^{(i)}}}}(x^{(i)}_{j_{l^{(i)}}},y^{(i)}_{j_{l^{(i)}}})\right)\right)}^{csc_{l^{(i)}}}}_{ccc_{[l^{(1)}..l^{(N)}]}}=\underset{\substack{\forall e\in\breve{\mathcal{E}}_{ccc_{[l^{(1)}..l^{(N)}]}}/\\ O_e(ccc_{[l^{(1)}..l^{(N)}]})=\eta_e}}{\cdot}(e,\Pi_e(\tilde{x}_e,\tilde{y}_e)) \tag{8.11}$$

Pour que le produit de simultanéité du membre droit de l'équation (8.11) soit une chaîne de transitions globales de $\epsilon=\mathcal{L}(\tilde{x},\tilde{y})$, il faut assurer qu'il respecte les règles de construction d'une chaîne de transitions globales (Définition 6.3.2, page 105). Rappelons cette définition :

Soit un ensemble d'évènements ϵ, on appelle **chaîne de transitions globales**, *une liste ordonnée $\{(\tilde{x}_1,\tilde{y}_1,e_1,\Pi_{e_1}(\tilde{x}_1,\tilde{y}_1)),\ldots,(\tilde{x}_C,\tilde{y}_C,e_C,\Pi_{e_C}(\tilde{x}_C,\tilde{y}_C))\}$ composée des C chaînons de transition globale de $\tilde{\mathcal{T}}_\epsilon$, qui respecte les règles suivantes :*

1. $\varrho_{e_1}<\varrho_{e_2}<\ldots<\varrho_{e_C}$
2. $\forall i\in[2..C]$
   $\tilde{x}_i=\tilde{y}_{i-1}$
3. $\nexists(\tilde{x},\tilde{y},e,\Pi_e(\tilde{x},\tilde{y}))\in\tilde{\mathcal{T}}_\epsilon$ *tel que*
   $\varrho_e<\varrho_{e_1}$ *et* $\tilde{y}=\tilde{x}_1$
4. $\nexists(\tilde{x},\tilde{y},e,\Pi_e(\tilde{x},\tilde{y}))\in\tilde{\mathcal{T}}_\epsilon$ *tel que*
   $\varrho_{e_C}<\varrho_e$ *et* $\tilde{x}=\tilde{y}_C$
5. $\forall i\in[1..C-1],\nexists(\tilde{x},\tilde{y},e,\Pi_e(\tilde{x},\tilde{y}))\in\tilde{\mathcal{T}}_\epsilon$ *tel que*
   $\varrho_{e_i}<\varrho_e<\varrho_{e_{i+1}}$ *et* $\tilde{y}_i=\tilde{x}$

On va montrer que, pour l'ensemble d'événements $\epsilon=pot(\tilde{x})$, le produit de simultanéité des tuples de transition globale du membre droit de l'équation (8.11) est une chaîne de transitions globales $g\in\mathcal{L}(\tilde{x},\tilde{y})$ car il respecte les règles présentées précédemment.

La règle 1 établit l'ordre d'enchaînement de tuples de transition globale. Par l'ordre de factorisation des événements qu'on vient de faire (du plus prioritaire au moins prioritaire), les tuples de transitions globales obtenus par cette factorisation respectent cette règle.

La règle 2 établit l'enchaînement de tuples de transition globale. Autrement dit, il faut que l'état d'arrivée d'un tuple de transition globale soit égal à l'état de départ du tuple de transition globale suivant. Pour montrer l'enchaînement des tuples de transition globale, on va montrer l'enchaînement de chaque composante des états globaux obtenus par la factorisation des tuples de transition locale. Pour cela, on va montrer les quatre cas d'enchaînement possibles pour les événements e_1 et e_2, les plus prioritaires (qui respectent leur degré d'idempotence), de l'ensemble $\{e \in \breve{\mathcal{E}}_{ccc_{[l^{(1)}..l^{(N)}]}} / O_e(ccc_{[l^{(1)}..l^{(N)}]}) = \eta_e\}$ et le raisonnement suit le même pour tous les suivants :

- Soit l'automate $\mathcal{A}^{(i)}$ est concerné par les deux événements. Par construction du tuple de transition globale, la composante d'indice i des états globaux $\tilde{x}_1$ et $\tilde{y}_1$ du tuple de transition globale de l'événement e_1 est égale à l'état $x_1^{(i)}$ et $y_1^{(i)}$ du tuple de transition locale de l'événement e_1 de la chaîne de transitions locales $l^{(i)}$ et les états globaux $\tilde{x}_2$ et $\tilde{y}_2$ du tuple de transition globale de l'événement e_2 est égale à l'état $x_2^{(i)}$ et $y_2^{(i)}$ du tuple de transition locale de l'événement e_2 de la chaîne de transitions locales $l^{(i)}$. En sachant que les tuples de transition locale sont enchaînés, on a $y_1^{(i)} = x_2^{(i)}$, et, toutes les composantes des états globaux de tuples de transition globale telles que $i \in \mathcal{O}_{e_1}$ et $i \in \mathcal{O}_{e_2}$ sont enchaînées ;
- Soit l'automate $\mathcal{A}^{(i)}$ est concerné par l'événement e_1 et n'est pas concerné par l'événement e_2. Par construction du tuple de transition globale, la composante d'indice i des états globaux $\tilde{x}_1$ et $\tilde{y}_1$ du tuple de transition globale de l'événement e_1 est égale à l'état $x_1^{(i)}$ et $y_1^{(i)}$ du tuple de transition locale de l'événement e_1 de la chaîne de transitions locales $l^{(i)}$ et en sachant que l'automate $\mathcal{A}^{(i)}$ n'est pas concerné par l'événement e_2, la composante d'indice i des états globaux $\tilde{x}_2$ et $\tilde{y}_2$ est égale à la composante d'indice i de l'état global $\tilde{y}_1$. Autrement dit, la composante d'indice i des états globaux du tuple de transition globale de l'événement e_2 ne change pas et est égale à l'état d'arrivée du tuple de transition globale précédent, donc elle est enchaînée.
- Soit l'automate $\mathcal{A}^{(i)}$ n'est pas concerné par l'événement e_1 mais il est concerné par l'événement e_2. Par construction du tuple de transition globale, la composante d'indice i des états globaux $\tilde{x}_1$ et $\tilde{y}_1$ du tuple de transition globale de l'événement e_1 est égale à l'état de départ $x^{(i)}$ considéré dans le lemme. En sachant que l'événement e_2 est le premier événement de la chaîne de transitions locales $l^{(i)}$, alors l'état de départ du tuple de transition locale de l'événement e_2 est forcement égal à l'état de départ $x^{(i)}$ considéré dans le lemme. Donc, on a la composante d'indice i de l'état global $\tilde{y}_1$ du tuple de transition globale de l'événement e_1 est égale à $x^{(i)}$ et la composante d'indice i de l'état global $\tilde{x}_2$ du tuple de transition globale de l'événement e_2 aussi égale à $x^{(i)}$, alors elles sont enchaînées.
- Soit l'automate $\mathcal{A}^{(i)}$ n'est pas concerné ni par l'événement e_1, ni par l'événement e_2, alors, la composante d'indice i des états globaux $\tilde{x}_1$ et $\tilde{y}_1$, ainsi que $\tilde{x}_2$ et $\tilde{y}_2$, sont égaux à la composante d'indice i de $\tilde{x}$. Alors elles sont enchaînées.

Ceci nous prouve l'enchaînement des tuples de transition globale de la chaîne de transition globale g. Alors, la règle 2 des règles de construction d'une chaîne de transitions globales est respectée.

Pour les règles 3, 4 et 5, nous allons raisonner par l'absurde. Supposons qu'on puisse enchaîner un tuple de transition globale $(e, \Pi_e(\tilde{x}_e, \tilde{y}_e))$ d'un événement $e \in pot(\tilde{x})$ au début de la chaîne de transitions globales $g \in \mathcal{L}(\tilde{x}, \tilde{y})$. Dans ce cas, on obtient une chaîne de transitions globales g' telle que :

$$g' = (e, \Pi_e(\tilde{x}_e, \tilde{y}_e)) \cdot g \tag{8.12}$$

Par définition d'un chaînon de transition globale (Définition 6.3.1), représenté ici par un tuple de transition globale, la probabilité de routage globale de ce chaînon est calculée à partir de la probabilité de routage de tuple de transition locale. On reprend le calcul de la probabilité de routage globale d'un chaînon ci-dessous :

$$\Pi_e(\tilde{x}_e, \tilde{y}_e) = \prod_{i \in \mathcal{O}_e, \exists (e, \pi_e(x_e^{(i)}, y_e^{(i)})) \in \mathcal{P}(x^{(i)}, y^{(i)})} \pi_e(x_e^{(i)}, y_e^{(i)})$$

Autrement dit, pour tout i dans $\mathcal{O}_e$, il existe un tuple de transition locale qui peut être enchaîné au début de la chaîne de transitions locales $l^{(i)}$. Alors, en enchaînant ce tuple de transition locale à la chaîne $l^{(i)}$, on obtient une chaîne de transitions locales $l'^{(i)}$ telle que :

$$l'^{(i)} = (e, \pi_e(x_e^{(i)}, x_e^{(i)})) \cdot l^{(i)} \tag{8.13}$$

La chaîne de transitions locales $l'^{(i)}$ est forcement plus grande que la chaîne de transitions locales $l^{(i)}$, vu qu'elle a un tuple de transition locale de plus, ce qui contredit l'hypothèse initiale, qui dit que les chaînes de transitions locales $l^{(i)}$ sont maximales.

On a supposé l'enchaînement d'un tuple de transition globale au début de la chaîne de transitions globales $g \in \mathcal{L}(\tilde{x}, \tilde{y})$, cette même procédure peut être appliquée pour un tuple de transition globale qui s'enchaîne au milieu ou à la fin de la chaîne.

Par l'absurde, on a montré qu'on ne peut pas enchaîner d'autre tuple de transition globale d'un événement $e \in pot(\tilde{x})$ à la chaîne de transitions globales $g \in \mathcal{L}(\tilde{x}, \tilde{y})$.

En conclusion, on a pu vérifier toute les règles de construction d'une chaîne de transitions globales (Définition 6.3.2, page 105) pour le produit de simultanéité des tuples de transition de l'équation (8.11) (page 144) $g = \overset{N}{\underset{i=1}{*}} l^{(i)}$ et donc $g \in \mathcal{L}(\tilde{x}, \tilde{y})$.

□

Le lemme suivant montre que, étant donné une chaîne de transitions globales à partir des états globaux $\tilde{x}$ et $\tilde{y}$, on peut factoriser cette chaîne de transitions globales en chaînes de transitions locales de façon à ce que le produit de concurrence de ces chaînes de transitions locales soit égal à cette chaîne de transitions globales.

Lemme 8.2.2. *Soient $\tilde{x} = (x^{(1)}, \ldots, x^{(N)})$ et $\tilde{y} = (y^{(1)}, \ldots, y^{(N)})$ des états globaux d'un* SAN *complété $(\breve{\mathcal{E}}, \breve{\mathcal{A}}^{(i)}, \mathcal{S}, \mathcal{F})$, où $i \in [1..N]$, si $g \neq O_{\breve{\mathcal{T}}}$ et $g \in \mathcal{L}(\tilde{x}, \tilde{y})$, alors $\exists! l^{(1)} \in \mathcal{L}^{(1)}(x^{(1)}, y^{(1)}), .., \exists! l^{(N)} \in \mathcal{L}^{(N)}(x^{(N)}, y^{(N)})$ tel que $g = \overset{N}{\underset{i=1}{*}} l^{(i)}$.*

Démonstration. Par la définition 8.1.1 (page 136), étant donné une chaîne de transitions globales $g = \{(\tilde{x}_1, \tilde{y}_1, e_1, \Pi_{e_1}(\tilde{x}_1, \tilde{y}_1)), \ldots, (\tilde{x}_{C_g}, \tilde{y}_{C_g}, e_{C_g}, \Pi_{e_{C_g}}(\tilde{x}_{C_g}, \tilde{y}_{C_g}))\}$ composée de C_g chaînons de transition globale de $\tilde{\mathcal{T}}_{pot(\tilde{x})}$ (Définition 6.3.2), et $g \in \mathcal{L}(\tilde{x}, \tilde{y})$, alors cette chaîne de transitions globales g peut être représentée par le produit de simultanéité des tuples de transition globale :

$$g = \overset{C_g}{\underset{j=1}{\cdot}} \Big(e_{j_g}, \Pi_{e_{j_g}}(\tilde{x}_{j_g}, \tilde{y}_{j_g}) \Big) \tag{8.14}$$

Ces tuples de transition globale sont dérivés des chaînons transition globale de la chaîne de transitions globales g. À partir de cette chaîne, on va construire de façon incrémentale les chaînes de transitions locales $l'^{(i)}$ en décomposant (opération inverse de celle faite dans le lemme 8.2.1) ces tuples de transition globale. *Attention, ces chaînes de transitions locales $l'^{(i)}$ ne sont pas les chaînes de transitions locales $l^{(i)}$ attendues en résultat.* Ces chaînes de transitions locales $l'^{(i)}$ seront complétées pour obtenir les chaînes de transitions locales $l^{(i)}$ souhaitées.

Par la propriété d'idempotence (Propriété 7.2.1, page 129), on peut décomposer (opération inverse de la factorisation) le *premier tuple de transition globale* (de l'événement le plus prioritaire) de l'équation (8.14) de façon à ce que ce tuple de transition soit réécrit par un produit de concurrence de la façon suivante :

$$\begin{aligned} \overset{C_g}{\underset{j=1}{\cdot}} \Big(e_{j_g}, \Pi_{e_{j_g}}(\tilde{x}_{j_g}, \tilde{y}_{j_g}) \Big) &= \left(\overset{N}{\underset{i=1}{*}} csc_{l'^{(i)}} \right) \cdot \left(\overset{C_g}{\underset{j=2}{\cdot}} \Big(e_{j_g}, \Pi_{e_{j_g}}(\tilde{x}_{j_g}, \tilde{y}_{j_g}) \Big) \right), \\ csc_{l'^{(i)}} &= \begin{cases} \big(e_{1_g}, \pi_{e_{1_g}}(x^{(i)}_{1_g}, y^{(i)}_{1_g}) \big) & \text{si } i \in \mathcal{O}_{e_{1_g}} \\ I_{\breve{\mathcal{T}}} & \text{si } i \notin \mathcal{O}_{e_{1_g}} \end{cases} \end{aligned} \tag{8.15}$$

En sachant que la probabilité de routage globale $\Pi_{e_{1_g}}(\tilde{x}_{1_g}, \tilde{y}_{1_g})$ du tuple de transition globale est la probabilité de routage du chaînon de transition globale (Définition 6.3.1, page 104) pour ce tuple et avec la connaissance du modèle SAN, on peut obtenir les probabilités de routage $\pi_{e_{1_g}}(x^{(i)}_{1_g}, y^{(i)}_{1_g})$ de chaque tuple de transition locale à partir de cette probabilité de routage globale. Cette probabilité de routage globale est calculée par :

$$\Pi_{e_{1_g}}(\tilde{x}_{1_g}, \tilde{y}_{1_g}) = \prod_{i \in \mathcal{O}_{e_{1_g}}, \exists (e_{1_g}, \pi_{e_{1_g}}(x^{(i)}_{1_g}, y^{(i)}_{1_g})) \in \mathcal{P}(x^{(i)}, y^{(i)})} \pi_{e_{1_g}}(x^{(i)}_{1_g}, y^{(i)}_{1_g}),$$

Par la propriété 3 (tuple de transition neutre) de la définition de l'opérateur de concurrence (Définition 7.2.9, page 128), les tuples éventuels de transition neutre disparaissent dans le produit de concurrence du membre droit de l'équation (8.15). Donc, on peut réécrire le produit de concurrence du membre droit de l'équation (8.15) par :

$$\underset{\substack{i\in[1..N]/\\ i\in\mathcal{O}_{e_{j_g}}}}{\overset{*}{}}$$

Si on considère le produit de simultanéité ($\underset{i\in\mathcal{O}_{e_{j_g}}}{\overset{\cdot}{}}$) de tuples de transition locale d'un ensemble vide ($i \notin \mathcal{O}_{e_{j_g}}$) égal au tuple de transition $I_{\breve{\mathcal{T}}}$, on peut réécrire l'équation (8.15) par :

$$\overset{C_g}{\underset{j=1}{\cdot}}\Big(e_{j_g}, \Pi_{e_{j_g}}(\tilde{x}_{j_g}, \tilde{y}_{j_g})\Big) = \left(\overset{N}{\underset{i=1}{*}} \overbrace{\left(\underset{i\in\mathcal{O}_{e_{1_g}}}{\overset{\cdot}{}} (e_{1_g}, \pi_{e_{1_g}}(x_{1_g}^{(i)}, y_{1_g}^{(i)}))\right)}^{csc_{l'(i)}}\right) \cdot \left(\overset{C_g}{\underset{j=2}{\cdot}}\Big(e_{j_g}, \Pi_{e_{j_g}}(\tilde{x}_{j_g}, \tilde{y}_{j_g})\Big)\right) \tag{8.16}$$

En décomposant le deuxième tuple de transition globale, on réécrit l'équation (8.16) par :

$$\overset{C_g}{\underset{j=1}{\cdot}}\Big(e_{j_g}, \Pi_{e_{j_g}}(\tilde{x}_{j_g}, \tilde{y}_{j_g})\Big) = \left(\overset{N}{\underset{i=1}{*}} \overbrace{\left(\underset{\substack{k\in[1..2]/\\ i\in\mathcal{O}_{e_{k_g}}}}{\overset{\cdot}{}} (e_{k_g}, \pi_{e_{k_g}}(x_{k_g}^{(i)}, y_{k_g}^{(i)}))\right)}^{csc_{l'(i)}}\right) \cdot \left(\overset{C_g}{\underset{j=3}{\cdot}}\Big(e_{j_g}, \Pi_{e_{j_g}}(\tilde{x}_{j_g}, \tilde{y}_{j_g})\Big)\right) \tag{8.17}$$

Alors, en décomposant les C_g tuples de transition de la chaîne de transitions globales g, on obtient :

$$g = \overset{C_g}{\underset{j=1}{\cdot}}\Big(e_{j_g}, \Pi_{e_{j_g}}(\tilde{x}_{j_g}, \tilde{y}_{j_g})\Big) = \overset{N}{\underset{i=1}{*}} \overbrace{\left(\underset{\substack{k\in[1..C_g]/\\ i\in\mathcal{O}_{e_{k_g}}}}{\overset{\cdot}{}} \Big(e_{k_g}, \pi_{e_{k_g}}(x_{k_g}^{(i)}, y_{k_g}^{(i)})\Big)\right)}^{csc_{l'(i)}} \tag{8.18}$$

Il faut maintenant assurer que tous les $csc_{l'(i)}$ respectent les règles de construction d'une chaîne de transitions locales (Définition 8.2.1, page 138) pour l'ensemble $\epsilon = pot(x^{(i)})$. Rappelons cette définition :

Soit un ensemble d'évènements ϵ et un automate complété $\breve{\mathcal{A}}^{(i)}$, on appelle **chaîne de transitions locales**, *une liste ordonnée $\{(e_1, \pi_{e_1}(x_1^{(i)}, y_1^{(i)})), \ldots, (e_C, \pi_{e_C}(x_C^{(i)}, y_C^{(i)}))\}$ composée de C tuples de transition locale de $\mathcal{T}_\epsilon^{(i)}$, qui respectent les règles suivantes :*

1. $\varrho_{e_1} < \varrho_{e_2} < \ldots < \varrho_{e_C}$

2. $\forall j \in [2..C]$
   $x_j^{(i)} = y_{j-1}^{(i)}$
3. $\nexists (e, \pi_e(x^{(i)}, y^{(i)})) \in \mathcal{T}_\epsilon^{(i)}$ *tel que*
   $\varrho_e < \varrho_{e_1}$ *et* $y^{(i)} = x_1^{(i)}$
4. $\nexists (e, \pi_e(x^{(i)}, y^{(i)})) \in \mathcal{T}_\epsilon^{(i)}$ *tel que*
   $\varrho_{e_C} < \varrho_e$ *et* $x^{(i)} = y_C^{(i)}$
5. $\forall j \in [1..C-1], \nexists (e, \pi_e(x^{(i)}, y^{(i)})) \in \mathcal{T}_\epsilon^{(i)}$ *tel que*
   $\varrho_{e_j} < \varrho_e < \varrho_{e_{j+1}}$ *et* $y_j^{(i)} = x^{(i)}$

On va montrer que les chaînes de transitions locales $l'^{(i)}$ sont bien des chaînes de transitions locales pour l'ensemble $\epsilon = pot(x^{(i)})$.

La règle 1 établit l'ordre d'enchaînement de tuples de transition locale. Selon l'ordre de décomposition des événements qu'on vient de faire (du plus prioritaire au moins prioritaire), les tuples de transition locale obtenus par cette décomposition respectent cette règle.

La règle 2 établit l'enchaînement de tuples de transition locale. Pour cela, on va montrer l'enchaînement de chaque composante de l'état de départ et d'arrivée des tuples de transition globale des événements e_1 et e_2, les plus prioritaires de la chaîne de transitions globales g. En sachant que les tuples de transition globale $(e_1, \Pi_{e_1}(\tilde{x}_1, \tilde{y}_1))$ et $(e_2, \Pi_{e_2}(\tilde{x}_2, \tilde{y}_2))$ sont enchaînés, on a que $\tilde{y}_1 = \tilde{x}_2$. Alors, on a quatre cas possibles d'enchaînement des tuples de transition locale des chaînes de transitions locales $l'^{(i)}$, où $i \in [1..N]$:

- Soit l'automate $\mathcal{A}^{(i)}$ est concerné par les deux événements. Par construction des tuples de transition locale, par l'ordre de priorité, l'état de départ $x_1^{(i)}$ et d'arrivée $y_1^{(i)}$ du tuple de transition locale de l'événement e_1 sont égaux à la composante d'indice i des état globaux $\tilde{x}_1$ et $\tilde{y}_1$ du tuple de transition globale de l'événement e_1 et l'état de départ $x_2^{(i)}$ et d'arrivée $y_2^{(i)}$ du tuple de transition locale de l'événement e_2 sont égaux à la composante d'indice i des état globaux $\tilde{x}_2$ et $\tilde{y}_2$ du tuple de transition globale de l'événement e_2. En sachant que $\tilde{y}_1 = \tilde{x}_2$, par conséquence $y_1^{(i)} = x_2^{(i)}$ et les tuples de transition locale sont enchaînés.
- Soit l'automate $\mathcal{A}^{(i)}$ est concerné par l'événement e_1 et n'est pas concerné par l'événement e_2. Par construction des tuples de transition locale, il n'existe pas de tuple de transition locale de l'événement e_2 dans la chaîne de transitions locales $l'^{(i)}$. Selon la méthode de décomposition proposée (Équation (8.16)) la décomposition de ces deux tuples de transition globale, la chaîne de transitions locales $l'^{(i)}$ est composée uniquement du tuple de transition locale de l'événement e_1 (donc il n'y as pas d'enchaînement à étudier).
- Soit l'automate $\mathcal{A}^{(i)}$ n'est pas concerné par l'événement e_1 mais il est concerné par l'événement e_2. Par construction des tuples de transition locale, il n'existe pas de tuple de transition locale de l'événement e_1 dans la chaîne de transitions locales $l'^{(i)}$. Après la décomposition de ces deux événements, la chaîne de transitions locales $l'^{(i)}$ est composée uniquement du tuple de transition locale de l'événement e_2.
- Soit l'automate $\mathcal{A}^{(i)}$ n'est pas concerné par l'événement e_1 ni par l'événement e_2. Par construction des tuples de transition locale, il n'existe pas de tuples de transition locale ni de l'événement e_1, ni de l'événement e_2 dans $l'^{(i)}$. Après la décomposition de ces deux

tuples de transition globale, la chaîne de transitions locales $l'^{(i)}$ continue à être vide. Cette chaîne de transition locale sera vide jusqu'à ce qu'on décompose un tuple de transition globale de g avec un événement e tel que $i \in \mathcal{O}_e$.

On vient de montrer que les deux premières règles de construction d'une chaîne de transitions locales sont vraies pour les chaînes de transitions locales $l'^{(i)}$, où $i \in [1..N]$. Cependant, pour les règles 3, 4 et 5, qui établissent la maximalité de la chaîne, il est possible que $l'^{(i)}$ ne soit pas une chaîne de transitions locales pour l'ensemble $\epsilon = pot(x^{(i)})$, car il se peut qu'il existe des tuples de transition locale d'événements de $pot(x^{(i)})$ qui sont enchaînable en $l'^{(i)}$. Donc, on va compléter ces chaînes $l'^{(i)}$ en $l^{(i)}$ de façon à ce que $\overset{N}{\underset{i=1}{*}} l'^{(i)} = \overset{N}{\underset{i=1}{*}} l^{(i)}$ et cette complétion sera unique car $g \neq O_{\check{\mathcal{T}}}$.

Considérons les événements e de $pot(x^{(i)})$ qui peuvent être enchaînés dans un $l'^{(i)}$:

– Soit e est un événement actif ($e \in \mathcal{E}$), alors si :
Cas (1*)* on rajoute un tuple de transition locale de l'événement e dans toutes les chaînes de transitions locales $l'^{(i)}$, où $i \in \mathcal{O}_e$; alors dans le produit de concurrence des chaînes de transitions locales $l'^{(i)}$ on obtient d'après le lemme 1 une chaîne de transitions globales g' qui a un événement de plus que la chaîne de transitions globales g. En sachant que g est maximale, il est impossible de rajouter un événement e actif dans toutes les chaînes de transitions locales $l'^{(i)}$, où $i \in \mathcal{O}_e$. *Le cas (*1*) est donc impossible*.

Cas (2*)* on rajoute un tuple de transition locale de l'événement e et e n'apparaît pas dans toutes les chaînes $l'^{(i)}$, où $i \in \mathcal{O}_e$, alors le produit de concurrence de ces chaînes est égal à $O_{\check{\mathcal{T}}}$, car l'événement e ne respecte pas leur degré d'idempotence (Propriété 7.2.1, item 3). En sachant qu'on considère uniquement les chaînes de transitions globales $g \neq O_{\check{\mathcal{T}}}$ alors, il est impossible de rajouter un événement e actif dans un sous-ensemble des chaînes de transitions locales $l'^{(i)}$, où $i \in \mathcal{O}_e$. *Le cas (*2*) est donc impossible*.

– Soit e est un événement complémentaire ($e \in \bar{\mathcal{E}}$), alors si :
Cas (3*)* on rajoute un tuple de transition locale de l'événement e dans toutes les chaînes de transitions locales $l'^{(i)}$, où $i \in \mathcal{O}_e$; alors dans le produit de concurrence des chaînes de transitions locales $l'^{(i)}$ on obtient d'après le lemme 1 une chaîne de transitions globales g' qui a un événement de plus que la chaîne de transitions globales g. En sachant que g est maximale, alors il est impossible de rajouter un événement e complémentaire dans toutes les chaînes de transitions locales $l'^{(i)}$, où $i \in \mathcal{O}_e$. *Le cas (*3*) est donc impossible*.

Cas (4*)* on rajoute un tuple de transition locale de l'événement e et e n'apparaît pas dans toutes les chaînes $l'^{(i)}$, où $i \in \mathcal{O}_e$, alors dans le produit de concurrence de ces chaînes, ces tuples de transitions locales vont disparaître (Propriété 7.2.1, item 2), car l'événement e ne respecte pas leur degré d'idempotence. *Le cas (*4*) est donc le seul cas possible*.

On a pu constater que uniquement les événements complémentaires qui ne respectent pas

leur degré d'idempotence sont enchaînables. On va maintenant compléter les chaînes de transitions locales $l'^{(i)}$, où $i \in [1..N]$ de façon à obtenir des chaînes de transitions locales $l^{(i)}$ maximales. Pour cela on va prendre uniquement les événements complémentaires de l'ensemble $pot(x^{(i)})$ et qui n'ont pas un tuple de transition globale dans la chaîne de transitions globales $g \in \mathcal{L}(\tilde{x}, \tilde{y})$. Autrement dit, les événements complémentaires qui ne respectent pas leur degré d'idempotence.

Prenons un événement complémentaire e quelconque de l'ensemble $pot(x^{(i)})$ et qui n'a pas de tuple de transition globale dans la chaîne de transitions globales $g \in \mathcal{L}(\tilde{x}, \tilde{y})$. Si on peut enchaîner un tuple de transition locale de cet événement e dans la chaîne de transitions locales $l'^{(i)}$, alors ce tuple de transition locale peut être enchaîné dans une seule position de la chaîne de transition locale $l'^{(i)}$ car la règle 1 de construction d'une chaîne de transitions locales établit un ordre d'enchaînement selon la priorité des événements.

On remarque que, dans un tuple de transition locale $(e, \pi_e(x_e^{(i)}, y_e^{(i)}))$ d'un événement complémentaire e l'état de départ $x_e^{(i)}$ et l'état d'arrivée $y_e^{(i)}$ sont égaux.

On va montrer les trois cas d'enchaînement possibles :

– Soit l'événement e est l'événement le plus prioritaire ($\varrho_e < \varrho_{e_1}$, où e_1 est l'événement le plus prioritaire d'un tuple de transition locale de la chaîne $l'^{(i)}$). Dans ce cas, s'il existe un tuple de transition locale $(e, \pi_e(x_e^{(i)}, y_e^{(i)}))$ tel que $y_e^{(i)}$ est égal à $x_1^{(i)}$, alors, en enchaînant $(e, \pi_e(x_e^{(i)}, y_e^{(i)}))$ à $l'^{(i)}$ on obtient une chaîne de transitions locales $l''^{(i)}$ telle que :

$$l''^{(i)} = (e, \pi_e(x_e^{(i)}, y_e^{(i)})) \cdot l'^{(i)} \tag{8.19}$$

– Soit l'événement e est l'événement le moins prioritaire ($\varrho_{e_C} < \varrho_e$, où e_C est l'événement le moins prioritaire d'un tuple de transition locale de la chaîne $l'^{(i)}$). Dans ce cas, s'il existe un tuple de transition locale $(e, \pi_e(x_e^{(i)}, y_e^{(i)}))$ tel que $x_e^{(i)}$ est égal à $y_C^{(i)}$, alors, en enchaînant $(e, \pi_e(x_e^{(i)}, y_e^{(i)}))$ à $l'^{(i)}$ on obtient une chaîne de transitions locales $l''^{(i)}$ telle que :

$$l''^{(i)} = l'^{(i)} \cdot (e, \pi_e(x_e^{(i)}, y_e^{(i)})) \tag{8.20}$$

– Soit l'événement e n'est pas l'événement le plus prioritaire, ni le moins prioritaire ($\varrho_{e_j} < \varrho_e < \varrho_{e_{j+1}}$, où e_j est l'événement d'indice j d'un tuple de transition locale de la chaîne $l'^{(i)}$). Dans ce cas, s'il existe un tuple de transition locale $(e, \pi_e(x_e^{(i)}, y_e^{(i)}))$ tel que $x_e^{(i)}$ est égal à $y_j^{(i)}$ et $y_e^{(i)}$ est égal à $x_{j+1}^{(i)}$, alors, en enchaînant $(e, \pi_e(x_e^{(i)}, y_e^{(i)}))$ à $l'^{(i)}$ on obtient une chaîne de transitions locales $l''^{(i)}$ telle que :

$$l''^{(i)} = \overset{j}{\underset{k=1}{\cdot}}(e_k, \pi_{e_k}(x_k^{(i)}, y_k^{(i)})) \cdot (e, \pi_e(x_e^{(i)}, y_e^{(i)})) \cdot \overset{C}{\underset{k=j+1}{\cdot}}(e_k, \pi_{e_k}(x_k^{(i)}, y_k^{(i)})) \tag{8.21}$$

où les tuples de transition locale $(e_k, \pi_{e_k}(x_k^{(i)}, y_k^{(i)}))$ sont les tuples de transition locale de la chaîne $l'^{(i)}$.

En sachant que la chaîne de transition locale $l'^{(i)}$ respecte les règles 1 et 2 de construction d'une chaîne de transitions locale et par l'enchaînement de l'événement complémentaire qu'on vient de faire, la chaîne de transition locale $l''^{(i)}$ respecte aussi ces deux règles.

En enchaînant un tuple de transition locale de cet événement $e \in pot(x^{(i)})$ à toutes les chaînes de transitions locales $l'^{(i)}$, où $i \in \mathcal{O}_e$, telles qu'il existe un tuple de transition locale enchaînable à $l'^{(i)}$. On obtient un jeu de chaînes de transitions locales $l''^{(i)}$, où $i \in [1..N]$ et on veut que :

$$\overset{N}{\underset{i=1}{*}} l''^{(i)} = \overset{N}{\underset{i=1}{*}} l'^{(i)} = g \tag{8.22}$$

Si $\overset{N}{\underset{i=1}{*}} l''^{(i)} = g' \neq g$ ceci voudrait dire que la chaîne de transitions globales g n'est pas maximale (de nouveau en contradiction à l'hypothèse initial) et on est donc forcement dans le cas (4) de la page 150. L'événement complémentaire e qu'on vient d'enchaîner à $l'^{(i)}$ ne respecte pas son degré d'idempotence et il va produire un tuple de transition neutre dans le produit de concurrence des chaînes $l''^{(i)}$.

En enchaînent, successivement, pour tout i et pour tous les événements complémentaires possibles des ensembles $pot(x^{(i)})$, tel qu'il n'existe pas un tuple de transition globale de cet événement dans $g \in \mathcal{L}(\tilde{x}, \tilde{y})$, on obtient la chaîne de transitions locales $l^{(i)}$ telle que :

$$\overset{N}{\underset{i=1}{*}} l^{(i)} = g \tag{8.23}$$

On va montrer maintenant que $l^{(i)}$ est une chaîne de transitions locales pour l'ensemble $\epsilon = pot(x^{(i)})$.

Par la complétion qu'on vient de faire et en sachant que les chaînes de transitions locales $l'^{(i)}$ respectent les règles 1 et 2, les chaînes de transitions locales $l^{(i)}$ respectent aussi ces règles.

Pour les règles 3, 4 et 5, nous allons raisonner par l'absurde. Supposons qu'on puisse enchaîner un tuple de transition locale $(e, \pi_e(x_e^{(i)}, y_e^{(i)}))$ d'un événement $e \in pot(x^{(i)})$ qui n'a pas un tuple de transition globale dans $g \in \mathcal{L}(\tilde{x}, \tilde{y})$ au début de la chaîne de transitions locales $l^{(i)} \in \mathcal{L}^{(i)}(x^{(i)}, y^{(i)})$. Dans ce cas, on obtient une chaîne de transitions locales $l^{+(i)}$ telle que :

$$l^{+(i)} = (e, \pi_e(x_e^{(i)}, y_e^{(i)})) \cdot l^{(i)} \tag{8.24}$$

Considérons les trois de quatre cas possibles, qu'on a présenté précédemment :

- Soit l'événement e est un événement actif et il apparaît dans tout les chaînes de transitions locales $l^{+(i)}$, où $i \in \mathcal{O}_e$. Alors dans le produit de concurrence des chaînes de transitions locales $l'^{(i)}$ on obtient, d'après le lemme 1, une chaîne de transitions globales g' qui a un événement de plus que la chaîne de transitions globales g, ce qui contredit l'hypothèse initiale, qui dit que la chaîne de transitions globale g est maximale.
- Soit l'événement e est un événement actif et il n'apparaît pas dans tout les chaînes de transitions locales $l^{+(i)}$, où $i \in \mathcal{O}_e$. En sachant qu'on a pour hypothèse que les chaînes

de transitions globales g sont différentes de $O_{\check{\mathcal{T}}}$ et que le produit de concurrence de ces chaînes de transitions locales est égal à $O_{\check{\mathcal{T}}}$, cela contredit cette hypothèse initiale.
- Soit l'événement e est un événement complémentaire et il apparaît dans tout les chaînes de transitions locales $l^{+(i)}$, où $i \in \mathcal{O}_e$. Alors dans le produit de concurrence des chaînes de transitions locales $l'^{(i)}$ on obtient d'après le lemme 1 une chaîne de transitions globales g' qui a un événement de plus que la chaîne de transitions globales g, ce qui contredit l'hypothèse initiale, qui dit que la chaîne de transitions globale g est maximale.
- Soit l'événement e est un événement complémentaire et il n'apparaît pas dans tout les chaînes de transitions locales $l^{+(i)}$, où $i \in \mathcal{O}_e$. Alors, en sachant que, par la complétion qu'on vient de faire, on a déjà enchaîné tous les tuples de transition locale possibles de $pot(x^{(i)})$, donc il n'existe pas d'autres événement e est un événement complémentaire qui ne respect pas leur degré d'idempotence dans $pot(x^{(i)})$ alors, ce tuple de transition locale ne peut pas être enchaîné.

Par l'absurde, on a montré qu'on ne peut pas enchaîner d'autre tuple de transition locale d'un événement $e \in pot(x^{(i)})$ aux chaînes de transitions locales $l^{(i)}$, où $i \in [1..N]$ et que $l^{(i)}$ respecte les règles 3, 4 et 5 de règles de construction d'une chaîne de transitions locales.

Ce même raisonnement peut être appliquée pour un tuple de transition locale qui s'enchaîne au milieu ou à la fin de la chaîne.

En conclusion, on a pu vérifier toute les règles de construction d'une chaîne de transitions locales (Définition 8.2.1, page 138) pour le produit de simultanéité des tuples de transition locale des chaînes de l'équation (8.18) (page 148) $\overset{N}{\underset{i=1}{*}} l'^{(i)} = \overset{N}{\underset{i=1}{*}} l^{(i)} = g$ et donc $l^{(i)} \in \mathcal{L}^{(i)}(x^{(i)}, y^{(i)})$, où $i \in [1..N]$.

Cette décomposition est unique car on ne recherche que des chaînes de transitions locales $l^{(i)}$ maximales contenant les événements de la chaîne de transitions globales g. □

Dans le premier lemme (Lemme 8.2.1) on a montré que le produit de concurrence d'un jeu de chaînes de transitions locales $[l^{(1)}, .., l^{(N)}]$ telles que $l^{(1)} \in \mathcal{L}^{(1)}(x^{(1)}, y^{(1)}), .., l^{(N)} \in \mathcal{L}^{(N)}(x^{(N)}, y^{(N)})$ est égal à une chaîne de transitions globale $g \in \mathcal{L}(\tilde{x}, \tilde{y})$. On peut donc affirmer que tout jeu de chaînes de transitions locales, tels que le produit de concurrence de ces chaînes de transitions locales est différent de $O_{\check{\mathcal{T}}}$, alors ce produit de concurrence est une chaîne de transitions globales $g \in \mathcal{L}(\tilde{x}, \tilde{y})$.

On note $\mathcal{L}^{(1)}(x^{(1)}, y^{(1)}) * .. * \mathcal{L}^{(N)}(x^{(N)}, y^{(N)})$ l'ensemble de produits de concurrence de tous les jeux de chaînes de transitions locales telles que le produit de concurrence de ces chaînes est différent de $O_{\check{\mathcal{T}}}$.

Dans le deuxième lemme (Lemme 8.2.2) on a montré qu'une chaîne de transitions globales $g \in \mathcal{L}(\tilde{x}, \tilde{y})$ peut être décomposée en un jeu de chaînes de transitions locales $l^{(i)} \in \mathcal{L}^{(i)}(x^{(i)}, y^{(i)})$, où $i \in [1..N]$. Alors, on peut affirmer que toutes les chaînes de transitions globales $g \in \mathcal{L}(\tilde{x}, \tilde{y})$ sont égales à un produit de concurrence unique de chaînes de transitions locales dans l'ensemble $l^{(i)} \in \mathcal{L}^{(1)}(x^{(1)}, y^{(1)}) * .. * \mathcal{L}^{(N)}(x^{(N)}, y^{(N)})$.

Ces constatations nous permet d'affirmer une bijection entre les ensembles $\mathcal{L}(\tilde{x}, \tilde{y})$ et $\mathcal{L}^{(1)}(x^{(1)}, y^{(1)}) * .. * \mathcal{L}^{(N)}(x^{(N)}, y^{(N)})$. Cette constatation est présentée dans le corollaire suivant.

Corollaire 8.2.1. *Soient $\mathcal{L}(\tilde{x}, \tilde{y})$ l'ensemble de chaînes de transitions globales qui amènent de l'état $\tilde{x}$ à l'état $\tilde{y}$ et $\mathcal{L}^{(1)}(x^{(1)}, y^{(1)}) * .. * \mathcal{L}^{(N)}(x^{(N)}, y^{(N)})$ l'ensemble des jeux de chaînes de transitions locales telles que le produit de concurrence de ces chaînes est différent de $O_{\breve{\mathcal{T}}}$ et qui amènent de l'état local $x^{(i)}$ à l'état local $y^{(i)}$, où $i \in [1..N]$, alors*

$$\mathcal{L}(\tilde{x}, \tilde{y}) = \mathcal{L}^{(1)}(x^{(1)}, y^{(1)}) * .. * \mathcal{L}^{(N)}(x^{(N)}, y^{(N)})$$

Dans la section suivante, on présente le *descripteur discret* d'un modèle SAN à temps discret utilisant les matrices d'événements définies précédemment.

8.3 Descripteur discret

Le *descripteur discret* est une façon compacte de représenter les transitions d'un modèle SAN à temps discret. Le descripteur discret d'un modèle SAN est représenté par une formule tensorielle complexe (*i.e.*, via l'algèbre tensorielle complexe défini dans le chapitre 7) utilisant les matrices d'événements de ce modèle.

En utilisant l'algèbre tensorielle complexe, les *événements* (Définition 6.2.3) d'un modèle SAN à temps discret sont des *éléments actifs* (Définition 7.1.1), et les *événements complémentaires* (Définition 6.2.9) du modèle sont des *événements complémentaires* (Définition 7.1.1) de l'algèbre tensorielle complexe. De plus, les *tuples de transition* (Définition 6.2.4) d'un modèle SAN sont de *couple d'éléments* (Définition 7.2.1) de l'algèbre tensorielle complexe.

On présente ici le descripteur discret d'un modèle SAN et la preuve d'égalité entre ce descripteur et la matrice de transition de l'automate global de ce modèle.

Théorème 8.3.1. *Étant donné un* SAN *complété $(\breve{\mathcal{E}}, \breve{\mathcal{A}}^{(i)}, \mathcal{S}, \mathcal{F})$, où $i \in [1..N]$, et pour chaque automate $\breve{\mathcal{A}}^{(i)}$ sa matrice locale d'événements $P^{(i)}$, alors le descripteur discret P est défini par la formule tensorielle complexe :*

$$P = \bigoplus_{i=1}^{N} P^{(i)} \tag{8.25}$$

Ce descripteur est égal à la matrice de transition $\mathcal{G}$ de l'automate global de ce modèle comme présenté dans l'équation (8.3) *(page 137).*

Démonstration. Les éléments de l'algèbre tensorielle complexe utilisés pour cette démonstration sont explicités dans le chapitre 7.

Selon la définition du produit tensoriel complexe (Définition 7.3.3), les éléments de la matrice complexe $P = \bigoplus_{i=1}^{N} P^{(i)}$ ont la forme :

$$\begin{array}{l} x^{(1)}, y^{(1)} \in [1..n_1], \ldots, x^{(N)}, y^{(N)} \in [1..n_N], \\ P_{[x^{(1)},\ldots,x^{(N)}],[y^{(1)},\ldots,y^{(N)}]} = \underset{i=1}{\overset{N}{*}} P^{(i)}(x^{(i)}, y^{(i)}) \end{array} \tag{8.26}$$

Les éléments de la matrice de transition $\mathcal{G}$ de l'automate global sont des chaînes de transitions globales. Par la réécriture de ces chaînes de transitions globales présentée dans l'équation (8.3), les éléments de la matrice de transition $\mathcal{G}$ ont la forme :

$$\boxed{\begin{array}{l} \forall \tilde{x} \in \mathcal{S}, \forall \tilde{y} \in \mathcal{S} \text{ et ce couple } (\tilde{x}, \tilde{y}) \text{ est fixé par la suite} \\ \qquad \mathcal{G}(\tilde{x}, \tilde{y}) = \underset{\forall g \in \mathcal{L}(\tilde{x},\tilde{y})}{+} \overset{C_g}{\underset{i=1}{\cdot}} \Big(e_{i_g}, \Pi_{e_{i_g}}(\tilde{x}_{i_g}, \tilde{y}_{i_g}) \Big) \end{array}} \tag{8.27}$$

Pour montrer l'égalité entre les éléments du descripteur discret P et de la matrice de transition $\mathcal{G}$ de l'automate global, on procédera en deux étapes :

1. On montre que tous les éléments différents de $O_{\breve{\mathcal{I}}}$ du descripteur P ont un élément égal et dans la même position dans la matrice de transition $\mathcal{G}$ de l'automate global ;
2. On montre tous les éléments différents de $O_{\breve{\mathcal{I}}}$ de la matrice de transition $\mathcal{G}$ de l'automate global ont un élément égal et dans la même position dans le descripteur P.

Première partie

L'élément $P^{(i)}(x^{(i)}, y^{(i)})$, présenté dans l'équation (8.26), est l'élément de la matrice locale d'événements $P^{(i)}$ de la ligne $x^{(i)}$ et la colonne $y^{(i)}$. Selon la définition des éléments de la matrice locale d'événements $P^{(i)}$ (Définition 8.2.3), chaque $P^{(i)}(x^{(i)}, y^{(i)})$ peut être réécrit, *i.e.* :

$$P^{(i)}(x^{(i)}, y^{(i)}) = \underset{\forall l^{(i)} \in \mathcal{L}^{(i)}(x^{(i)}, y^{(i)})}{+} \overset{C_{l^{(i)}}}{\underset{j=1}{\cdot}} \Big(e_{j_{l^{(i)}}}, \pi_{e_{j_{l^{(i)}}}}(x^{(i)}_{j_{l^{(i)}}}, y^{(i)}_{j_{l^{(i)}}}) \Big) \tag{8.28}$$

ce qui donne pour l'élément $P_{[x^{(1)},\ldots,x^{(N)}],[y^{(1)},\ldots,y^{(N)}]}$ de la matrice complexe du descripteur P :

$$P_{[x^{(1)},\ldots,x^{(N)}],[y^{(1)},\ldots,y^{(N)}]} = \underset{i=1}{\overset{N}{*}} \left(\underset{\forall l^{(i)} \in \mathcal{L}^{(i)}(x^{(i)}, y^{(i)})}{+} \overbrace{\overset{C_{l^{(i)}}}{\underset{j=1}{\cdot}} \Big(e_{j_{l^{(i)}}}, \pi_{e_{j_{l^{(i)}}}}(x^{(i)}_{j_{l^{(i)}}}, y^{(i)}_{j_{l^{(i)}}}) \Big)}^{csc_{l^{(i)}}} \right) \tag{8.29}$$

En utilisant la distributivité du produit de concurrence sur la somme de choix (Définition 7.2.9), on peut réécrire l'équation (8.29) tel que :

$$P_{[x^{(1)},...,x^{(N)}],[y^{(1)},...,y^{(N)}]} = \underset{\forall l^{(1)} \in \mathcal{L}^{(1)}(x^{(1)},y^{(1)})}{+} \cdots \underset{\forall l^{(N)} \in \mathcal{L}^{(N)}(x^{(N)},y^{(N)})}{+} \underbrace{\left(\overset{N}{\underset{i=1}{*}} \overbrace{\overset{C_{l^{(i)}}}{\underset{j=1}{\cdot}} \left(e_{j_{l^{(i)}}}, \pi_{e_{j_{l^{(i)}}}} \left(x^{(i)}_{j_{l^{(i)}}}, y^{(i)}_{j_{l^{(i)}}} \right) \right)}^{csc_{l^{(i)}}} \right)}_{ccc_{[l^{(1)}..l^{(N)}]}} \tag{8.30}$$

En appliquant le lemme 8.2.1, pour un $ccc_{[l^{(1)}..l^{(N)}]} \neq O_{\breve{\mathcal{I}}}$, il existe une chaîne de transitions globales $g_{[l^{(1)}..l^{(N)}]} \in \mathcal{L}(\tilde{x}, \tilde{y})$, telle que : ($ccc_{[l^{(1)}..l^{(N)}]}$ et $g_{[l^{(1)}..l^{(N)}]}$ sont notés en gras dans l'équation suivante)

$$\begin{aligned} \boldsymbol{ccc}_{[\boldsymbol{l^{(1)}..l^{(N)}}]} &= \left(\overset{N}{\underset{i=1}{*}} \overbrace{\overset{C_{l^{(i)}}}{\underset{j=1}{\cdot}} \left(e_{j_{l^{(i)}}}, \pi_{e_{j_{l^{(i)}}}} \left(x^{(i)}_{j_{l^{(i)}}}, y^{(i)}_{j_{l^{(i)}}} \right) \right)}^{csc_{l^{(i)}}} \right) \\ &= \overset{C_{\boldsymbol{g}_{[\boldsymbol{l^{(1)}..l^{(N)}}]}}}{\underset{j=1}{\cdot}} \left(e_{j_{\boldsymbol{g}_{[\boldsymbol{l^{(1)}..l^{(N)}}]}}}, \Pi_{e_{j_{\boldsymbol{g}_{[\boldsymbol{l^{(1)}..l^{(N)}}]}}}} (\tilde{x}_{j_{\boldsymbol{g}_{[\boldsymbol{l^{(1)}..l^{(N)}}]}}}, \tilde{y}_{j_{\boldsymbol{g}_{[\boldsymbol{l^{(1)}..l^{(N)}}]}}}) \right) \end{aligned} \tag{8.31}$$

Et d'après le corollaire 8.2.1, tout produit de concurrence d'un jeu de chaînes de transitions locales différents de $O_{\breve{\mathcal{I}}}$ de l'ensemble $\mathcal{L}^{(1)}(x^{(1)}, y^{(1)}) * .. * \mathcal{L}^{(N)}(x^{(N)}, y^{(N)})$ a une chaîne de transitions globales $g \in \mathcal{L}(\tilde{x}, \tilde{y})$. En sachant que $ccc_{[l^{(1)}..l^{(N)}]}$ est une chaîne de transitions globales dans $\mathcal{L}(\tilde{x}, \tilde{y})$, on va noter un $ccc_{[l^{(1)}..l^{(N)}]}$ simplement par g et on peut réécrire l'équation 8.30 par :

$\forall \tilde{x} \in \mathcal{S}, \forall \tilde{y} \in \mathcal{S}$,

$$\begin{aligned} P_{[x^{(1)},...,x^{(N)}],[y^{(1)},...,y^{(N)}]} &= \underset{\forall l^{(1)} \in \mathcal{L}^{(1)}(x^{(1)},y^{(1)})}{+} \cdots \underset{\forall l^{(N)} \in \mathcal{L}^{(N)}(x^{(N)},y^{(N)})}{+} ccc_{[l^{(1)}..l^{(N)}]} \\ &= \underset{\forall l^{(1)} \in \mathcal{L}^{(1)}(x^{(1)},y^{(1)})}{+} \cdots \underset{\forall l^{(N)} \in \mathcal{L}^{(N)}(x^{(N)},y^{(N)})}{+} g \\ &= \underset{\forall g \in \mathcal{L}(\tilde{x},\tilde{y})}{+} \overset{C_g}{\underset{j=1}{\cdot}} \left(e_{j_g}, \Pi_{e_{j_g}} (\tilde{x}_{j_g}, \tilde{y}_{j_g}) \right) \end{aligned}$$

Cette dernière expression est celle de $\mathcal{G}(\tilde{x}, \tilde{y})$ donnée dans l'équation 8.27.

On a montré dans cette première partie que tous les éléments du descripteur P différents de $O_{\breve{\mathcal{I}}}$ ont un élément égal et dans la même position dans la matrice de transition $\mathcal{G}$ de l'automate global.

Deuxième partie

On rappelle que les éléments de la matrice de transition $\mathcal{G}$ de l'automate global qui sont différents de $O_{\breve{\mathcal{I}}}$ ont la forme :

$\forall \tilde{x} \in \mathcal{S}, \forall \tilde{y} \in \mathcal{S}$ et ce couple $(\tilde{x}, \tilde{y})$ est fixé par la suite

$$\mathcal{G}(\tilde{x}, \tilde{y}) = \underset{\forall g \in \mathcal{L}(\tilde{x},\tilde{y})}{+} \quad \overset{C_g}{\underset{j=1}{\cdot}} \left(e_{j_g}, \Pi_{e_{j_g}}(\tilde{x}_{j_g}, \tilde{y}_{j_g}) \right) \tag{8.32}$$

En appliquant le lemme 8.2.2, pour une chaîne de transitions globales $g \in \mathcal{L}(\tilde{x}, \tilde{y})$, on a un jeu unique $[l_g^{(1)}..l_g^{(N)}] \in \mathcal{L}^{(1)}(x^{(1)}, y^{(1)}) * .. * \mathcal{L}^{(N)}(x^{(N)}, y^{(N)})$ de chaînes de transitions locales, telles que $\overset{N}{\underset{i=1}{*}} l_g^{(i)} = g$

$$\overset{C_g}{\underset{j=1}{\cdot}} \left(e_{j_g}, \Pi_{e_{j_g}}(\tilde{x}_{j_g}, \tilde{y}_{j_g}) \right) = \overset{N}{\underset{i=1}{*}} l_g^{(i)} = \overset{N}{\underset{i=1}{*}} \overbrace{\overset{C_{l^{(i)}}}{\underset{j=1}{\cdot}} \left(e_{j_{l^{(i)}}}, \pi_{e_{j_{l^{(i)}}}} \left(x^{(i)}_{j_{l^{(i)}}}, y^{(i)}_{j_{l^{(i)}}} \right) \right)}^{csc_{l_g^{(i)}}}$$

D'après le corollaire 8.2.1,

$$\begin{aligned} \mathcal{G}(\tilde{x}, \tilde{y}) &= \underset{\forall g \in \mathcal{L}(\tilde{x},\tilde{y})}{+} \quad \overset{C_g}{\underset{j=1}{\cdot}} \left(e_{j_g}, \Pi_{e_{j_g}}(\tilde{x}_{j_g}, \tilde{y}_{j_g}) \right) \\ &= \underset{\forall l^{(1)} \in \mathcal{L}^{(1)}(x^{(1)},y^{(1)})}{+} \cdots \underset{\forall l^{(N)} \in \mathcal{L}^{(N)}(x^{(N)},y^{(N)})}{+} \left(\overset{N}{\underset{i=1}{*}} \overbrace{\overset{C_{l^{(i)}}}{\underset{j=1}{\cdot}} \left(e_{j_{l^{(i)}}}, \pi_{e_{j_{l^{(i)}}}} \left(x^{(i)}_{j_{l^{(i)}}}, y^{(i)}_{j_{l^{(i)}}} \right) \right)}^{csc_{l_g^{(i)}}} \right) \end{aligned}$$

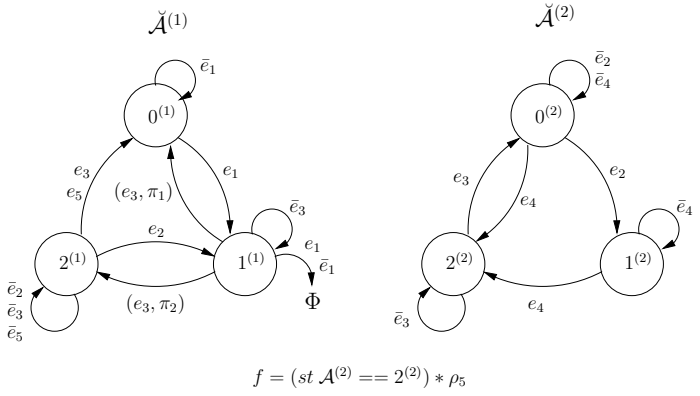
Cette dernière expression est celle de $P_{[x^{(1)},...,x^{(N)}],[y^{(1)},...,y^{(N)}]}$ donnée dans l'équation 8.30.

Dans cette deuxième partie, on a montré que chaque l'élément différent de $O_{\breve{\mathcal{I}}}$ de la matrice de transition $\mathcal{G}$ de l'automate globale a un élément égal et dans la même position dans le descripteur P.

Dans ces deux parties, on a montré que tous les éléments différents de $O_{\breve{\mathcal{I}}}$ du descripteur P ont un élément égal et dans la même position dans la matrice de transition $\mathcal{G}$ de l'automate global et que tous les éléments différents de $O_{\breve{\mathcal{I}}}$ de la matrice de transition $\mathcal{G}$ de l'automate global ont un élément égal et dans la même position dans le descripteur P. Donc, les deux matrices ont les éléments nuls dans les mêmes positions et des éléments égaux dans les autres positions. □

8.4 Exemple d'obtention du descripteur discret

Dans cette section, on va présenter, pas à pas, un exemple d'obtention du descripteur discret. Pour illustrer la procédure d'obtention du descripteur, on va reprendre l'exemple d'un réseau d'automate complété (FIG. 6.12) présenté au chapitre 6. On rappelle ce modèle dans FIG. 8.2.



Événement	Prob.	Priorité
e_1	ρ_1	3
e_2	ρ_2	1
e_3	ρ_3	2
e_4	ρ_4	4
e_5	f	5

Événement	Prob.	Priorité
$\bar{e}_1$	$1-\rho_1$	3
$\bar{e}_2$	$1-\rho_2$	1
$\bar{e}_3$	$1-\rho_3$	2
$\bar{e}_4$	$1-\rho_4$	4
$\bar{e}_5$	$1-f$	5

FIG. 8.2 – Exemple d'un réseau d'automates complétés

On remarque que l'automate $\breve{\mathcal{A}}^{(1)}$ est le même que celui de FIG. 8.1, donc on va présenter ici uniquement l'obtention de la matrice d'événements de l'automate $\breve{\mathcal{A}}^{(2)}$, car l'obtention de la matrice d'événements $P^{(1)}$ de l'automate $\breve{\mathcal{A}}^{(1)}$ a été déjà présentée dans la section 8.2.

On va commencer par définir l'ensemble d'événements possibles ($pot(x^{(i)})$) pour chaque état de l'automate $\breve{\mathcal{A}}^{(2)}$. À partir du $pot(x^{(i)})$, on peut obtenir pour cet automate les chaînes de transitions locales de cet automate. Ces chaînes de transitions sont listées dans TAB. 8.2.

On rappelle que les éléments d'une matrice d'événements sont des produits de simultanéité des tuples de transition des chaînes de transitions locales et sont définis selon la définition 8.2.3.

Pour l'automate complété $\breve{\mathcal{A}}^{(2)}$ de FIG. 8.2, à partir des chaînes de transitions locales présentées dans TAB. 8.2, on obtient la matrice d'événements $P^{(2)}$ suivante :

$$P^{(2)} = \left(\begin{array}{c|c|c} (\bar{e}_2,\pi_{\bar{e}_2}(0^{(2)},0^{(2)}))\cdot(\bar{e}_4,\pi_{\bar{e}_4}(0^{(2)},0^{(2)})) & (e_2,\pi_{e_2}(0^{(2)},1^{(2)}))\cdot(\bar{e}_4,\pi_{\bar{e}_4}(1^{(2)},1^{(2)})) & \begin{array}{c}(e_2,\pi_{e_2}(0^{(2)},1^{(2)}))\cdot(e_4,\pi_{e_4}(1^{(2)},2^{(2)}))\\ +\\ (\bar{e}_2,\pi_{\bar{e}_2}(0^{(2)},0^{(2)}))\cdot(e_4,\pi_{e_4}(0^{(2)},2^{(2)}))\end{array} \\ \hline O_{\breve{\mathcal{T}}} & (\bar{e}_4,\pi_{\bar{e}_4}(1^{(2)},1^{(2)})) & (e_4,\pi_{e_4}(1^{(2)},2^{(2)})) \\ \hline (e_3,\pi_{e_3}(2^{(2)},0^{(2)})) & O_{\breve{\mathcal{T}}} & (\bar{e}_3,\pi_{\bar{e}_3}(2^{(2)},2^{(2)})) \end{array}\right)$$

Le descripteur P est défini par le produit tensoriel complexe des matrices d'événements du modèle. Pour le réseau d'automates complété de FIG. 8.2, on a le descripteur suivant :

État de départ $0^{(2)}$; $pot(0^{(2)}) = \{e_2; \bar{e}_2; e_4; \bar{e}_4; \}$	
État d'arrivée	Chaînes de transitions locales
$0^{(2)}$	$\{(\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})); (\bar{e}_4, \pi_{\bar{e}_4}(0^{(2)}, 0^{(2)}))\}$
$1^{(2)}$	$\{(e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})); (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))\}$
$2^{(2)}$	$\{(e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})); (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))\}$
$2^{(2)}$	$\{(\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})); (e_4, \pi_{e_4}(0^{(2)}, 2^{(2)}))\}$

État de départ $1^{(2)}$; $pot(1^{(2)}) = \{e_4; \bar{e}_4\}$	
État d'arrivée	Chaînes de transitions locales
$1^{(2)}$	$\{(\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))\}$
$2^{(2)}$	$\{(e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))\}$

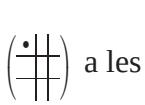
État de départ $2^{(2)}$; $pot(2^{(2)}) = \{e_3; \bar{e}_3\}$	
État d'arrivée	Chaînes de transitions locales
$0^{(2)}$	$\{(e_3, \pi_{e_3}(2^{(2)}, 0^{(2)}))\}$
$2^{(2)}$	$\{(\bar{e}_3, \pi_{\bar{e}_3}(2^{(2)}, 2^{(2)}))\}$

TAB. 8.2 – Les chaînes de transitions locales de l'automate complété $\breve{\mathcal{A}}^{(2)}$ de FIG. 8.2

$$P = P^{(1)} \circledast P^{(2)}$$

Étant donné la taille de la matrice P du descripteur, on va présenter les pas intermédiaires de la construction de la matrice par bloc. En appliquant le produit tensoriel complexe sur les matrices d'événements $P^{(1)}$ et $P^{(2)}$, les blocs on la forme suivante :

$$P = \left(\begin{array}{c|c|c} p^{(1)}_{0^{(1)},0^{(1)}} * P^{(2)} & p^{(1)}_{0^{(1)},1^{(1)}} * P^{(2)} & p^{(1)}_{0^{(1)},2^{(1)}} * P^{(2)} \\ \hline p^{(1)}_{1^{(1)},0^{(1)}} * P^{(2)} & p^{(1)}_{1^{(1)},1^{(1)}} * P^{(2)} & p^{(1)}_{1^{(1)},2^{(1)}} * P^{(2)} \\ \hline p^{(1)}_{2^{(1)},0^{(1)}} * P^{(2)} & p^{(1)}_{2^{(1)},1^{(1)}} * P^{(2)} & p^{(1)}_{2^{(1)},2^{(1)}} * P^{(2)} \end{array} \right)$$

Le bloc $p^{(1)}_{0^{(1)},0^{(1)}} * P^{(2)} = (\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * P^{(2)}$ dans la position  a les éléments suivants :

$(\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(0^{(2)}, 0^{(2)}))$	$(\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$	$(\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$ $+$ $(\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (e_4, \pi_{e_4}(0^{(2)}, 2^{(2)}))$
$(\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * O_{\check{\mathcal{I}}}$	$(\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$	$(\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$
$(\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * (e_3, \pi_{e_3}(2^{(2)}, 0^{(2)}))$	$(\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * O_{\check{\mathcal{I}}}$	$(\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * (\bar{e}_3, \pi_{\bar{e}_3}(2^{(2)}, 2^{(2)}))$

Par la propriété d'idempotence (Propriété 7.2.1), les éléments de ce bloc de la matrice sont réduits à :

$(\bar{e}_1, \Pi_{\bar{e}_1}(0^{(1)}0^{(2)}, 0^{(1)}0^{(2)})) \cdot (\bar{e}_4, \Pi_{\bar{e}_4}(0^{(1)}0^{(2)}, 0^{(1)}0^{(2)}))$	$O_{\check{\mathcal{I}}}$	$(\bar{e}_1, \Pi_{\bar{e}_1}(0^{(1)}0^{(2)}, 0^{(1)}0^{(2)})) \cdot (e_4, \Pi_{e_4}(0^{(1)}0^{(2)}, 0^{(1)}2^{(2)}))$
$O_{\check{\mathcal{I}}}$	$(\bar{e}_1, \Pi_{\bar{e}_1}(0^{(1)}1^{(2)}, 0^{(1)}1^{(2)})) \cdot (\bar{e}_4, \Pi_{\bar{e}_4}(0^{(1)}1^{(2)}, 0^{(1)}1^{(2)}))$	$(\bar{e}_1, \Pi_{\bar{e}_1}(0^{(1)}1^{(2)}, 0^{(1)}1^{(2)})) \cdot (e_4, \Pi_{e_4}(0^{(1)}1^{(2)}, 0^{(1)}2^{(2)}))$
$O_{\check{\mathcal{I}}}$	$O_{\check{\mathcal{I}}}$	$(\bar{e}_1, \Pi_{\bar{e}_1}(0^{(1)}2^{(2)}, 0^{(1)}2^{(2)}))$

Le bloc $p^{(1)}_{0^{(1)},1^{(1)}} * P^{(2)} = (e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * P^{(2)}$ dans la position $\left(\begin{smallmatrix} & \bullet & \\ \# & \# & \# \\ & \# & \end{smallmatrix}\right)$ a les éléments suivants :

$(e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(0^{(2)}, 0^{(2)}))$	$(e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$	$(e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$ $+$ $(e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (e_4, \pi_{e_4}(0^{(2)}, 2^{(2)}))$
$(e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * O_{\check{\mathcal{T}}}$	$(e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$	$(e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$
$(e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * (e_3, \pi_{e_3}(2^{(2)}, 0^{(2)}))$	$(e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * O_{\check{\mathcal{T}}}$	$(e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * (\bar{e}_3, \pi_{\bar{e}_3}(2^{(2)}, 2^{(2)}))$

Par la propriété d'idempotence (Propriété 7.2.1), les éléments de ce bloc de la matrice sont réduits à :

$(e_1, \Pi_{e_1}(0^{(1)}0^{(2)}, 1^{(1)}0^{(2)})) \cdot (\bar{e}_4, \Pi_{\bar{e}_4}(1^{(1)}0^{(2)}, 1^{(1)}0^{(2)}))$	$O_{\check{\mathcal{T}}}$	$(e_1, \Pi_{e_1}(0^{(1)}0^{(2)}, 1^{(1)}0^{(2)})) \cdot (e_4, \Pi_{e_4}(1^{(1)}0^{(2)}, 1^{(1)}2^{(2)}))$
$O_{\check{\mathcal{T}}}$	$(e_1, \Pi_{e_1}(0^{(1)}1^{(2)}, 1^{(1)}1^{(2)})) \cdot (\bar{e}_4, \Pi_{\bar{e}_4}(1^{(1)}1^{(2)}, 1^{(1)}1^{(2)}))$	$(e_1, \Pi_{e_1}(0^{(1)}1^{(2)}, 1^{(1)}1^{(2)})) \cdot (e_4, \Pi_{e_4}(1^{(1)}1^{(2)}, 1^{(1)}2^{(2)}))$
$O_{\check{\mathcal{T}}}$	$O_{\check{\mathcal{T}}}$	$(e_1, \Pi_{e_1}(0^{(1)}2^{(2)}, 1^{(1)}2^{(2)}))$

Le bloc $p^{(1)}_{0^{(1)},2^{(1)}} * P^{(2)} = O_{\breve{\mathcal{T}}} * P^{(2)}$ dans la position $\left(\begin{smallmatrix}\cdot & \cdot & \bullet\\ \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot\end{smallmatrix}\right)$ a les éléments suivants :

$O_{\breve{\mathcal{T}}} * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(0^{(2)}, 0^{(2)}))$	$O_{\breve{\mathcal{T}}} * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$	$O_{\breve{\mathcal{T}}} * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$ $+$ $O_{\breve{\mathcal{T}}} * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (e_4, \pi_{e_4}(0^{(2)}, 2^{(2)}))$
$O_{\breve{\mathcal{T}}} * O_{\breve{\mathcal{T}}}$	$O_{\breve{\mathcal{T}}} * (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$	$O_{\breve{\mathcal{T}}} * (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$
$O_{\breve{\mathcal{T}}} * (e_3, \pi_{e_3}(2^{(2)}, 0^{(2)}))$	$O_{\breve{\mathcal{T}}} * O_{\breve{\mathcal{T}}}$	$O_{\breve{\mathcal{T}}} * (\bar{e}_3, \pi_{\bar{e}_3}(2^{(2)}, 2^{(2)}))$

Par la propriété (2) de la définition 7.2.9, les éléments de ce bloc de la matrice sont réduits à :

$O_{\breve{\mathcal{T}}}$	$O_{\breve{\mathcal{T}}}$	$O_{\breve{\mathcal{T}}}$
$O_{\breve{\mathcal{T}}}$	$O_{\breve{\mathcal{T}}}$	$O_{\breve{\mathcal{T}}}$
$O_{\breve{\mathcal{T}}}$	$O_{\breve{\mathcal{T}}}$	$O_{\breve{\mathcal{T}}}$

Prenons maintenant les blocs de la deuxième ligne. Le bloc $p^{(1)}_{1^{(1)},0^{(1)}} * P^{(2)} = (e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * P^{(2)}$ dans la position $\left(\begin{smallmatrix} \cdot & \\ \hline & \end{smallmatrix}\right)$ a les éléments suivants :

$(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(0^{(2)}, 0^{(2)}))$	$(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$	$(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$ + $(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (e_4, \pi_{e_4}(0^{(2)}, 2^{(2)}))$
$(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * O_{\check{\mathcal{T}}}$	$(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$	$(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$
$(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * (e_3, \pi_{e_3}(2^{(2)}, 0^{(2)}))$	$(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * O_{\check{\mathcal{T}}}$	$(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (\bar{e}_1, \pi_{\bar{e}_1}(0^{(1)}, 0^{(1)})) * (\bar{e}_3, \pi_{\bar{e}_3}(2^{(2)}, 2^{(2)}))$

Par la propriété d'idempotence (Propriété 7.2.1), les éléments de ce bloc de la matrice sont réduits à :

$O_{\check{\mathcal{T}}}$	$O_{\check{\mathcal{T}}}$	$O_{\check{\mathcal{T}}}$
$O_{\check{\mathcal{T}}}$	$O_{\check{\mathcal{T}}}$	$O_{\check{\mathcal{T}}}$
$(e_3, \Pi_{e_3}(1^{(1)}2^{(2)}, 0^{(1)}0^{(2)})) \cdot (\bar{e}_1, \Pi_{\bar{e}_1}(0^{(1)}0^{(2)}, 0^{(1)}0^{(2)}))$	$O_{\check{\mathcal{T}}}$	$O_{\check{\mathcal{T}}}$

Le bloc $p^{(1)}_{1^{(1)},1^{(1)}} * P^{(2)} = \big((e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) + (\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)}))\big) * P^{(2)} = (e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * P^{(2)} + (\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * P^{(2)}$ dans la position $\left(\begin{smallmatrix} + & + & + \\ + & \bullet & + \\ + & + & + \end{smallmatrix}\right)$ a les éléments suivants :

$(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(0^{(2)}, 0^{(2)}))$ $+$ $(\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(0^{(2)}, 0^{(2)}))$	$(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$ $+$ $(\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$	$(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$ $+$ $(\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$ $+$ $(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (e_4, \pi_{e_4}(0^{(2)}, 2^{(2)}))$ $+$ $(\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (e_4, \pi_{e_4}(0^{(2)}, 2^{(2)}))$
$(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * O_{\bar{\mathcal{I}}}$ $+$ $(\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * O_{\bar{\mathcal{I}}}$	$(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$ $+$ $(\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$	$(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$ $+$ $(\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$
$(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * (e_3, \pi_{e_3}(2^{(2)}, 0^{(2)}))$ $+$ $(\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * (e_3, \pi_{e_3}(2^{(2)}, 0^{(2)}))$	$(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * O_{\bar{\mathcal{I}}}$ $+$ $(\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * O_{\bar{\mathcal{I}}}$	$(e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) \cdot (e_1, \pi_{e_1}(0^{(1)}, 1^{(1)})) * (\bar{e}_3, \pi_{\bar{e}_3}(2^{(2)}, 2^{(2)}))$ $+$ $(\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * (\bar{e}_3, \pi_{\bar{e}_3}(2^{(2)}, 2^{(2)}))$

Par la propriété d'idempotence (Propriété 7.2.1), les éléments de ce bloc de la matrice sont réduits à :

$(\bar{e}_4, \Pi_{\bar{e}_4}(1^{(1)}0^{(2)}, 1^{(1)}0^{(2)}))$	$O_{\bar{\mathcal{I}}}$	$(e_4, \Pi_{e_4}(1^{(1)}0^{(2)}, 1^{(1)}2^{(2)}))$
$O_{\bar{\mathcal{I}}}$	$(\bar{e}_4, \Pi_{\bar{e}_4}(1^{(1)}1^{(2)}, 1^{(1)}1^{(2)})$	$(e_4, \Pi_{e_4}(1^{(1)}1^{(2)}, 1^{(1)}2^{(2)}))$
$(e_3, \Pi_{e_3}(1^{(1)}2^{(2)}, 0^{(1)}0^{(2)})) \cdot (e_1, \Pi_{e_1}(0^{(1)}0^{(2)}, 1^{(1)}0^{(2)}))$	$O_{\bar{\mathcal{I}}}$	$(\bar{e}_3, \Pi_{\bar{e}_3}(1^{(1)}2^{(2)}, 1^{(1)}2^{(2)}))$

Le bloc $p^{(1)}_{1^{(1)},2^{(1)}} * P^{(2)} = (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) * P^{(2)}$ dans la position $\left(\begin{smallmatrix} + & + & \\ + & + & \bullet \\ & & \end{smallmatrix}\right)$ a les éléments suivants :

$(e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(0^{(2)}, 0^{(2)}))$	$(e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$	$(e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$ $+$ $(e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (e_4, \pi_{e_4}(0^{(2)}, 2^{(2)}))$
$(e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) * O_{\check{\mathcal{I}}}$	$(e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) * (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$	$(e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) * (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$
$(e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) * (e_3, \pi_{e_3}(2^{(2)}, 0^{(2)}))$	$(e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) * O_{\check{\mathcal{I}}}$	$(e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) * (\bar{e}_3, \pi_{\bar{e}_3}(2^{(2)}, 2^{(2)}))$

Par la propriété d'idempotence (Propriété 7.2.1), les éléments de ce bloc de la matrice sont réduits à :

$O_{\check{\mathcal{I}}}$	$O_{\check{\mathcal{I}}}$	$O_{\check{\mathcal{I}}}$
$O_{\check{\mathcal{I}}}$	$O_{\check{\mathcal{I}}}$	$O_{\check{\mathcal{I}}}$
$(e_3, \Pi_{e_3}(1^{(1)}2^{(2)}, 2^{(1)}0^{(2)}))$	$O_{\check{\mathcal{I}}}$	$O_{\check{\mathcal{I}}}$

Prenons maintenant les blocs qui composent la troisième ligne. Le bloc $p^{(1)}_{2^{(1)},0^{(1)}} * P^{(2)} = \big((e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) + (e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) + (\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (e_3, \pi_{e_3}(2^{(1)}, 0^{(1)})) + (\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)}))\big) * P^{(2)}$.

Par la propriété de distributivité de l'opérateur de concurrence sur l'opérateur de choix, on a $(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) * P^{(2)} + (e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * P^{(2)} + (\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (e_3, \pi_{e_3}(2^{(1)}, 0^{(1)})) * P^{(2)} + (\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * P^{(2)}$ dans la position $\left(\begin{smallmatrix}\cdot & \cdot & \cdot \\ \bullet & \cdot & \cdot \\ \cdot & \cdot & \cdot\end{smallmatrix}\right)$. Les les éléments de ce bloc sont les suivants :

$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(0^{(2)}, 0^{(2)}))$ + $(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(0^{(2)}, 0^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (e_3, \pi_{e_3}(2^{(1)}, 0^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(0^{(2)}, 0^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(0^{(2)}, 0^{(2)}))$	$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$ + $(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (e_3, \pi_{e_3}(2^{(1)}, 0^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$	$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$ + $(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (e_3, \pi_{e_3}(2^{(1)}, 0^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$ + $(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (e_4, \pi_{e_4}(0^{(2)}, 2^{(2)}))$ + $(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (e_4, \pi_{e_4}(0^{(2)}, 2^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (e_3, \pi_{e_3}(2^{(1)}, 0^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (e_4, \pi_{e_4}(0^{(2)}, 2^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (e_4, \pi_{e_4}(0^{(2)}, 2^{(2)}))$
$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) * O_{\bar{T}}$ + $(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * O_{\bar{T}}$ + $((\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (e_3, \pi_{e_3}(2^{(1)}, 0^{(1)})) * O_{\bar{T}}$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * O_{\bar{T}}$	$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) * (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$ + $(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (e_3, \pi_{e_3}(2^{(1)}, 0^{(1)})) * (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$	$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) * (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$ + $(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (e_3, \pi_{e_3}(2^{(1)}, 0^{(1)})) * (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$
$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) * (e_3, \pi_{e_3}(2^{(2)}, 0^{(2)}))$ + $(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * (e_3, \pi_{e_3}(2^{(2)}, 0^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (e_3, \pi_{e_3}(2^{(1)}, 0^{(1)})) * (e_3, \pi_{e_3}(2^{(2)}, 0^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * (e_3, \pi_{e_3}(2^{(2)}, 0^{(2)}))$	$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) * O_{\bar{T}}$ + $(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * O_{\bar{T}}$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (e_3, \pi_{e_3}(2^{(1)}, 0^{(1)})) * O_{\bar{T}}$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * O_{\bar{T}}$	$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 0^{(1)})) * (\bar{e}_3, \pi_{\bar{e}_3}(2^{(2)}, 2^{(2)}))$ + $(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)})) * (\bar{e}_3, \pi_{\bar{e}_3}(2^{(2)}, 2^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (e_3, \pi_{e_3}(2^{(1)}, 0^{(1)})) * (\bar{e}_3, \pi_{\bar{e}_3}(2^{(2)}, 2^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (e_5, \pi_{e_5}(2^{(1)}, 0^{(1)}))) * (\bar{e}_3, \pi_{\bar{e}_3}(2^{(2)}, 2^{(2)}))$

Par la propriété d'idempotence (Propriété 7.2.1), les éléments de ce bloc de la matrice sont réduits à :

$(\bar{e}_2, \Pi_{\bar{e}_2}(2^{(1)}0^{(2)}, 2^{(1)}0^{(2)})) \cdot (\bar{e}_4, \Pi_{\bar{e}_4}(2^{(1)}0^{(2)}, 2^{(1)}0^{(2)})) \cdot (e_5, \Pi_{e_5}(2^{(1)}0^{(2)}, 0^{(1)}0^{(2)}))$	$O_{\bar{T}}$	$(\bar{e}_2, \Pi_{\bar{e}_2}(2^{(1)}0^{(2)}, 2^{(1)}0^{(2)})) \cdot (e_4, \pi_{e_4}(2^{(1)}0^{(2)}, 2^{(1)}2^{(2)})) \cdot (e_5, \Pi_{e_5}(2^{(1)}2^{(2)}, 0^{(1)}2^{(2)}))$
$O_{\bar{T}}$	$(\bar{e}_4, \Pi_{\bar{e}_4}(2^{(1)}1^{(2)}, 2^{(1)}1^{(2)})) \cdot (e_5, \Pi_{e_5}(2^{(1)}1^{(2)}, 0^{(1)}1^{(2)}))$	$(e_4, \Pi_{e_4}(2^{(1)}1^{(2)}, 2^{(1)}2^{(2)})) \cdot (e_5, \Pi_{e_5}(2^{(1)}2^{(2)}, 0^{(1)}2^{(2)}))$
$(e_3, \Pi_{e_3}(2^{(1)}2^{(2)}, 0^{(1)}0^{(2)}))$	$O_{\bar{T}}$	$(\bar{e}_3, \Pi_{\bar{e}_3}(2^{(1)}2^{(2)}, 2^{(1)}2^{(2)})) \cdot (e_5, \Pi_{e_5}(2^{(1)}2^{(2)}, 0^{(1)}2^{(2)}))$

Le bloc $p^{(1)}_{2^{(1)},1^{(1)}} * P^{(2)} = (e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * P^{(2)}$dans la position $\left(\begin{smallmatrix}\cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot\\ \cdot & \bullet & \cdot\end{smallmatrix}\right)$ a les éléments suivants :

$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(0^{(2)}, 0^{(2)}))$	$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$	$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$ $+$ $(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (e_4, \pi_{e_4}(0^{(2)}, 2^{(2)}))$
$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * O_{\check{\mathcal{T}}}$	$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$	$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$
$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * (e_3, \pi_{e_3}(2^{(2)}, 0^{(2)}))$	$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * O_{\check{\mathcal{T}}}$	$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(1^{(1)}, 1^{(1)})) * (\bar{e}_3, \pi_{\bar{e}_3}(2^{(2)}, 2^{(2)}))$

Par la propriété d'idempotence (Propriété 7.2.1), les éléments de ce bloc de la matrice sont réduits à :

$O_{\check{\mathcal{T}}}$	$(e_2, \Pi_{e_2}(2^{(1)}0^{(2)}, 1^{(1)}1^{(2)})) \cdot (\bar{e}_4, \Pi_{\bar{e}_4}(1^{(1)}1^{(2)}, 1^{(1)}1^{(2)}))$	$(e_2, \Pi_{e_2}(2^{(1)}0^{(2)}, 1^{(1)}1^{(2)})) \cdot (e_4, \Pi_{e_4}(1^{(1)}1^{(2)}, 1^{(1)}2^{(2)}))$
$O_{\check{\mathcal{T}}}$	$O_{\check{\mathcal{T}}}$	$O_{\check{\mathcal{T}}}$
$O_{\check{\mathcal{T}}}$	$O_{\check{\mathcal{T}}}$	$O_{\check{\mathcal{T}}}$

Le bloc $p^{(1)}_{2^{(1)},2^{(1)}} * P^{(2)} = \big((e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) + (e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)}))\big) * P^{(2)}$.

Par la propriété de distributivité de l'opérateur de concurrence sur l'opérateur de choix, on a $(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * P^{(2)} + (e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * P^{(2)}$ dans la position $\left(\begin{smallmatrix}\| & \| & \| \\ \| & \| & \| \\ \| & \bullet & \end{smallmatrix}\right)$. Les les éléments de ce bloc sont les suivants :

$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(0^{(2)}, 0^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(0^{(2)}, 0^{(2)}))$	$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$	$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * (e_2, \pi_{e_2}(0^{(2)}, 1^{(2)})) \cdot (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$ + $(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (e_4, \pi_{e_4}(0^{(2)}, 2^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * (\bar{e}_2, \pi_{\bar{e}_2}(0^{(2)}, 0^{(2)})) \cdot (e_4, \pi_{e_4}(0^{(2)}, 2^{(2)}))$
$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * O_{\bar{\mathcal{T}}}$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * O_{\bar{\mathcal{T}}}$	$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * (\bar{e}_4, \pi_{\bar{e}_4}(1^{(2)}, 1^{(2)}))$	$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * (e_4, \pi_{e_4}(1^{(2)}, 2^{(2)}))$
$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * (e_3, \pi_{e_3}(2^{(2)}, 0^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * (e_3, \pi_{e_3}(2^{(2)}, 0^{(2)}))$	$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * O_{\bar{\mathcal{T}}}$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * O_{\bar{\mathcal{T}}}$	$(e_2, \pi_{e_2}(2^{(1)}, 1^{(1)})) \cdot (e_3, \pi_{e_3}(1^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * (\bar{e}_3, \pi_{\bar{e}_3}(2^{(2)}, 2^{(2)}))$ + $(\bar{e}_2, \pi_{\bar{e}_2}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_3, \pi_{\bar{e}_3}(2^{(1)}, 2^{(1)})) \cdot (\bar{e}_5, \pi_{\bar{e}_5}(2^{(1)}, 2^{(1)})) * (\bar{e}_3, \pi_{\bar{e}_3}(2^{(2)}, 2^{(2)}))$

Par la propriété d'idempotence (Propriété 7.2.1), les éléments de ce bloc de la matrice sont réduits à :

$(\bar{e}_2, \Pi_{\bar{e}_2}(2^{(1)}0^{(2)}, 2^{(1)}0^{(2)})) \cdot (\bar{e}_4, \Pi_{\bar{e}_4}(2^{(1)}0^{(2)}, 2^{(1)}0^{(2)})) \cdot (\bar{e}_5, \Pi_{\bar{e}_5}(2^{(1)}0^{(2)}, 2^{(1)}0^{(2)}))$	$O_{\bar{\mathcal{T}}}$	$(\bar{e}_2, \Pi_{\bar{e}_2}(2^{(1)}0^{(2)}, 2^{(1)}0^{(2)})) \cdot (e_4, \Pi_{e_4}(2^{(1)}0^{(2)}, 2^{(1)}2^{(2)})) \cdot (\bar{e}_5, \Pi_{\bar{e}_5}(2^{(1)}2^{(2)}, 2^{(1)}2^{(2)}))$
$O_{\bar{\mathcal{T}}}$	$(\bar{e}_4, \Pi_{\bar{e}_4}(2^{(1)}1^{(2)}, 2^{(1)}1^{(2)})) \cdot (\bar{e}_5, \Pi_{\bar{e}_5}(2^{(1)}1^{(2)}, 2^{(1)}1^{(2)}))$	$(e_4, \Pi_{e_4}(2^{(1)}1^{(2)}, 2^{(1)}2^{(2)})) \cdot (\bar{e}_5, \Pi_{\bar{e}_5}(2^{(1)}2^{(2)}, 2^{(1)}2^{(2)}))$
$O_{\bar{\mathcal{T}}}$	$O_{\bar{\mathcal{T}}}$	$(\bar{e}_3, \Pi_{\bar{e}_3}(2^{(1)}2^{(2)}, 2^{(1)}2^{(2)})) \cdot (\bar{e}_5, \Pi_{\bar{e}_5}(2^{(1)}2^{(2)}, 2^{(1)}2^{(2)}))$

À partir des chaînes de transitions globales du descripteur discret P (égal à la matrice de transition de l'automate global) on peut obtenir la matrice transition de la chaîne de Markov en évaluant les chaînes de transitions globales selon la définition 6.4.1 (page 109).

Évaluons par exemple la chaîne de transitions globales de la dernière ligne et dernière colonne (position $P_{[2^{(1)},2^{(2)}][2^{(1)},2^{(2)}]}$. Cette chaîne de transitions globales est égale à :

$$P_{[2^{(1)},2^{(2)}][2^{(1)},2^{(2)}]} = (\bar{e}_3, \Pi_{\bar{e}_3}(2^{(1)}2^{(2)}, 2^{(1)}2^{(2)})) \cdot (\bar{e}_5, \Pi_{\bar{e}_5}(2^{(1)}2^{(2)}, 2^{(1)}2^{(2)}))$$

Évaluons chaque tuple de transition globale séparément. Le premier tuple de transition globale $(\bar{e}_3, \Pi_{\bar{e}_3}(2^{(1)}2^{(2)}, 2^{(1)}2^{(2)}))$ a la probabilité d'occurrence de l'événement $\bar{e}_3$ égale à $1-\rho_3$. Cet événement a une probabilité d'occurrence constante ($1-\rho_3$), c'est-à-dire, sa valeur ne dépende pas de l'état global du modèle. La probabilité de routage globale (produit des probabilités de routage des tuples de transition locale) de ce tuple de transition globale est aussi constante et égale à 1.

Le deuxième tuple de transition globale $(\bar{e}_5, \Pi_{\bar{e}_5}(2^{(1)}2^{(2)}, 2^{(1)}2^{(2)}))$ a la probabilité d'occurrence de l'événement $\bar{e}_5$ égale à $1-f$, où f est la fonction :

$$f = (st\ \mathcal{A}^{(2)}\ == 2^{(2)}) * \rho_5$$

En évaluant cette fonction pour l'état globale $2^{(1)}2^{(2)}$ (état de départ du tuple de transition globale), on obtient :

$$f = \rho_5$$

Alors, la probabilité d'occurrence de ce tuple de transition globale, lorsque sa fonction est évaluée pour l'état globale $2^{(1)}2^{(2)}$, est égale à $1-\rho_5$. La probabilité de routage de ce tuple de transition globale est constante et égale à 1.

En évaluant cette chaîne de transitions globales selon la définition 6.4.1, on a l'évaluation suivante :

$$\begin{aligned}\Pi(P_{[2^{(1)},2^{(2)}][2^{(1)},2^{(2)}]}) &= (1-\rho_3(2^{(1)}2^{(2)}) \times \Pi_{\bar{e}_3}(2^{(1)}2^{(2)}, 2^{(1)}2^{(2)})(2^{(1)}2^{(2)})) \times \\ &\quad (1-f(2^{(1)}2^{(2)}), \Pi_{\bar{e}_5}(2^{(1)}2^{(2)}, 2^{(1)}2^{(2)})(2^{(1)}2^{(2)})) \\ &= (1-\rho_3)(1-\rho_5)\end{aligned}$$

En évaluant toutes les chaînes de transitions globales du descripteur P du modèle de FIG. 8.2 (page 158), on obtient la matrice de transition suivante :

$$P = \left(\begin{array}{ccc|ccc|ccc}
(1-\rho_1)(1-\rho_4) & 0 & (1-\rho_1)\rho_4 & \rho_1(1-\rho_4) & 0 & \rho_1\rho_4 & 0 & 0 & 0 \\
0 & (1-\rho_1)(1-\rho_4) & (1-\rho_1)\rho_4 & 0 & \rho_1(1-\rho_4) & \rho_1\rho_4 & 0 & 0 & 0 \\
0 & 0 & (1-\rho_1) & 0 & 0 & \rho_1 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & (1-\rho_4) & 0 & \rho_4 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & (1-\rho_4) & \rho_4 & 0 & 0 & 0 \\
\rho_3\pi_1(1-\rho_1) & 0 & 0 & \rho_3\pi_1\rho_1 & 0 & (1-\rho_3) & \rho_3\pi_2 & 0 & 0 \\
\hline
0 & 0 & (1-\rho_2)\rho_4\rho_5 & 0 & \rho_2(1-\rho_4) & \rho_2\rho_4 & (1-\rho_2)(1-\rho_4) & 0 & (1-\rho_2)\rho_4(1-\rho_5) \\
0 & 0 & \rho_4\rho_5 & 0 & 0 & 0 & 0 & (1-\rho_4) & \rho_4(1-\rho_5) \\
\rho_3 & 0 & (1-\rho_3)\rho_5 & 0 & 0 & 0 & 0 & 0 & (1-\rho_3)(1-\rho_5)
\end{array}\right) \quad (8.2)$$

8.5 Conclusion

Dans ce chapitre, on a proposé une représentation compacte pour la matrice de transition de l'automate global d'un modèle SAN. Cette représentation compacte, appelée *Descripteur discret*, est une représentation tensorielle qui utilise l'algèbre tensorielle complexe définie dans le chapitre 7.

L'association de l'algèbre tensorielle complexe et des règles de construction des chaînes de transitions locales (et par conséquence des matrices d'événements) cache à l'utilisateur toute la complexité d'un modèle à temps discret, ce qui fait de cette association (algèbre tensorielle complexe et algèbre des événements) un outil puissant pour la construction de modèles SAN à temps discret. Plus précisément, les règles de construction des chaînes de transitions locales masquent la complexité de l'occurrence simultanée de plusieurs événements dans un même automate pendant que le produit tensoriel complexe masque la modélisation de l'occurrence d'événements concurrentes (l'occurrence d'événements dans différents automates). Les cas de conflit sont gérés par la priorité et par la définition de l'ensemble d'événements possibles de chaque état.

Ce chapitre clôt cette deuxième partie de cette thèse. Nous y avons présenté une définition complète pour le formalisme SAN à temps discret, en décrivant la sémantique, en terme d'événements aléatoires et d'évolution d'un modèle SAN à temps discret, la définition d'un ensemble d'opérateurs pour représenter cette sémantique de combinaison d'événements qui nous a permis de proposer, en combinaison avec l'algèbre tensorielle de matrices, une représentation compacte d'un modèle SAN.

Troisième partie

Outil de calcul : le logiciel PEPS

Chapitre 9

Structure du logiciel PEPS

Depuis la première version du logiciel PEPS [69], plusieurs modifications ont été apportées. Étant donné que PEPS est un logiciel académique, son caractère monolithique et l'importance du code généré qui a augmenté au fur et à mesure que des multiples méthodes ont été implantées dans le logiciel, sa maintenance et l'implantation de nouvelles théories développées pour le formalisme SAN sont devenue difficiles.

PEPS2007 adopte une nouvelle structure qui vise faciliter sa maintenance mais surtout le développement de nouveau modules dans le logiciel. Cette nouvelle version est disponible sur le site web du projet :

http ://www-id.imag.fr/Logiciels/peps/

Le logiciel a été découpé en plusieurs modules indépendants. Chaque module implante une méthode ou un ensemble de méthodes. Les modules générés par le découpage du logiciel ont été classés en 3 groupes. Chaque groupe correspond à une phase dans la résolution complète d'un modèle.

Dans ce chapitre, on présente dans la première section un bref historique de l'évolution du logiciel PEPS. La section 9.2 présente les phases nécessaires à la résolution complète d'un modèle. La section 9.3 explique la méthode de découpage choisie pour la création des modules du PEPS2007. Les sections suivantes sont dédiées à la présentation de chaque phase et des modules qui la composent. Enfin, la Section 9.7 présente quelques conclusions et perspectives sur cette nouvelle structure.

9.1 Historique du logiciel PEPS

La première version du logiciel a été proposée en 1988 par Plateau, Fourneau et Lee [76]. Avec la version PEPS 2.0, le logiciel connaît de grandes modifications. Notamment, cette ver-

sion implante l'ensemble les nouveaux algorithmes proposés par Fernandes [35], tels que la méthode de multiplication vecteur-descripteur, l'agrégation algébrique et les méthodes de projection pour la solution stationnaire.

Dans cette version, avec la méthode de multiplication vecteur-descripteur, appelée *shuffle*, chaque produit d'un terme tensoriel est multiplié par une partie du vecteur, sans jamais générer aucune partie de la matrice complète. Après toutes les multiplications tensorielles, on a une multiplication complète du vecteur-matrice. L'algorithme d'agrégation algébrique consiste à réaliser des opérations tensorielles entre les automates de façon à grouper plusieurs automates en un seul. Les méthodes de projection implantées dans cette version sont cette d'Arnoldi et de GMRES [86, 92].

L'interface utilisateur du logiciel était assez rudimentaire. L'absence d'un langage de description d'un modèle SAN obligeait l'utilisateur à fournir directement les matrices et fonctions du descripteur markovien. Un deuxième problème présente dans la version PEPS 2.0 était l'obtention des indices de performance, car seulement le vecteur des probabilités de la distribution stationnaire était fourni en résultat.

9.1.1 PEPS2000

Au cours de l'année 2000 une nouvelle version du logiciel a été mise en ligne. PEPS2000 apporte des optimisations sur les algorithmes déjà en place, tel qu'une manipulation optimisée des dépendances fonctionnelles [35], et il propose aussi un langage de description d'un modèle SAN. Ce premier langage de description permet une description hiérarchique du modèle, où un modèle est composé d'automates, qui sont composés d'états, où chaque état a des transitions et chaque transition a un ensemble d'événements associé. Avec le langage de description du modèle, une fonctionnalité importante a été implantée dans PEPS 2000 : les fonctions d'intégrations. Les fonctions d'intégrations permettent de définir, par des fonctions, une fonction sur lesquels le vecteur de probabilités sera sommé. Il s'agit du calcul de la valeur moyenne de la fonction sur le vecteur de probabilité stationnaire.

9.1.2 PEPS2003

La version PEPS2003 [7] apporte plusieurs nouveautés, telles que l'utilisation des vecteurs creux, l'agrégation sémantique, l'évaluation de fonctions *just-in-time* et aussi un nouveau langage de description. L'implantation des structures de vecteurs creux dans PEPS a permis le calcul uniquement sur l'espace d'états atteignables. Ce format réduit de façon considérable l'espace de stockage pour les modèles creux. L'agrégation sémantique d'automates identiques (répliqua) consiste de la représentation par un seul état le nombre d'automates dans le même état de l'automate original [6].

Une autre fonctionnalité implantée dans PEPS2003 est l'évaluation de fonctions *just-in-time*. Cette méthode consiste à générer pour chaque fonction déclarée dans le modèle SAN, un

code C++. Le code C++ d'une fonction est compilé et lié aux méthodes de résolution du PEPS. Ce code est appelé à chaque fois que la fonction est évaluée. Cette amélioration représente, dans certains modèles, jusqu'à 30% du temps de calcul par rapport à l'ancienne méthode d'évaluation des fonctions par interprétation.

Le nouveau langage de description implanté dans cette version est plus intuitif et permet une description plus compacte, car ce nouveau langage commence à implanter quelques fonctionnalités qui permettent la réplication *identique* des automates et des états. Cependant les possibilités de réplications sont très limitées et se restreignent à des modèles assez simples.

9.1.3 PEPS2007

Dans la version de 2007 [13], une nouvelle structure a été adoptée pour le logiciel. PEPS2007 a été découpé en un ensemble de modules indépendants. Chaque module implante une phase/étape de la résolution complète d'un modèle. Cette approche améliore la maintenance et le développement du logiciel. Avec cette approche, des nouvelles méthodes peuvent être développées et testées indépendamment de la version stable de PEPS de façon à réduire l'inclusion de bogues et la cohérence du logiciel. Autre avantage, cette version permet un développement localement et/ou logiquement distribué.

La section suivante présente les phases et chemins possibles pour la résolution complète d'un modèle.

9.2 Résolution complète d'un modèle SAN

La résolution complète d'un modèle SAN passe par plusieurs phases et différents chemins peuvent être employés pour obtenir les indices de performance souhaités. FIG. 9.1 présente les deux chemins possibles pour résoudre un modèle.

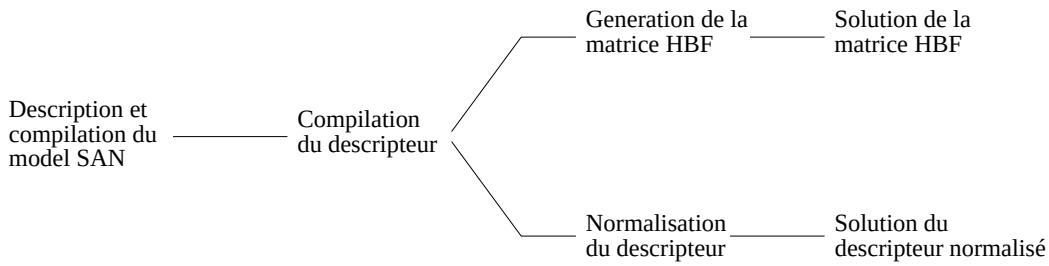


FIG. 9.1 – Étapes nécessaires pour la résolution d'un modèle.

La première phase consiste à décrire et compiler le modèle SAN. Cette phase est appelée *analyse syntaxique*. La deuxième phase, appelée *structures de données* consiste à manipuler le descripteur markovien obtenu à partir de la compilation du modèle de façon à l'optimiser. La

dernière phase, appelée *méthodes de résolution* consiste à solutionner effectivement le modèle, soit directement sur le descripteur markovien, soit sur la matrice au format HBF (*Harwell-Boeing Format*).

Les phases nécessaires à la résolution du modèle sont utilisées comme base pour le découpage du logiciel. La méthode de découpage est présentée dans la section suivante.

9.3 Méthode de découpage

Différentes versions du descripteur markovien sont utilisées lors de la compilation du modèle SAN jusqu'à la version normalisée utilisée dans la résolution du modèle (multiplication vecteur-descripteur).

Chaque version du descripteur est composée d'un dictionnaire d'identifiants (fonctions, automates, états et événements), d'un tableau des fonctions, de la fonction d'atteignabilité, d'un tableau des fonctions d'intégration et d'un ensemble des matrices.

La première version du descripteur markovien, appelé *Descripteur Markovien Plein*, est générée par la compilation fichier `.san` du modèle SAN dans la phase d'analyse syntaxique. La deuxième version du descripteur markovien, appelé *Descripteur Markovien Creux*, est obtenue par la transformation des matrices pleines du descripteur markovien en matrices creuses (première étape de la phase de structures de données). La dernière version du descripteur markovien est obtenue par la normalisation du descripteur markovien précédent (deuxième étape de la phase de structures de données). Ce descripteur, appelé *Descripteur Markovien Normalisé*, est utilisé pour les méthodes de résolution (phase des méthodes de résolution).

Le découpage du logiciel PEPS suit l'évolution du descripteur markovien. Chaque module résultant de ce découpage est responsable pour une phase/étape de la transformation du descripteur markovien ou pour la résolution du modèle. De cette façon, on a découpé le logiciel de manière à garder dans chaque module uniquement les classes nécessaires à une phase/étape. Il faut remarquer que parfois, différents modules implantent la même phase/étape, c'est le cas, par exemple, des modules pour l'analyse syntaxique, où chaque module a un langage de description différents.

La version 2007 du logiciel PEPS est composée de 29 classes et chaque classe participe à un ou plusieurs modules. TAB. 9.3 présente la liste de classes utilisées dans PEPS et aussi leurs relations de dépendance.

À partir de TAB. 9.3, on peut aisément séparer certaines classes, par exemple, les GRAMMAR, AUTOMATON, STATE, EDIC, NDIC et FUNC sont uniquement dépendants entre elles et sont utilisées uniquement dans les modules de la phase d'analyse syntaxique. Cette indépendance des autres classes du logiciel rend facile la création des modules de cette phase.

Cependant, les autres classes du logiciel sont fortement liées et on a pu constater qui un

Classe	Description	Utilise la classe ...	Est utilisée par la classe ...
Analyse Syntaxique			
AUTOMATON	automates	STATE	GRAMMAR
DOM	définition des domaines		GRAMMAR
EDIC	dictionaire d'événements		GRAMMAR
FUNC	dictionaires de fonctions		GRAMMAR
GRAMMAR	regles de grammaires	AUTOMATON, DOM, NDIC, EDIC, FUNC	
NDIC	dictionaire d'automates et d'états		GRAMMAR
STATE	description d'états		AUTOMATON
Classes répliquées			
aut_list	liste d'automates	aut_set, aut_st, d_graph	aut_set, aut_st, CND, d_graph, jumps, VCT
aut_set	ensemble d'automates	aut_list, aut_st	aut_list, aut_st, d_graph, elem, FTB func_tree, RSS, san_mat, st_map
aut_st	automates ou états	aut_set, aut_list, jumps	aut_list, aut_set, CND, DCT, d_graph, DSC, elem, FTB, func_tree, HBF, INF, jumps, LUD, RSS, san_mat, st_map, VCT
bools	vecteur de booleans		full_mat, RSS, san_mat, sp_mat, st_map
DCT	Dictionnaire	aut_st	
d_graph	graphes de dépendance fonctionnelle	aut_list, aut_set, aut_st	aut_list
elem	éléments des matrices	aut_set, aut_st, st_map	elem_list, san_mat
elem_list	liste d'éléments	elem	scr_ft
FTB	tableau de fonctions	func_tree, aut_set, aut_st, st_map	scr_ft
full_mat	matrice d'éléments constants au format plein	bools	
func_tree	arbre d'évaluation d'une fonction	aut_set, aut_st	FTB, INF, src_ft
INF	tableau des fonctions d'intégration (résultats)	aut_st, func_tree	
jumps	tableau de saut de permutation	aut_list, aut_st	aut_st, CND, LUD, VCT
RSS	espace d'états atteignable	aut_set, aut_st, bools, func_tree	
san_mat	matrice ordinaire d'éléments	aut_set, aut_st, elem, st_map, bools	CND, DSC, sp_mat, VCT
scr_ft	scratch du tableau de fonctions	FTB, func_tree, elem_list	
sp_mat	matrice d'éléments constants au format creux	san_mat, st_map, bools	LUD
st_map	map d'états	aut_set, aut_st, bools	CND, DSC, elem, FTB, sp_mat, VCT
Classes non-répliquées			
CND	descripteur normalisé	aut_st, jumps, st_map, aut_list, san_mat	VCT
DSC	descripteur	aut_st, san_mat, st_map	
HBF	matrices creuses au format Harwell-Boeing	aut_st	VCT
VCT	vecteurs de probabilités	aut_list, aut_st, jumps, san_mat, st_map, HBF, CND	

TAB. 9.1 – Relation des dépendances de classes

ensemble de 19 classes sont nécessaires à tous les modules qui suivent la compilation du fichier `.san`. Autrement dit, ces classes sont utilisées dans la représentation du descripteur markovien plein, du descripteur markovien creux et aussi du descripteur markovien normalisé. Ces classes sont en gras dans TAB. 9.3.

Les 4 dernières classes (CND, DSC, HBF et VCT) sont utilisées uniquement dans la partie de la résolution complète du modèle.

9.3.1 Réplication du code

Pour l'ensemble de classes utilisées dans plusieurs modules, deux options de découpage se sont présentées : la création d'un noyau central ou la réplication du code dans chaque module.

La création d'un noyau central, commun à tous les modules, a l'avantage que les corrections nécessaires à cet ensemble de classes sont réalisées une seule fois. Cependant, cet avantage n'est pas crucial car cet ensemble de classes a été déjà largement utilisée et on considère que peu de modifications ou corrections seront nécessaires.

La réplication du code dans chaque module apporte plusieurs avantages : toutes les classes nécessaires au module sont au même endroit, les modifications nécessaires à de futurs modules n'ont pas d'influence sur les modules déjà existants, autrement dit, chaque module est complètement indépendant.

On a choisi la réplication du code pour le découpage du logiciel car cette méthode est la méthode qui s'approche le plus de la proposition de découpage du logiciel qu'on propose. De cette façon, les 19 classes mentionnées précédemment, ont été répliquées dans chaque module résultant du découpage, à l'exception des modules d'analyse syntaxique, qui n'utilisent pas ces classes.

Les sections suivantes présentent les modules de chaque phase de la résolution complète d'un modèle.

9.4 Analyse syntaxique

La phase d'analyse syntaxique consiste à décrire un modèle SAN selon les règles du langage de description et de compilation de ce modèle. La compilation du modèle transforme le modèle SAN dans un ensemble de fichiers qui décrivent le descripteur markovien du modèle.

Différents langages de description peuvent être utilisés. La version actuelle du PEPS2007 accepte 2 langages de description, chaque langage a été implanté dans un module différent.

Le *Langage de description* SAN implanté dans le module appelé *compile_san* est le langage standard utilisé dans PEPS2007. Ce langage est décrit dans l'annexe B.

Un deuxième module (*compile_phsan*) implante une extension du langage standard pour inclure la description des événements phase-type dans le modèle. Ce langage a été proposé par Sbeity et une description détaillée est dans [89].

Les deux modules de cette phase reçoivent un fichier du type `.san` qui suit les règles de description et génèrent, après l'analyse syntaxique, un ensemble de fichiers décrivant le descripteur markovien plein, dans le format proposé dans [34].

9.5 Structures de données

Les modules inclus dans la phase "*structure de données*" manipulent le descripteur markovien de façon à le transformer pour l'adapter à la phase de "*méthodes de résolution*".

On peut classer les modules de cette phase en 2 groupes, divisés selon la structure de stockage utilisée : tensorielle et HBF.

9.5.1 Structure tensorielle

Deux étapes sont nécessaires pour préparer le descripteur markovien plein obtenu par la compilation du modèle SAN pour la phase "méthodes de résolution" : compilation du descripteur et normalisation du descripteur.

Compilation du descripteur

Cette étape de traitement de "structure de données" transforme le descripteur markovien plein stocké en un ensemble des petites matrices au format plein en des matrices stockées au format HBF. Cette conversion a pour but de réduire le nombre de multiplications inutiles (multiplication par zéro) au moment de la résolution du modèle, vu que, assez souvent, pour des petites matrices le format HBF prend plus de place de stockage que le format plein. Notons que la matrice de transition de la chaîne de Markov équivalente continue à être représentée en forme tensorielle, et uniquement l'ensemble de matrices du descripteur markovien utilise le format HBF.

Cependant, les méthodes les plus importantes du module de cette étape sont les méthodes d'agrégation (algébrique et sémantique) en charge des groupement des automates. Le groupement d'automates par agrégation algébrique est réalisé par l'application des opérations de somme et de produit tensoriel sur un ensemble matrices du descripteur markovien. Avec ce type d'agrégation on réduit le nombre d'automates du modèle mais on a des automates plus grands [34]. La méthode d'agrégation sémantique peut être utilisée uniquement sur un ensemble d'automates identiques (replicas). Cette méthode permet de représenter avec un seul automate le nombre d'automates dans chaque état des automates agrégés [6]. Cette méthode réduit de

façon significative le nombre d'états du modèle, mais elle est applicable à un ensemble réduit de modèles.

Le module qui réalise l'étape de compilation du descripteur est appelé *compile_dsc*. Il compile l'ensemble des fichiers générés par le module *compile_san* pour décrire le descripteur markovien plein et génère un ensemble de fichiers qui décrivent le descripteur markovien creux.

Normalisation du descripteur

La dernière étape de la phase *traitement de structures de données* normalise le descripteur markovien creux et réalise le pré-calcul de la diagonale du descripteur.

Le pré-calcul de la diagonale du descripteur consiste à extraire toutes les matrices de normalisation des événements synchronisants et les éléments diagonaux des matrices des événements locaux. Ces éléments sont utilisés pour construire le vecteur diagonal utilisé dans les méthodes de résolution. Cette technique améliore la performance des méthodes de résolution. Une description détaillée de la méthode de pré-calcul de la diagonale du descripteur est décrite par Fernandes [34].

En plus de la normalisation du descripteur et du pré-calcul de la diagonale, les modules de cette étape font une analyse du modèle à la recherche de possibles optimisations, telles que, le remplacement de fonctions par des valeurs constantes (lorsque la fonction a la même valeur sur tous l'espace d'états) et la génération de l'ordre de multiplication des matrices de façon à minimiser l'évaluation des fonctions.

Cette étape de normalisation et d'optimisation est implantée par deux modules : *norm_dsc_ex* et *norm_dsc_sp*. Ces deux modules implantent les mêmes méthodes et techniques, ils utilisent l'ensemble de fichiers qui décrivent le descripteur markovien creux pour générer l'ensemble des fichiers qui décrivent le descripteur markovien normalisé. Cependant le module *norm_dsc_ex* travaille sur l'espace d'état produit et génère le vecteur diagonal avec cette taille, alors que le module *norm_dsc_sp* travaille sur l'espace d'états atteignables et le vecteur diagonal généré a cette taille.

9.5.2 Structure HBF

Un autre chemin possible pour résoudre un modèle est de générer directement la chaîne de Markov équivalente au modèle SAN. Après le découpage du logiciel, le module *gen_hbf* réalise la construction de la chaîne de Markov équivalente au descripteur markovien plein obtenu après la compilation du modèle SAN.

La matrice de transition de la chaîne de Markov équivalente au modèle SAN est stocké sous le format HBF (*Harwell-Boeing Format*). Optionnellement, ce module peut convertir la matrice générée pour qu'elle soit compatible avec le logiciel MARCA [92, 93].

9.6 Méthodes de résolution

Plusieurs méthodes de résolution sont implantées dans PEPS2007, telles que la méthode de la puissance, Arnoldi et GMRES pour la solution stationnaire et la méthode d'uniformisation pour les solutions transitoires.

Comme pour la phase de structures de données, on peut grouper les modules de cette phase selon le type de structure utilisée pour le stockage : *tensorielle* et HBF (*Harwell-Boeing Format*).

9.6.1 Structure tensorielle

Les modules de cette phase qui utilisent une structure tensorielle sont basés sur la méthode de *shuffle* [35]. La méthode de *shuffle* permet de réaliser des multiplications vecteur-descripteur sans jamais avoir à générer la matrice de transition de la chaîne de Markov. Autrement dit, ces modules travaillent directement sur le descripteur markovien normalisé du modèle.

Les méthodes de résolution qui utilisent cette structure tensorielle sont implantées dans trois modules : *solve_cnd_ex*, *solve_cnd_sp* et *solve_cnd_pav*.

Les modules *solve_cnd_ex* et *solve_cnd_ex* implantent quelques méthodes de résolution stationnaire classique : puissance, Arnoldi et GMRES et la méthode d'uniformisation pour des solutions transitoires. La différence entre ces deux modules est le format du vecteur de probabilité utilisé. Alors que le module *solve_cnd_ex* utilisé un vecteur plein, donc de la taille de l'espace d'état produit du modèle, pour stocker le vecteur de probabilité, le module *solve_cnd_sp* utilise un vecteur creux, où uniquement les probabilités des états atteignables sont stockées. L'utilisation des vecteur creux pour le stockage du vecteur de probabilités a été proposé par Benoit dans [9] et apporte plusieurs avantages lorsque l'espace d'états atteignables est beaucoup plus petit que l'espace d'états produit (la limité est à 50%).

Il faut remarquer que le *solve_cnd_ex* utilise le descripteur normalisé généré par le module *norm_dsc_ex*, où le vecteur diagonal a la taille de l'espace d'états produit, alors que *solve_cnd_sp* utilise le descripteur normalisé généré par le module *norm_dsc_sp*, avec un vecteur diagonal ayant uniquement les états atteignables. De la même façon, le résultat rendu par le module *solve_cnd_ex* est un vecteur de probabilités au format plein, alors que le module *solve_cnd_sp* rend un vecteur de probabilités au format creux. Optionnellement, les deux modules peuvent calculer les indices de performances sur les fonctions résultats définies dans le modèle.

Le troisième module implante plusieurs méthodes pour le calcul de la disponibilité ponctuelle. Ces méthodes sont décrites en détails dans le Chapitre 3. Ce module rend la valeur de la disponibilité ponctuelle à un instant t spécifié par l'utilisateur.

Notons que ce module a été conçu après le découpage du logiciel et il faut noter le gain de cette nouvelle structure du logiciel, car dans ce module, certaines méthodes utilisée pour le

calcul de la disponibilité ponctuelle emploient la multiplication à droite (descripteur-vecteur). La création d'un module indépendant a permis ce changement, sans toucher aux méthodes et classes utilisées aussi pour les autres modules.

9.6.2 Structure HBF

Un seul module implante la résolution du modèle en utilisant la matrice de transition stocké au format HBF. Ce module utilise la matrice de transition générée par le module *gen_hbf* et uniquement la méthode de la puissance est implantée.

De façon similaire au modules *solve_cnd_ex* et *solve_cnd_sp*, ce module rend comme résultat un vecteur de probabilités, mais contrairement au modules précédents, le calcul direct des indices de performances n'est plus possible, car les informations sur les identifiants d'états ont été perdu au moment de la génération de la matrice.

9.7 Conclusion

On a présenté dans ce chapitre la nouvelle version du logiciel PEPS. Cette nouvelle version, outre un nouveau langage de description (Annexe B) et des méthodes pour le calcul de la disponibilité ponctuelle (Chapitre 3), présente une toute nouvelle structure.

Cette structure améliore la maintenance et le développement du logiciel. Avec cette nouvelle structure, de nouvelles méthodes peuvent être développées et testées indépendamment de la version stable de PEPS. Actuellement, plusieurs modules expérimentaux implantent et testent de nouvelles méthodes, par exemple, le développement d'une structure basée en diagrammes de décision multi-valués (MDD - *Multi-valued Decision Diagrams*) [88] pour la représentation de l'espace d'états atteignables et la nouvelle méthode *split* pour la multiplication vecteur-descripteur [22]. De cette façon, on réduit l'inclusion de bogues et on garde la cohérence du logiciel.

Un autre avantage de cette nouvelle structure est la possibilité d'un développement local et/ou logiquement distribuée, car l 'implantation de nouvelles méthodes dans une même phase de la résolution du modèle, seront implantées sur des modules complètement indépendants.

On espère qu'avec cette nouvelle structure, le logiciel PEPS évolue parallèlement et plus rapidement dans les différents équipes qui travaillent sur le formalisme SAN.

Chapitre 10

Expressivité du logiciel

Dans ce chapitre, on présente le nouveau langage de description utilisé pour définir un modèle SAN. Ce nouveau langage de description a été pensé pour combler les difficultés présentes dans les anciennes versions du logiciel PEPS.

Le nouveau langage de description ne change pas le pouvoir d'expressivité apporté par la version utilisée dans PEPS2003. D'ailleurs, le nouveau langage a pour base le langage défini dans cette version là.

Ce nouveau langage a pour but de simplifier la description de modèles avec un grand nombre de composants répliqués.

10.1 Rappel des concepts développés sur PEPS2003

À partir de la version 2003 le langage de description du logiciel PEPS [7, 69] chercher à définir structures qui rendent plus facile les réplications d'automates, d'états, de transitions et d'événements. Pour facilement créer de modèles avec des structures répliquées, on a adopté le concept de modèle *paramétrable*. Autrement dit, il s'agit de définir un semble de paramètres qui sont utilisés pour définir le nombre de réplications de chaque structure du modèle.

De cette façon, un modèle avec réplications peut être facilement redimensionné en changent uniquement quelques paramètres. Pour rendre ce concept de modèle paramétrable possible, la structure du langage de description du PEPS2003 est divisée en 5 blocs : "**identifiers**", "**events**", "**reachability**", "**network**" et "**results**".

Dans le bloc "**identifiers**" sont déclarés tous les paramètres qui sont utilisés dans la description du modèle. Chaque paramètre est associé à un identifiant.

L'ensemble d'événements associés aux transitions du modèle sont déclarés dans le bloc "**events**". Un événement peut être répliqué et le nombre de réplications est facilement paramé-

trable en utilisant les identifiants précédemment déclarés.

Le bloc "**reachability**" définit les états atteignable du modèle. Cet ensemble d'état est défini par une fonction.

"**network**" est le bloc plus important de la description. La structure du modèle est décrite dans ce bloc. Cette description est structurée de façon hiérarchique, où, le modèle est un réseau d'automates stochastiques constitué d'un ensemble d'automates ; chaque automate est constitué d'un ensemble d'états ; chaque état a un ensemble de transitions sortantes ; chaque transition a un ensemble d'événements associés.

FIG. 10.1 présente graphiquement cette structure hiérarchique.

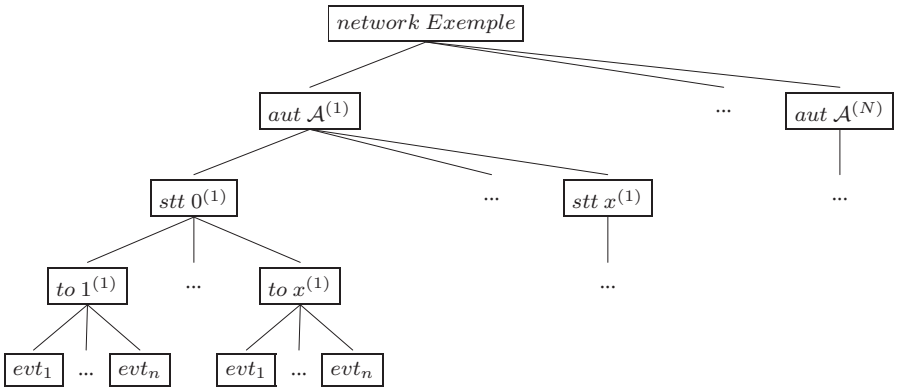


FIG. 10.1 – Structure hiérarchique de bloc "**network**"

Comme pour les événements, la réplication d'automates et/ou d'états est facilement paramétrable par l'utilisation des identifiants déclarés au bloc "**identifiers**".

Le dernier bloc "**results**" contient les fonctions des indices de performance qu'on veut calculer.

10.2 Problèmes dans le langage de description du PEPS2003

On a vu que des fonctionnalités qui permettent une description compacte des modèles avec des composants répliqués étaient déjà présentes dans PEPS2003. Cependant, cet ancien langage permet uniquement une réplication identique des automates, des états et de la déclaration des événements.

Lorsqu'on a un modèle avec des composants similaires, mais qui interagissent avec des composants différents, on était obligé de décrire chaque composant séparément, car il était impossible de modifier une caractéristique quelconque.

Un autre problème rencontré dans l'ancien langage est l'impossibilité de décrire de multi-

ples transitions sortantes d'une façon compacte. Chaque transition sortante devait être décrite individuellement. Ce même problème est présent dans l'association de multiples événements à une transition.

Enfin, l'impossibilité de réplication de fonctions dans le langage du PEPS2003 obligeait l'utilisateur à décrire individuellement chaque fonction, même si d'une fonction à l'autre, uniquement l'indice de l'automate/l'état changeait.

Dans le but de permettre des réplications plus flexibles, où les composants ne sont pas identiques, dans la section suivante, on propose 3 nouvelles fonctions qui nous permettront d'exprimer assez facilement plusieurs cas de réplications, là où avant une description individuelle de chaque composant était nécessaire.

10.3 Proposition : nouvelles fonctions pour la réplication

On va introduire ici un nouveau type de fonction, appelée "*fonction de description*". Trois nouvelles fonctions ont été proposées pour ce nouveau langage. Ces fonctions ont pour but de modéliser l'automate courant dans une réplication d'automates, l'état courant dans une réplication d'états et aussi l'état d'arrivée d'une transition dans une réplication d'état.

Automate courant - *at* La fonction *at* permet d'obtenir l'indice de l'automate qui est en train d'être traité. Cette fonction est utile pour différencier chaque automate dans un ensemble d'automates répliqués. Elle nous permet de définir des ensembles différents de transitions et d'événements selon l'automate. Cela est pratique pour modéliser un ensemble d'automates avec le même comportement, mais qui utilisent des événements différents. On utilise cette fonction pour faciliter la description des modèles où l'indice des événements dépend de l'automate. Par exemple, le modelage d'un ensemble de station de travail reliées par un réseau en anneau où chaque station doit synchroniser avec la station suivante. L'indice de la station courante récupéré par la fonction *at* est utilisé pour indexer les événements de synchronisation. Cet exemple est présenté dans FIG. 10.9 dans la section 10.7.1.

La fonction *at* n'a pas de paramètre. Cependant, il faut faire attention à son usage, car la fonction a des comportements différents si elle est utilisée pour obtenir l'automate courant au sein d'un ensemble d'automates répliqués ou dans un automate non-répliqué. Lorsqu'elle est utilisée pour obtenir l'indice de l'automate en cours dans un ensemble de réplication, elle donne comme résultat l'indice de cet automate dans le domaine de réplication. Lorsqu'elle apparaît dans un automate non-répliqué, l'indice rendu est l'indice interne de l'automate[1]. Par exemple, dans un modèle SAN avec 8 automates définis de la façon suivante : un automate A répliqués dans le domaine $[1..3]$ ($A[1]$, $A[2]$ et $A[3]$), un automate B non-répliqué et un automate

[1] L'indice interne de l'automate est un indice unique attribué par le logiciel à chaque automate. Cet indice est attribué de façon séquentiel et le premier automate décrit a l'indice interne 0, le deuxième l'indice 1 et ainsi de suite.

C répliqué dans le domaine $[6..9]$ ($C[6]$, $C[7]$, $C[8]$ et $C[9]$). Les indices donnés par la fonction *at* pour ce modèle sont : $1, 2, 3, 3, 6, 7, 8$ et 9. Pour ce même modèle, si on change l'ordre de description des automates en décrivent l'automate B en dernier, la fonction *at* rend les indices $1, 2, 3, 6, 7, 8, 9$ et 7.

État courant - *sts* La fonction *sts* permet d'obtenir l'indice de l'état courant dans un automate. Cette fonction est utile dans un ensemble d'états répliqués et permet de définir l'ensemble de transitions et d'événements qui sont utilisés par rapport à l'état courant. Cette fonction peut être utilisée pour définir l'indice d'un événement utilisé selon l'état courant, mais aussi dans une fonction de condition dans la définition d'une transition et d'un événement.

Cette fonction permet de choisir les transitions et événements selon l'état courant.

État d'arrivée - *std* La fonction *std* permet d'obtenir l'indice de l'état d'arrivée d'une transition. Cette fonction peut être utilisée pour définir l'indice d'un événement ou dans une condition pour la définition d'une transition ou d'un événement. De façon similaire aux fonctions précédentes, cette fonction est utilisée dans un ensemble d'états répliqués.

Cette fonction permet de choisir les transitions et événements selon l'état d'arrivée.

Des exemples d'utilisation sont présentés dans les sections suivantes.

10.4 Réplication de fonctions et événements

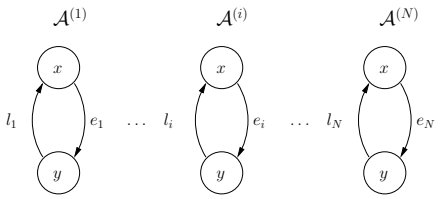
Les fonctions de description peuvent être utilisées pour exprimer une condition pour la définition d'une transition ou d'un événement, mais surtout pour calculer les indices des événements. Cependant, pour qu'on puisse utiliser les indices obtenus par ces fonctions, il faut que les événements soient aussi répliqués.

La réplication d'événements est présente dans le langage de description depuis la version PEPS2003. Cependant, dans cette version une seule dimension de réplication (vecteur $A[i]$) était permise. Dans PEPS2007 on peut répliquer un événement jusqu'à 3 dimensions ($A[i][j][k]$). Une réplication en 3 dimensions permet d'indicer plus facilement l'automate, l'état et la transition.

Contrairement au langage de description défini pour PEPS2003, le langage de PEPS2007 nous permet d'utiliser différents taux de déclenchement pour les événements répliqués. Si un seul taux de déclenchement est défini, ce taux sera utilisé pour tous les événements définis dans cet ensemble. On peut associer différents taux aux événements répliqués en spécifiant une liste d'identifiants ou une fonction répliquée. Dans les deux cas, le premier identifiant sera associé au premier événement répliqué et ainsi de suite. Si le nombre d'identifiants spécifié est plus petit que le nombre d'événement, on revient au début de la liste.

Un autre avantage proposé dans ce nouveau langage est la réplication de fonctions. La réplication de fonctions nous permet créer un ensemble de fonctions similaires mais qui sont évaluée sur des automates différents. Chaque fonction peut être répliquée en 3 dimensions au maximum.

L'exemple suivant présente des cas où la réplication de fonctions est utile. Dans cet exemple, on propose un modèle SAN assez simple où chaque automate a 2 événements locaux associés. Un de ces événements locaux a un taux fonctionnel. Ce modèle est décrit dans FIG. 10.2.



Type	Event	Rate	Type	Event	Rate
loc	e_1	f_1	loc	l_1	μ
loc	e_i	f_i	loc	l_i	μ
⋮	⋮	⋮	⋮	⋮	⋮
loc	e_N	f_N	loc	l_N	μ

$$f[i] = (nb\,[\mathcal{A}^{(1)} .. \mathcal{A}^{(i-1)}]\; x \;==\; (i-1)) * \alpha$$

FIG. 10.2 – réplications d'automate avec des événements avec différents taux

Dans ce modèle, on voit 3 cas de réplications : d'automates, d'événements et de fonctions. On va explorer dans ce modèle la réplication d'événements associée à différents taux de déclenchement et par conséquence, la réplication de fonctions.

Le but de ce modèle est de montrer l'utilisation de la réplication de fonctions. La réplication de fonctions permet de définir un ensemble des fonctions identiques où les automates qui sont les arguments de la fonction sont différents.

Ce modèle est décrit par le code ci-dessous. La description d'un seul automate est nécessaire. L'indice des événements de chaque automate est calculé par la fonction `i`. La fonction `i` récupère l'indice de l'automate dans la réplication.

```
identifiers
  N      = [1..10];  // domaine de réplication
  i      = at;       // indice de l'automate courant
  f[N]   = ((nb [[A[1]..A[at]] y) == (at-1)) * lambda;
                     // f[i] rend lambda si le nombre
                     // d'automates de A[1] jusqu'à A[i]
                     // dans l'état y est égal à (i-1)
  lambda = 0.1;      // taux de déclenchement des
  mu     = 0.2;      // événements

events
  loc e[N]  (f[N]);
  loc l[N]  (mu);
```

```
reachability = 1;    // tout les états du modèle sont
                     // atteignables

network aut_rep (continuous)
  aut A[N]           // automate répliqué 10 fois [1..10]
    stt x            // état x
      to (y) e[i]
    stt y            // état y
      to (x) l[i]
```

Ce modèle utilise différents taux de déclenchement. Pour associer différents de taux de déclenchement à un ensemble d'événements répliqués, on peut expliciter une liste d'identifiants de fonctions, *i.e.*`loc e[N]  (f[1] f[2] f[3])`, ou on peut définir un domaine de fonctions répliquées, *i.e.*, `loc e[N]  (f[N])`. Dans notre code, on définit les taux de déclenchement de chaque événement répliqué `e` en associant la fonction répliqué `f` de même indice.

La réplication d'une fonction est faite par l'association d'un domaine de réplication à l'identifiant de la fonction. Les indices de ce domaine sont utilisés dans la création de chaque fonction répliquée. Dans cet exemple, la réplication de la fonction `f[N]`, la fonction `at` sera remplacée par les indices de `N`. Ainsi, on a la fonction répliqué `f[2] = ((nb [[A[1]..A[2]] y) == (2-1)) * lambda;`.

La syntaxe exacte de réplications d'une fonction se trouve dans la section B.1 de l'annexe B.

La possibilité de réplication des fonctions, des événements et des automates dans le nouveau langage a permis une description assez compacte du modèle. Pour décrire ce même modèle dans le langage défini dans PEPS2003, on était obligé de décrire chaque fonction, événement et automate individuellement.

Les sections suivantes présentent quelques cas de réplications qu'on peut trouver sur des modèles et la façon de les exprimer dans ce nouveau langage.

10.5 Réplication d'automates et d'états

La réplication d'automates et d'états est déjà présente dans les anciennes versions du PEPS. Cependant, les automates et les états était répliqués de façon identiques. Autrement dit, un automate et/ou un état était répliqué avec exactement les mêmes transitions et les mêmes événements associés. La seule contrainte imposée dans ce nouveau langage est que tous les automates répliqués dans une domaine doivent avoir le même nombre d'état, car on considère que les automates répliqués représentent de composants similaires, donc avoir les mêmes états est une contrainte raisonnable.

La réplication d'automates et d'états se fait exactement de la même façon que dans PEPS2003. Pour répliquer un automate, on doit simplement ajouter un domaine de réplication à la déclaration de l'automate. Par exemple, dans l'exemple précèdent, on a répliqué l'automate `A` en ajoutant le domaine `[N]` à la déclaration de l'automate (`aut A[N]`).

De façon similaire, on ajoute un domaine de réplication à la déclaration d'un état pour qu'il soit répliqué. Par exemple, pour répliquer un état `S` dans un domaine `M` où `M = [1..10]`, on déclare `stt S[M]`.

La possibilité de création de répliques non-identiques d'automates et d'états a des conséquences sur la flexibilité de construction de transitions et sur l'association des événements à ces transitions. On explore donc dans les sections suivantes plusieurs cas de réplications de transitions et événements.

10.6 Réplication de transitions

Par défaut, les transitions sortantes d'un état, ainsi que événements associés sont répliqués de façon identique lors de la réplication d'un automate ou d'un état. Dans les cas suivants, on présente un ensemble de situations possibles lors de la réplication d'un état et d'une transition.

On peut classer les déclarations des transitions sortantes en deux groupes. Les états qui ont des transitions sortantes simples et ceux qui ont des transitions sortantes répliquées.

10.6.1 Transition simples

On considère une déclaration de transition simple, toute transition sortante vers un seul état à la fois. On peut avoir deux types de transitions simples. Soit vers un état unique, soit vers un état répliqué.

Transition vers un état unique

FIG. 10.3 représente un automate où tous les transitions sortantes d'un état répliqué vont vers le même état, c'est-à-dire, un état fixe. Ce type de transition apparaît dans plusieurs modèles, par exemple, pour modéliser la panne d'une mémoire tampon.

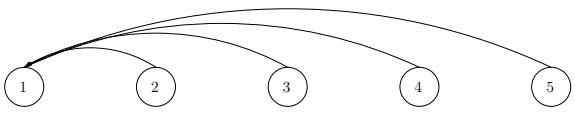


FIG. 10.3 – Réplications d'états et transitions vers un état fixe

Le code ci-dessous décrit cet automate. Les deux dernières lignes représentent la réplication des états et la transition de chaque état vers un état fixe. Cet état fixe peut être un état répliqué, comme dans cet exemple, ou n'importe quel état de l'automate. Dans cet exemple, l'état est indiqué par la fonction `i`, qui a la valeur constante 1.

```
identifiers
  N  = [1..5];                  // domaine de réplication
  i  = 1;                       // indice constant
  mu = 0.5                      // taux de déclenchement
                                // de l'événement e

events
  syn e  (mu);

reachability = 1;

network aut_rep (continuous)
  aut A
    stt s[N]                    // état répliqué 5 fois [1..5]
      to (s[i]) e               // transition vers l'état s[1]
```

Transition vers une réplication d'état

L'exemple suivant représente un automate où les états d'arrivée des transitions sortantes d'un état répliqué sont calculées par rapport à l'indice de chaque état.

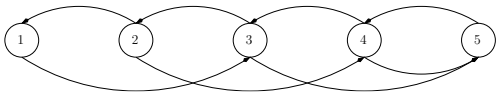


FIG. 10.4 – réplications d'états et transitions vers une réplication d'état

Dans l'automate présenté dans FIG. 10.4, chaque état répliqué a 1 ou 2 transitions sortantes selon sa position dans l'ensemble d'états répliqués. Le premier type de transition se fait vers l'état précédent. Autrement dit, vers l'état dont l'indice est décrémenté de 1 par rapport à l'indice de l'état courant. Ces transitions sont décrites par la ligne `to (- -) e` dans le code ci-dessous. Le deuxième type de transition de ce modèle se fait vers l'état d'indice $+2$ par rapport à l'indice de l'état courant, sauf si l'état d'arrivée est en dehors de l'ensemble d'états répliqués. Dans ce cas, l'état d'arrivée doit être le dernier état de l'ensemble d'états répliqués. Ce dernier type de transition est décrit par la ligne `to (s[i]) e`, où l'indice de l'état d'arrivée est calculé par la fonction `i`.

```
identifiers
```

```
  N      = [1..5];            // domaine de réplication
  i      = min((sts + 2), 5); // indice de l'état d'arrivée
  mu     = 0.5;               // taux de déclenchement des
  lambda = 1;                 // événements

events
  syn s  (mu);
  loc l  (lambda);

reachability = 1;

network aut_rep (continuous)
  aut A
    stt s[N]              // état répliqué 5 fois [1..5]
      to (--)   e         // transition vers l'état précédent
      to (s[i]) e         // transition vers l'état courant +2
```

La description de transition vers un état répliqué était déjà possible dans PEPS2003 par l'utilisation de primitives de positionnement relatif simples : (- -, ++, ==)[2]. Cependant, l'utilisation d'une fonction pour calculer l'indice de l'état d'arrivée d'une transition n'était pas possible et obligeait à une description différencié de chaque transition.

10.6.2 Transition répliquée

La possibilité de déclaration d'une transition répliquée a été introduit dans PEPS2007. La déclaration d'un transition répliquée permet de définir un ensemble de transition de façon compacte. La réplication de transition est un outil utile dans des modèles avec des schémas des probabilités de routage complexes, comme par exemple, le modèle de *Patron de mobilité aléatoire* présenté dans la Section 4.5.

FIG. 10.5 représente un automate où à partir de chaque état, on peut aller vers l'ensemble d'états répliqués. Dans cet exemple, à partir de chaque état répliqué on va vers tous les états de l'ensemble de réplication.

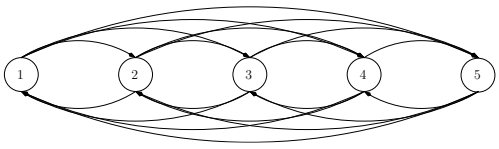


FIG. 10.5 – Réplications d'états et de transitions vers plusieurs états

[2]La syntaxe de ces primitives sont décrites dans la Section B.4.3

La ligne `to (s[N]) e`, dans le code ci-dessous exprime cette réplication. On remarque que, au lieu de placer une fonction pour calculer l'indice de l'état d'arrivée, à l'instar du modèle précèdent, on utilise un identifiant de domaine `N`, précédemment défini.

```
identifiers
  N  = [1..5];                  // domaine de réplication
  mu = 0.5;                     // taux de déclenchement de
                                // l'événement e

events
  syn e  (mu);

reachability = 1;

network aut_rep (continuous)
  aut A
    stt s[N]                    // état répliqué 5 fois [1..5]
      to (s[N]) e               // transition répliquée à tous
                                // les états
```

10.6.3 Condition d'inclusion

Une autre fonctionnalité implantée dans le langage de description de PEPS2007 est la possibilité de définir une condition d'inclusion dans la création d'une transition. Cette condition est une fonction qui prend en compte l'indice de l'automate, de l'état courant et/ou de l'état d'arrivée et permet d'inclure ou d'exclure une transition au moment de la création des transitions

Sur la base du modèle précédent, FIG. 10.6 représente un modèle similaire où les transition sont possibles uniquement si l'état d'arrivée a un indice plus grand que l'état courant.

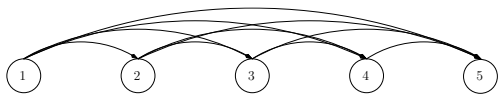


FIG. 10.6 – Réplications d'états et de transitions vers plusieurs états avec condition

Dans le code ci-dessous, la fonction `bigger` modélise cette condition nécessaire pour la création d'une transition. La fonction compare l'indice de l'état courant, obtenu par `sts` et l'indice de l'état d'arrivée, obtenu par `std`.

```
identifiers
  N      = [1..5];              // domaine de réplication
  bigger = (sts < std);         // fonction de condition
```

```
                                  // retourne 1 si l'état de
                                  // départ est plus petit que
                                  // l'état d'arrivée
  mu = 0.5;                       // taux de déclenchement de
                                  // l'événement e

events
  syn e  (mu);

reachability = 1;

network aut_rep (continuous)
  aut A
    stt s[N]                      // état répliqué 5 fois [1..5]
      to (s[N] / bigger) e        // transition répliquée à tous
                                  // les états qui remplissent
                                  // la condition (bigger)
```

On remarque que la condition pour la création d'une transition peut être utilisée pour toutes les descriptions d'une transition, même si cette transition part d'un état non-répliqué, ou si elle même n'est pas elle-même répliquée.

10.7 Réplication d'événements

Des cas similaires à ceux présentés pour les transitions sont étudiés pour les événements. On peut classer la description des événements selon deux types : unique et répliqué.

10.7.1 Événement unique

La déclaration d'un événement unique signifie que un seul événement est associé à une transition à la fois. L'événement associé peut être un événement simple ou répliqué.

Événement simple

Lorsqu'on déclare un événement simple, cet événement va apparaître sur toutes les transitions répliqués où il a été déclaré. FIG. 10.7 représente un automate avec un ensemble d'états répliqués où l'événement e apparaît sur toutes les transitions.

La déclaration de l'événement e dans la dernière ligne du code ci-dessous modélise cette situation. En effet, la réplication de l'événement sur plusieurs transitions se fait par la réplica-

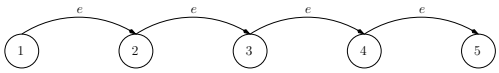


FIG. 10.7 – Réplication d'un événement

tion des états de l'automate, car au moment de la réplication d'un état, cet état est répliqué à l'identique, avec les mêmes transitions et événements, sauf si le contraire est spécifié. Concernant le modèle de Markov, il s'agit d'un événement unique qui peut se déclencher dans des états différents.

```
identifiers
  N  = [1..5];                  // domaine de réplication
  mu = 0.5;                     // taux de déclenchement de
                                // l'événement e

events
  syn e  (mu);

reachability = 1;

network aut_rep (continuous)
  aut A
    stt s[N]                    // état répliqué 5 fois [1..5]
      to (++) e                 // transition vers l'état suivant
```

Événement répliqué

On dit qu'un événement est répliqué lorsqu'on déclare un identifiant d'événement unique, mais indicé selon sa position. D'un point de vue du modèle de Markov, il s'agit d'événements différents.

FIG. 10.8 représente un modèle, où, pour un ensemble d'états répliqués, on utilise un événement différent selon l'état courant.

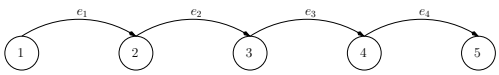


FIG. 10.8 – Réplication d'événements

La fonction `i`, dans le code ci-dessous, calcule l'indice qui est utilisé dans le choix de l'événement qui sera associé à chaque transition.

```
identifiers
  N  = [1..5];          // domaine de réplication d'états
  E  = [1..4];          // domaine de réplication d'événements
  i  = sts;             // indice de l'état de départ
  mu = 0.5              // taux de déclenchement des
                        // événements e[E]

events
  syn e[E]  (mu);       // événement répliqué 4 fois [1..4]

reachability = 1;

network aut_rep (continuous)
  aut A
    stt s[N]            // état répliqué 5 fois [1..5]
      to (++) e[i]      // transition vers l'état suivant
                        // avec l'événement e d'indice i (e[i])
```

Dans un deuxième exemple, on montre que ce même principe d'indexation utilisé pour définir un ensemble d'événements (chacun associé à un état répliqué) peut être utilisé pour indexer les événements lors de la réplication d'automates.

FIG. 10.9 présente un modèle SAN où chaque automate doit se synchroniser avec l'automate suivant. Le dernier automate se synchronise avec le premier, de façon à former un anneau.

Le problème rencontré dans ce modèle est l'utilisation d'un ensemble d'événements différents pour chaque automate. Dans les anciennes versions de PEPS, une description différente de chaque automate était nécessaire.

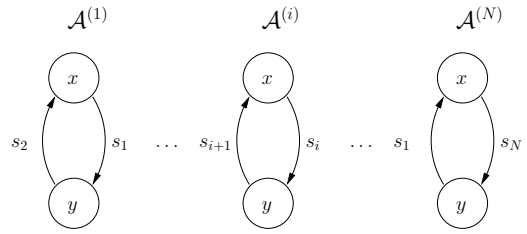


FIG. 10.9 – Réplication d'automates avec des événements synchronisants différents

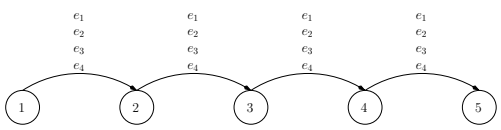
Pour décrire ce modèle de façon compacte, on peut décrire un seul automate et le répliquer. L'utilisation de deux fonctions, `i` et `next`, permet d'associer des événements différents pour chaque automate. La fonction `i` récupère l'indice de l'automate dans une réplication et on l'utilise pour indexer l'événement associé à la transition de x à y. De façon similaire, la fonction `next` calcule l'indice qui est utilisé pour indexer l'événement associé à la transition de y à x, aussi basé sur l'indice de l'automate.

```
identifiers
  N    = [1..10];         // domaine de réplication
  i    = at;              // indice de l'automate courant
  next = (at mod 10) + 1; // indice du prochain automate dans
                          // une file circulaire
  mu   = 0.2;             // taux de déclenchement des
                          // événements s[N]

events
  syn s[N]  (mu);         // événement répliqué 10 fois [1..10]

reachability = 1;

network aut_rep (continuous)
  aut A[N]              // automate répliqué 10 fois [1..10]
    stt x
      to (y) s[i]       // transition de l'état x vers l'état y
                        // avec l'événement s avec l'indice de
                        // l'automate courant (i)
    stt y
      to (x) s[next]    // transition de l'état y vers l'état x
                        // avec l'événement s avec l'indice de
                        // l'automate suivant dans une file
                        // circulaire (next)
```

Comparé à l'ancien langage de description de PEPS2003, où on était obligé de décrire chaque automate individuellement, cette version est beaucoup plus compacte et on peut changer facilement le nombre d'automates.

10.7.2 Ensemble d'événements

Un autre problème rencontré dans les anciens langages était l'association d'un ensemble d'événements répliqués à une transition. FIG. 10.7.2 représente un modèle où un ensemble d'événements répliqués est associé à une transition dans un ensemble d'états répliqués.



Dans l'ancien langage, chaque événement devait être associé individuellement à la transition. Dans le nouveau langage, on a la possibilité d'associer un ensemble d'événements répliqués de façon compacte. La déclaration de l'ensemble d'événements répliqués `e[E]` dans la

dernière ligne, permet d'associer en une seule fois tous les événements répliqués d'un domaine. De façon similaire à la déclaration des transitions répliquées, il suffit de spécifier le domaine de réplication lors de la déclaration de l'événement.

```
identifiers
  N      = [1..5];      // domaine de réplication d'états
  E      = [1..4];      // domaine de réplication d'événements
  mu     = 0.5;         // taux de déclenchement des
                        // événements e[E]

events
  syn e[E]  (mu);       // événement réplique 4 fois [1..4]

reachability = 1;

network aut_rep (continuous)
  aut A
    stt s[N]            // état répliqué 5 fois [1..5]
      to (++) e[E]      // transition vers l'état suivant avec
                        // tous les événements de e[E]
```

10.7.3 Condition d'inclusion

De façon identique aux conditions utilisées pour la création des transitions, la définition d'une condition d'inclusion permet de définir si un événement, ou un ensemble d'événements sera inclus lors de l'association des événements aux transitions.

Dans l'exemple présenté dans FIG. 10.10, des ensembles différents d'événements sont associés à chaque transition selon l'état d'arrivée des transitions. En plus, on veut établir la condition qui est d'avoir uniquement des transitions sortantes de l'état 1. La fonction `is_1` modélise la condition pour la création des transitions, alors que les fonctions `f_e` et `f_l` modélisent les conditions d'inclusion de l'ensemble d'événements `e[E]` et `l[L]`, respectivement.

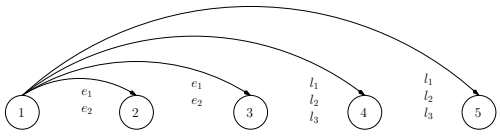


FIG. 10.10 – Réplication d'événements avec condition

```
identifiers
  N    = [1..5];        // domaine de réplication d'états
  E    = [1..2];        // domaine de réplication d'événements
```

```
  L    = [1..3];        // domaine de réplication d'événements
  is_1 = (sts == s[1]); // fonction de condition des transitions
                        // transition sortante uniquement de
                        // l'état s[1]
  f_e  = (std < 4);     // fonction de condition des événements e
                        // état d'arrivée plus petit que 4
  f_l  = (std >= 4);    // fonction de condition des événements l
                        // état d'arrivée plus grand ou égal à 4
  mu   = 0.5            // taux de déclenchement des
                        // événements e[E]
  alpha= 1;             // taux de déclenchement des
                        // événements l[L]

events
  syn e[E]  (mu);       // événement réplique 2 fois [1..2]
  loc l[L]  (alpha);    // événement réplique 3 fois [1..3]

reachability = 1;

network aut_rep (continuous)
  aut A
    stt s[N]            // état répliqué 5 fois [1..5]
      to (s[N] / is_1) e[E]/f_e l[L]/f_l
                        // transitions vers tous les états en
                        // sortant de l'état s[1]
                        // (condition is_1) avec les événements
                        // e[E] pour les états d'arrivée s[2]
                        // et e[3] et avec les événements l[L]
                        // pour les états s[4] et s[5]
```

10.8 Conclusion

Dans ce chapitre on a présenté les potentialités de réplication de PEPS2007 et on les a illustrés par quelques cas de réplications sur des modèles simples. Pour chaque cas on a présenté le code qui décrit de façon compacte le modèle. Pour la plus grande partie de ces modèles, cette description compacte est uniquement possible par l'utilisation des fonctions *at*, *sts* et *std* proposées au début de ce chapitre.

Ces nouvelles fonctions, l'extension des structures du langage telles que la réplication de fonctions, la déclaration de transitions et d'événements répliqués et la définition des conditions d'inclusion, ont permis la description de modèles par des réplications qui, avant, n'était pas faisable pour des grands espaces de réplications. PEPS permet maintenant d'exprimer de tels grands modèles.

Chapitre 11

Conclusion et perspectives

Dans ce chapitre nous dressons un bilan de travaux présentés dans cette thèse ainsi que les perspectives de travaux futurs. Dans la section suivante (Section 11.1), on rappelle les principaux résultats obtenus et les mises en ouvre pour les obtenir. Dans la section 11.2, nous évoquons des perspectives de travaux qui donnent suite aux travaux développés ici.

11.1 Conclusion

L'apport scientifique de cette thèse a été focalisé sur le formalisme de Réseaux d'Automate Stochastiques (SAN- *Stochastic Automata Network*) et s'articule en deux axes :

- méthodes numériques pour l'analyse transitoire sur SAN à temps continu ;
- proposition d'une représentation tensorielle pour le formalisme SAN à temps discret.

11.1.1 Analyse transitoire sur SAN à temps continu

L'efficacité de la représentation tensorielle dans le stockage d'un modèle SAN et de l'algorithme *shuffle* pour la multiplication vecteur-descripteur ont été largement exploitée pour l'obtention de la distribution de probabilité stationnaire du modèle. Cependant, les travaux sur des méthodes pour l'analyse transitoire restent réduits.

Dans cette thèse nous avons exploité des méthodes pour l'analyse transitoire d'un système. Comme on l'a vu, le nombre d'itérations nécessaires pour obtenir la distribution transitoire à l'instant de temps t peut être très grand dépendant des caractéristiques du modèle. Toutefois, ce nombre peut être réduit par des méthodes de détection du régime stationnaire et le coût de calcul peut être réduit considérablement.

Nous avons focalisé nos travaux sur l'efficacité de méthodes de détection du régime station-

naire dans le calcul de disponibilité ponctuelle pour de grands modèles. Pour tester l'efficacité des méthodes de détection du régime stationnaire nous avons comparé différentes méthodes de détection selon deux critères : nombre d'itérations nécessaires à la détection du régime stationnaire et la précision des résultats obtenus. Ces deux critères nous permettent de définir la méthode la plus efficace (le plus petit nombre d'itérations) et qui respect effectivement l'erreur maximum acceptée.

Pour comparer ces différentes méthodes de détection du régime stationnaire, nous avons d'abord proposé une adaptation de l'algorithme *shuffle* pour rendre possible la multiplication d'un descripteur markovien par un vecteur (multiplication à droite).

Sur les mesures obtenues, on a pu constater que la performance de la méthode de *convergence de vecteur avec norme infinie* (CAI) est nettement meilleure que la méthode de *convergence de vecteur avec norme* $L1$ (CAA) et que la méthode de *contrôle de la suite de vecteur* (CSW) pour le critère nombre d'itérations.

Néanmoins, les problèmes de fausse détection du régime stationnaire entraînent des erreurs importantes sur la valeur de la disponibilité ponctuelle et compromet les bons résultats obtenus sur le critère nombre d'itérations. Ces problèmes de fausse détection du régime stationnaire ont été aussi observés pour la méthode CAA, ce qui rend l'utilisation de ces deux méthodes non fiable pour de grands modèles.

Sur la base de modèles testés dans cette thèse, on a pu constater que la méthode CSW permet un calcul fiable et rapide de la disponibilité au régime stationnaire avec un surcoût sur le nombre d'itérations entre 20% et 500% par rapport à la méthode CAI.

Un travail complémentaire relatif au calcul de la disponibilité ponctuelle a été l'utilisation de différentes implantations pour le vecteur constant. Les expérimentations réalisées sur les 3 implantations proposées : *vecteur plein*, *vecteur creux* et *fonction* ont montré que l'utilisation d'un vecteur creux présente un bon compromis entre la vitesse de calcul et l'espace de stockage nécessaire, mais on peut aisément envisager l'implantation d'une heuristique qui choisit automatiquement le meilleur format de représentation selon les caractéristiques du modèle et de l'espace mémoire disponible pour le calcul.

11.1.2 Descripteur pour SAN à temps discret

L'apport le plus important de cette thèse s'articule autour du descripteur discret proposé pour le formalisme SAN à temps discret.

Cet apport se développe en 4 parties :

- la construction de chaînes de transition globales et de l'automate global d'un modèle SAN ;
- la proposition de l'algèbre tensorielle complexe ;
- la définition des matrices d'événements ;

– la formalisation du descripteur discret.

La première partie de cet apport est la définition formelle de règles de construction de chaînes de transitions globales et de l'automate global du modèle. Les chaînes de transitions globales sont des listes ordonnées de tuples de transitions globales (chaînons de transitions) qui ont lieu dans une même unité de temps. La définition formelle de ces chaînes de transitions globales nous a permit de définir précisément le comportement du système et d'obtenir l'automate global du modèle. Ces règles de construction de chaînes de transitions globales masquent à l'utilisateur la complexité de gérer l'occurrence simultanée de plusieurs événements ainsi que la gestion des conflits.

La deuxième partie de cet apport a été la proposition d'une algèbre tensorielle capable d'exprimer les relations de simultanéité, de concurrence et de choix d'événements. Pour ce faire, on a définit trois opérateurs capables d'exprimer les différentes relations entre les événements : *opérateur de simultanéité*, *opérateur de choix* et *opérateur de concurrence*.

L'opérateur de simultanéité a été défini afin d'exprimer l'occurrence simultanée des événements dans un même automate dans une même unité de temps. L'opérateur de concurrence a été défini afin d'exprimer l'occurrence de plusieurs événements dans des automates différents aussi dans une même unité de temps. L'opérateur de choix a été défini afin d'associer plusieurs probabilités de chaînes de transitions (occurrence simultanée des événements) à une même transition.

Ces opérateurs ont été d'abord définis sur un ensemble d'événements, mais son expressivité n'était pas suffisante pour l'expressivité nécessaire pour le formalisme SAN. L'inclusion d'une probabilité de routage a permis une extension de l'algèbre sur des tuples de transitions. Cette extension a permis de représenter les chaînes de transitions d'un modèle SAN par l'algèbre proposée.

Sur la base des règles définies pour la construction de chaînes de transitions globales, on a pu définir des chaînes de transitions locales. Ces chaînes ont été utilisées pour la définition de matrices d'événements qui composent le descripteur discret.

Enfin, le produit tensoriel complexe proposé pour cette algèbre a permis de définir un descripteur discret basé sur le produit tensoriel complexe des matrices d'événements.

L'opérateur de concurrence et le produit tensoriel des matrices d'événements ont été définis de façon à ce que ce produit génère une chaîne de transitions globales. L'égalité de la matrice de transition de l'automate global et du descripteur a été montrée dans la section 8.3 du chapitre 8. Ceci constitue le résultat clé de cette partie.

La définition de cette algèbre et d'un descripteur discret sont des travaux innovants qui ont permis une description très compacte des modèles SAN à temps discret et ouvrent de nombreuses perspectives.

Différentes propriétés, telle que la décomposition en facteurs normaux ont été démontrées pour cette algèbre. Ces propriétés sont présentées par Sales dans [87], ainsi que l'utilisation de

l'algorithme *shuffle* dans la multiplication vecteur-descripteur discret.

11.2 Perspectives

On présente tout d'abord les perspectives de travaux autour de méthodes d'analyse transitoire. Ensuite, on présente les perspectives pour le formalisme SAN à temps discret. Enfin, on présente des perspectives de travaux dans des domaines voisins.

11.2.1 Analyse transitoire

Nous avons proposé dans cette thèse l'utilisation de méthodes d'analyse transitoire pour de grands modèles. Cependant, le développement de méthodes d'analyse transitoire qui explorent la représentation tensorielle des SAN reste encore limitée.

Il semble possible d'envisager des méthodes de calcul du temps moyen d'absorption ou du temps moyen de défaillance qui tirent partie de la représentation tensorielle de SAN. À l'instar des représentations des vecteurs constants proposés dans le chapitre 5 section 5.4, on peut envisager la représentation des états $DOWN$ (*i.e.*, les états où le système ne fournit pas de service) dans le calcul du temps moyen de défaillance par de vecteurs creux ou de fonctions.

Comme on a vu, ces représentations peuvent apporter des gains considérables en espace mémoire sans compromettre le temps de calcul.

11.2.2 SAN à temps discret

Du côté du formalisme SAN à temps discret, les perspectives des travaux dans ce domaine sont assez nombreuses. Dans un premier temps, une comparaison du pouvoir de description du formalisme SAN à temps discret avec des formalismes analogues permettrait de mieux définir l'ensemble des systèmes qui pourront être décrits par ce formalisme.

La nouvelle algèbre tensorielle utilisée pour la représentation tensorielle du modèle ouvre aussi des axes de recherche. Un de ces axes est dans la recherche de nouvelles méthodes de multiplication vecteur-descripteur adaptées aux caractéristiques de cette algèbre, *i.e.*, l'existence d'un seul terme tensoriel contrairement au descripteur markovien en temps continu, qui peut avoir plusieurs termes tensoriels. Il est aussi possible d'envisager des méthodes de simulation qui en profite non seulement de la structure tensorielle du descripteur mais aussi de la puissance de description du formalisme : par exemple, de méthodes de simulation prenant en compte les priorités des événements, ou même des méthodes qui réalisent la simulation directement sur les chaînes de transitions. Des propositions de simulation basée sur de méthodes de simulation parfaite [98] ont déjà été faites pour le formalisme SAN à temps continu [36]. Le même chemin pourra être exploité pour le formalisme SAN à temps discret.

11.2.3 Parallélisation des algorithmes

L'idée de paralléliser les algorithmes de multiplication vecteur-descripteur n'est pas récente. Une première approche parallèle a déjà été proposé dans [4]. Cette approche consiste à paralléliser chaque terme d'un produit tensoriel dans un noeud d'une grappe de calcul.

Toutefois, pour de très grands modèles, le transfert des vecteurs de probabilités à chaque noeud et à chaque itération réduit considérablement l'efficacité de la parallélisation.

Avec les machines à multiprocesseurs et multicoeurs actuelles, nous pouvons envisager d'autres type de parallélisation. On peut par exemple envisager l'utilisation des processeurs du type Cell [80] ou des processeurs graphiques (GPU) [68], où on peut compter le nombre de coeurs par de centaines.

Bibliographie

[1] H. Abdallah and M. Hamza. Sensibilité de mesures transitoires des réseaux d'automates stochastiques : approche parallèle. Technical Report 4190, INRIA, Rennes, France, 2001.

[2] M. Ajmone-Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley and Sons, 1995.

[3] M. Ajmone-Marsan, G. Conte, and G. Balbo. A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems*, 2(2) :93–122, 1984.

[4] L. Baldo, L. G. Fernandes, P. Roisenberg, P. Velho, and T. Webber. Parallel PEPS Tool Performance Analysis using Stochastic Automata Networks. In M. Donelutto, D. Laforenza, and M. Vanneschi, editors, *Euro-Par 2004 International Conference on Parallel Processing*, volume 3149 of *Lecture Notes in Computer Science*, pages 214–219, Pisa, Italy, August/September 2004. Springer-Verlag Heidelberg.

[5] F. Baskett, K. Chandy, R. Muntz, and F. Palacios. Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. *Journal of the ACM*, 22(2) :248–260, 1975.

[6] A. Benoit, L. Brenner, P. Fernandes, and B. Plateau. Aggregation of Stochastic Automata Networks with replicas. *Linear Algebra and its Applications*, 386 :111–136, July 2004.

[7] A. Benoit, L. Brenner, P. Fernandes, B. Plateau, and W. J. Stewart. The PEPS Software Tool. In *Computer Performance Evaluation / TOOLS 2003*, volume 2794 of *LNCS*, pages 98–115, Urbana, IL, USA, 2003. Springer-Verlag Heidelberg.

[8] A. Benoit, B. Plateau, and W. J. Stewart. Memory-efficient Kronecker algorithms with applications to the modelling of parallel systems. In *Proceedings of International Parallel and Distributed Processing Symposium*, pages 275–282, Nice, France, April 2003.

[9] Anne Benoit. *Méthodes et algorithmes pour l'évaluation des performances des systèmes informatiques à grand espace d'états*. PhD thesis, Institut National Polytechnique de Grenoble, Grenoble, France, 2003.

[10] C. Berge. *Graphes et Hypergraphes*. Dunod, Paris, 1970.

[11] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains : Modeling and Performance Evaluation with Computer Science Applications*. John Wiley & Sons, 1998.

[12] B.Plateau and B.Ycart. Fast simulation kernel for urban trafic. Technical Report 41, Rapport de Recherche MAI/IMAG, Grenoble, France, 1997.

[13] L. Brenner, P. Fernandes, B. Plateau, and I. Sbeity. Stochastic automata networks software tool. In *4th International Conference on the Quantitative Evaluation of SysTems (QEST) 2007*, pages 163–164, Edimbourgh, UK, sep 2007.

[14] L. Brenner, P. Fernandes, and A. Sales. The Need for and the Advantages of Generalized Tensor Algebra for Kronecker Structured Representations. In 20^{th} *Annual UK Performance Engineering Workshop*, pages 48–60, Bradford, UK, July 2004.

[15] Peter Buchholz. A class of hierarchical queueing networks and their analysis. *Queueing Systems*, 15 :59–80, 1994.

[16] Peter Buchholz, Joost-Pieter Katoen, Peter Kemper, and Carsten Tepper. Model-checking large structured markov chains. *J. Log. Algebr. Program.*, 56(1-2) :69–97, 2003.

[17] Juan A. Carrasco. Bounding steady-state availability models with group repair and phase type repair distributions. *Performance Evaluation*, 35(3-4) :193–214, 1999.

[18] K.M. Chandy and C.H. Sauer. Approximate methods for analyzing queueing network models of computing systems. *ACM Comput. Surv.*, 10(3) :281–317, 1978.

[19] G. Ciardo. Discrete-time Markovian stochastic Petri nets. In 2^{th} *International Conference on the Numerical Solution of Markov Chains*, pages 339–358, Raleigh, NC, USA, 1995.

[20] G. Ciardo, A. Blakemore, P.F.J. Chimento, J.K. Muppala, and K.S. Trivedi. *Linear Algebra, Markov Chains, and Queueing Models*, chapter Automated Generation and Analysis of Markov Reward Models Using Stochastic Reward Nets, pages 145–191. Springer-Verlag, 1993.

[21] G. Ciardo and A. S. Miner. Storage Alternatives for Large Structured State Spaces. In *9th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, volume 1245 of *LNCS*, pages 44–57, St. Malo, France, 1997. Springer-Verlag Heidelberg.

[22] Ricardo M. Czekster, Paulo Fernandes, Jean-Marc Vincent, and Thais Webber. Split : a flexible and efficient algorithm to vector-descriptor product. In *ValueTools'07 : Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, pages 1–8, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[23] Y. Dallery. Approximate an alysis of general open queueing networks with restricted capacity. Technical report, Grenoble, 1998.

[24] Y. Dallery, Z. Liu, and D. Towsley. Equivalence, reversibility, symmetry and concavity properties in fork-join queuing networks with blocking. *Journal of the ACM*, 41(5) :903–942, 1994.

[25] Edmundo de Souza e Silva and H. Richard Gail. Calculating cumulative operational time distribution of repairable computer systems. *IEEE Transactions on Computers*, 35(4) :322–332, 1986.

[26] Edmundo de Souza e Silva and H. Richard Gail. Calculating availability and performability measures of repairable computer systems using randomization. *Journal of the ACM*, 36(1) :171–193, 1989.

[27] F. Delamare, F. L. Dotti, P. Fernandes, C. M. Nunes, and L. C. Ost. Analytical modeling of random waypoint mobility patterns. In *PE-WASUN '06 : Proceedings of the 3rd ACM international workshop on Performance evaluation of wireless ad hoc, sensor and ubiquitous networks*, pages 106–113, New York, NY, USA, 2006. ACM.

[28] P.J. Denning and J.P. Buzen. The operational analysis of queueing network models. *ACM Comput. Surv.*, 10(3) :225–261, 1978.

[29] J. D. Diener and W. H. Sanders. *Computations with Markov Chains*, chapter Empirical comparison of uniformization methods for continuous-time Markov chains, pages 547–570. Kluwer Academic, 1995.

[30] S. Donatelli. Superposed stochastic automata : a class of stochastic Petri nets with parallel solution and distributed state space. *Performance Evaluation*, 18 :21–36, 1993.

[31] S. Donatelli. Superposed generalized stochastic Petri nets : definition and efficient solution. In R. Valette, editor, *Proceedings of the* 15^{th} *International Conference on Applications and Theory of Petri Nets*, pages 258–277. Springer-Verlag Heidelberg, 1994.

[32] C. Durbach, F. Quessette, and A. Troubnikoff. Solving Large Markov Model based on Stochastic automata networks. In *Advances in Computer and Information Sciences'98*, Belek-Antalya, Turkey, 1998. OIS Press.

[33] W. Feller. *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons, New York, 1966.

[34] P. Fernandes. *Méthodes numériques pour la solution de systèmes Markoviens à grand espace d'états*. PhD thesis, Institut National Polytechnique de Grenoble, Grenoble, France, 1998.

[35] P. Fernandes, B. Plateau, and W. J. Stewart. Efficient descriptor - Vector multiplication in Stochastic Automata Networks. *Journal of the ACM*, 45(3) :381–414, 1998.

[36] P. Fernandes, J-M. Vincent, and T. Webber. Perfect simulation of stochastic automata networks. In Khalid Al-Begain, Armin Heindl, and Miklós Telek, editors, *ASMTA*, volume 5055 of *Lecture Notes in Computer Science*, pages 249–263. Springer, 2008.

[37] G. Florin and S. Natkin. Les réseaux de Petri stochastiques. *Tecniques et Sciences Informatiques*, 4(1) :143–160, 1985.

[38] J-M. Fourneau. Stochastic automata networks : Using structural properties to reduce the state space. In 3^{th} *International Conference on the Numerical Solution of Markov Chains*. Prensas Universitarias de Zaragossa, 2000.

[39] J-M. Fourneau. Discrete time stochastic automata networks : using structural properties and stochastic bounds to simplify the san. In *Proceedings of the 2nd international conference on Performance evaluation methodologies and tools (ValueTools'07)*, pages 1–10, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[40] J-M. Fourneau. Product form steady-state distribution for stochastic automata networks with domino synchronizations. In Nigel Thomas and Carlos Juiz, editors, *5th European Performance Engineering Workshop (EPEW 2008)*, volume 5261 of *Lecture Notes in Computer Science*, pages 110–124. Springer, 2008.

[41] J-M. Fourneau, B. Plateau, and W. Stewart. An algebraic condition for product form in stochastic automata networks without synchronizations. *Performance Evaluation*, 65(11-12) :854–868, 2008.

[42] J.M. Fourneau, B. Plateau, and W. Stewart. Product form for stochastic automata networks. In *ValueTools '07 : Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, pages 1–10, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[43] Bennett L. Fox and Peter W. Glynn. Computing poisson probabilities. *Commun. ACM*, 31(4) :440–445, 1988.

[44] E. Gelenbe. Product form queueing networks with negative and positive customers. *Journal of Applied Probability*, 28 :656–663, 1991.

[45] E. Gelenbe and I. Mitrani. *Analysis and synthesis of computer systems*. Academic Press, London and New York, 1980.

[46] E. Gelenbe and G. Pujolle. *Introduction to queueing networks*. John Wiley & Sons, Inc., New York, NY, USA, 1987.

[47] A. Goyal and A.N. Tantawi. A measure of guaranteed availability and its numerical evaluation. *IEEE Transactions on Computers*, 37(1) :25–32, 1988.

[48] O. Gusak, T. Dayar, and J.-M. Fourneau. Iterative disaggregation for a class of lumpable discrete-time stochastic automata networks. *Performance Evaluation*, 53(1) :43–69, 2003.

[49] G.W.Brahms. *Réseaux de Petri : théorie et pratique*. Paris : Gauthier-Villars, 1983.

[50] S. Haddad. *Une catégorie régulière de réseaux de Petri de haut niveau : définition, propriétés et réduction*. PhD thesis, Université Paris VI, Paris, 1987.

[51] S. Haddad. A reduction theory for colored nets. In *High-Level Petri Nets : Theory and Application*, Springer-Verlag Heidelberg, 1991.

[52] B. Haverkort, H. Hermanns, and J.-P. Katoen. On the use of model checking techniques for dependability evaluation. In *Proc. 19th IEEE Symposium on Reliable Distributed Systems (SRDS'00)*, pages 228–237, Erlangen, Germany, October 2000.

[53] B. R. Haverkort. Approximate performability and dependability analysis using generalized stochastic petri nets. *Performance Evaluation*, 18(1) :61–78, 1993.

[54] J. Hillston. *A compositional approach to performance modelling*. Cambridge University Press, New York, USA, 1996.

[55] J. Hillston and L. Kloul. An Efficient Kronecker Representation for PEPA models. In L. de Alfaro and S. Gilmore, editors, *Proceedings of the first joint PAPM-PROBMIV Workshop)*, pages 120–135, Aachen, Germany, September 2001. Springer-Verlag Heidelberg.

[56] J.E. Hopcroft and J.D. Ullman. *Introduction to automata theory, languages and computation*. Addison-Welsey, 1979.

[57] J. R. Jackson. Networks of waiting lines. *Operations Research*, 5 :518–521, 1957.

[58] J. R. Jackson. Jobshop-like queueing systems. *Management Science*, 10 :131–142, 1963.

[59] K. Jensen. Colored Petri nets. In *High-Level Petri Nets : Theory and Application*, Springer-Verlag Heidelberg, 1991.

[60] S.Stiham Jr and R.Weber. A survey of Markov decision models for control of networks of queues. *Queueing Systems*, 13 :291–314, 1993.

[61] Joost-Pieter Katoen and Ivan S. Zapreev. Safe on-the-fly steady-state detection for time-bounded reachability. In *QEST '06 : Proceedings of the 3rd international conference on the Quantitative Evaluation of Systems*, pages 301–310, Washington, DC, USA, 2006. IEEE Computer Society.

[62] L. Kleinrock. *Queueing Systems*. John Wiley & Sons, New York, 1975.

[63] J. D. C. Little. A proof of the queueing formula $L = \lambda W$. *Operating Research*, 9 :383–387, 1961.

[64] B. Meyer. *Eiffel : The Language*. Prentice Hall, Englewood Cliffs, NJ, USA, 1992.

[65] M. K. Molloy. Discrete time stochastic petri nets. *IEEE Trans. Softw. Eng.*, 11(4) :417–423, 1985.

[66] R. R. Muntz, E. de Souza e Silva, and A. Goyal. Bounding availability of repairable systems. *IEEE Trans. Comput.*, 38(12) :1714–1723, 1989.

[67] N.Götz, H.Hermanns, U.Herzog, V.Mertsiotakis, and M.Rettelbach. Constructive specification techniques - integrating functional performance and dependability aspects. In *Quantitative Methods in Parallel Systems*. Springer Verlag, 1995.

[68] Nvidia. GPU Computing Solutions. http ://www.nvidia.com/object/GPU_Computing.html.

[69] PEPS. Performance Evaluation of Parallel Systems. http ://www-id.imag.fr/Logiciels/peps/.

[70] P.J.Courtois. *Decomposability : queueing and computer systems applications*. London : Academic Press, 1977.

[71] P.Lascaux and R.Théodor. *Analyse numérique matricielle appliquée à l'art de l'ingénieur (vol. 1 & 2)*. Masson, Paris, 1994.

[72] B. Plateau. *De l'Evaluation du Parallélisme et de la Synchronisation*. PhD thesis, Paris-Sud, Orsay, France, 1984.

[73] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *Proceedings of the 1985 ACM SIGMETRICS conference on Measurements and Modeling of Computer Systems*, pages 147–154, Austin, Texas, USA, 1985. ACM Press.

[74] B. Plateau and K. Atif. Stochastic Automata Networks for modelling parallel systems. *IEEE Transactions on Software Engineering*, 17(10) :1093–1108, 1991.

[75] B. Plateau and J-M. Fourneau. A methodology for solving markov models of parallel systems. *J. Parallel Distrib. Comput.*, 12(4) :370–387, 1991.

[76] B. Plateau, J.M. Fourneau, and K.H. Lee. PEPS : A package for solving complex Markv model of parallel systems. In R. Puigjaner and D. Potier, editors, *Modeling Techniques and Tools for Computer Performance Evaluation*. Spain, 1988.

[77] B. Plateau and W. J. Stewart. Stochastic Automata Networks : product forms and iterative solutions. Technical Report 2939, Grenoble, 1996. ftp ://ftp.inria.fr/INRIA/publication/RR/RR-2939.ps.gz.

[78] M. Reiser and S. S. Lavenberg. Mean-value analysis of closed multichain queueing networks. *Journal of the ACM*, 27(2) :313–322, 1980.

[79] W. Reisig. *Petri nets : an introduction*. Springer-Verlag Heidelberg, 1985.

[80] IBM Research. The Cell Architecture. http ://www.research.ibm.com/cell/.

[81] G. Richter. Clocks and their use for time modelling. In *Information Systems : Theoretical and Formal Aspects*, pages 49–66, North Holland, Amsterdam, 1985.

[82] O.H. Roux, D. Delfieu, and P. Molinaro. Discrete time approach of time petri nets for real-time systems analysis. *8th IEEE International Conference on Emerging Technologies and Factory Automation*, 2 :197–204, 2001.

[83] G. Rubino and B. Sericola. Interval availability distribution computation. In *23th IEEE International Symposium on Fault Tolerant Computing*, pages 48–55, Toulouse, France, 1993.

[84] Gerardo Rubino and Bruno Sericola. Interval availability analysis using operational periods. *Performance Evaluation*, 14(3-4) :257–272, 1992.

[85] Gerardo Rubino and Bruno Sericola. Interval availability analysis using denumerable markov processes : Application to multiprocessor subject to breakdowns and repair. *IEEE Transaction on Computers*, 44(2) :286–291, 1995.

[86] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Boston : PWS Publishing Company, 1995.

[87] A. Sales. *Réseaux d'Automates Stochastiques : Génération de l'espace d'états atteignables et Multiplication vecteur-descripteur pour une sémantique en temps discret*. PhD thesis, Institut National Polytechnique de Grenoble, Grenoble, France, 2009.

[88] A. Sales and B. Plateau. Reachable state space generation for structured models which use functional transitions. In *QEST '09 : Proceedings of the The Quantitative Evaluation of Systems*, pages 1–10, Budapest, Hungary, 2009. IEEE Computer Society.

[89] I. Sbeity. *Évaluation de Performance et Conception de Logiciel*. PhD thesis, Institut National Polytechnique de Grenoble, Grenoble, France, 2006.

[90] I. Sbeity and B. Plateau. Impact de l'irréductibilité dans le calcul des bornes stochastiques : illustration avec un modèle en grappe. In *6eme conférence Francophone de Modélisation et Simulation (MOSIM06)*, Rabat, Maroc, 2006.

[91] Bruno Sericola. Availability analysis of repairable computer systems and stationarity detection. *IEEE Transactions on Computers*, 48(11) :1166–1172, 1999.

[92] W. J. Stewart. *Introduction to the numerical solution of Markov chains*. Princeton University Press, 1994.

[93] William J. Stewart. MARCA : MARkov Chain Analyzer. http ://www4.ncsu.edu/ billy/MARCA/marca.html.

[94] W.J. Stewart, K. Atif, and B. Plateau. The numerical solution of stochastic automata networks. *European Journal of Operational Research*, 86 :503–525, 1995.

[95] W.J. Stewart and B. Plateau. Stochastic Automata Networks for Dependability Modelling. In *Aerospace Conference Proceedings*, volume 4, pages 435–448, USA, 2000. IEEE.

[96] T.G.Robertazzi. *Computer networks and systems : queueing theory and performance evaluation*. New York : Springer-Verlag, 1990.

[97] K. S. Trivedi. *Probability & statistics with reliability, queuing, and computer science applications*. Englewood Cliffs : Prentice-Hall, 1982.

[98] Jean-Marc Vincent. Perfect simulation of monotone systems for rare event probability estimation. In *Winter Simulation Conference*, pages 528–537. ACM, 2005.

[99] V.S.Borkar. Control of Markov chains with long-run average cost criterion : the dynamic programming equations. *SIAM Journal of Control Optimization*, 27 :642–657, 1989.

[100] B. Ycart. Simulation de modèles markoviens. Master's thesis, Université Joseph Fourier, Grenoble, 1997.

[101] Hakan L. S. Younes, Marta Kwiatkowska, Gethin Norman, and David Parker. Numerical vs. statistical probabilistic model checking. *Int. J. Softw. Tools Technol. Transf.*, 8(3) :216–228, 2006.

[102] Armin Zimmermann, Jörn Freiheit, and Günter Hommel. Discrete time stochastic petri nets for the modeling and evaluation of real-time systems. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS'01)*, pages 1069–1074, Washington, DC, USA, 2001. IEEE Computer Society.

Annexe A

Ensemble de mesures relevées pour les méthodes de disponibilité ponctuelle

On présente dans cette annexe l'ensemble des mesures relevées sur les expérimentations réalisées afin de tester l'efficacité des méthodes de détection du régime stationnaire.

Dans l'ensemble de graphes présentés dans cette annexe, on emploie les mêmes notations que dans le chapitre 3.

✍ **Rappelons**

UC	Nombre d'itérations calculé pour la méthode d'uniformisation classique par la méthode de troncature de Fox-Glynn (Chapitre 3, Section 3.2.1) ;
CRI	Méthode de convergence de vecteur (Algorithme 3.9) avec les tests de convergence du vecteur selon la convergence relative du vecteur (norme infinie relative) pour la multiplication à gauche ;
CAI	Méthode de convergence de vecteur (Algorithme 3.9) avec les tests de convergence du vecteur selon la convergence absolue du vecteur (norme infinie) pour la multiplication à droite ;
CAA	Méthode de convergence de vecteur (Algorithme 3.9) avec les tests de convergence du vecteur selon la convergence absolue accumulée du vecteur (norme L1) pour la multiplication à droite ;
CSW	Méthode de contrôle de la suite de w_n (Algorithme 3.10).

On remarque que les courbes qui représentent les méthodes UC et CRI ne sont pas incluses dans l'ensemble des graphes pour des raisons de clarté. Ces deux courbes ne sont pas notre sujet principal de comparaison. Cependant, les mesures relevées pour ces méthodes sont présentées sur les tableaux associés à chaque graphe à titre d'information.

A.1 FAS- Mesures relevées

On utilise le modèle FAS (Chapitre 4, Section 4.1) pour tester l'efficacité des méthodes de détection du régime stationnaire lorsque la taille du modèle augmente. Pour rendre compte de l'efficacité des méthodes sur ce modèle, 4 configurations ont été testées. Chaque configuration utilise un nombre différent de serveurs.

La première configuration utilise 5 serveurs alors que les autres configurations utilisent 10, 15 et 20 serveurs. Les espaces d'états de ces modèles sont, respectivement, égaux à 32, $1\ 024$, $32\ 768$ et $1\ 048\ 576$ états.

L'état initial de ces modèles a été défini avec tous les serveurs inactifs, c'est-à-dire, tous les automates sont dans l'état I. Cet état initial est défini par la fonction fas_{init} ci-dessous :

$$fas_{init} = (nb\ [\mathcal{S}^{(1)}\ ..\ \mathcal{S}^{(N)}]\ I)\ ==\ N$$

On considère qu'un état appartient à l'ensemble d'état UP si il y a, au moins, un serveur inactif (état I). L'ensemble d'états UP est défini par la fonction fas_{UP} :

$$fas_{UP} = (nb\ [\mathcal{S}^{(1)}\ ..\ \mathcal{S}^{(N)}]\ I)\ >\ 0$$

Pour tester ces modèles, on a choisi un taux d'arrivée de clients (λ) égal à $0,5$ par minute et le taux de service (μ) égal à $0,1$ par minute. Les valeurs de λ et μ utilisées dans nos tests sont les mêmes que les valeurs adoptées dans [14].

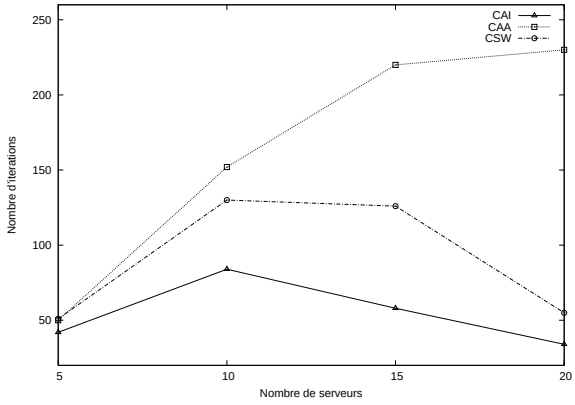
L'instant de temps t choisi pour ces tests est de $1\ 000$ minutes.

La section A.1.1 présente les graphes représentant le nombre d'itérations relevées des modèles testés alors que la section A.1.2 présente les graphes sur la précision des résultats obtenus pour la disponibilité ponctuelle à l'instant de temps choisi.

A.1.1 Nombre d'itérations

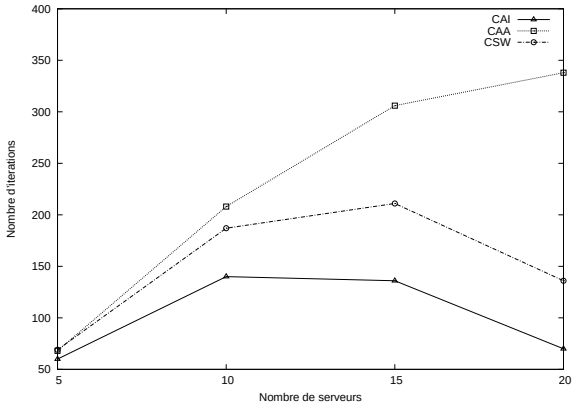
FIG. A.1 présente les mesures relevées organisées selon différents nombres de serveurs, alors que FIG. A.2 présente les mesures organisées selon l'erreur maximum acceptée.

Tolérance $1e^{-5}$



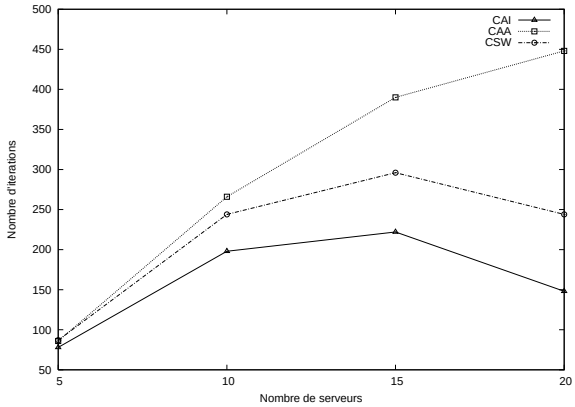
Nombre de serveurs	5	10	15	20
UC	1031	1562	2089	2612
CRI	46	120	136	166
CAI	42	84	58	34
CAA	50	152	220	230
CSW	51	130	126	55

Tolérance $1e^{-7}$



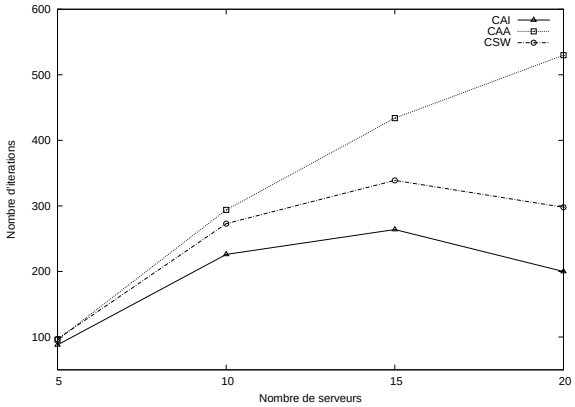
Nombre de serveurs	5	10	15	20
UC	1060	1599	2131	2659
CRI	66	176	220	274
CAI	60	140	136	70
CAA	68	208	306	338
CSW	69	187	211	136

Tolérance $1e^{-9}$



Nombre de serveurs	5	10	15	20
UC	1086	1630	2167	2700
CRI	84	234	306	384
CAI	78	198	222	148
CAA	86	266	390	448
CSW	87	244	296	244

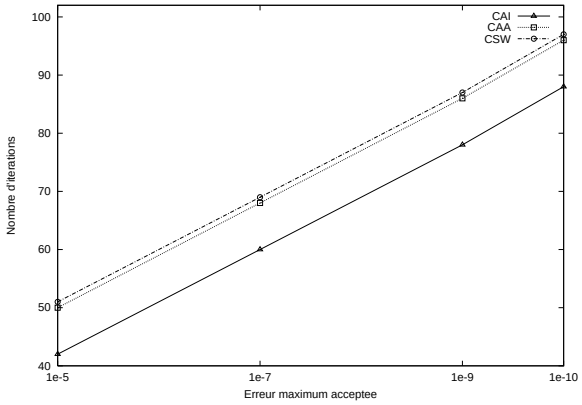
Tolérance $1e^{-10}$



Nombre de serveurs	5	10	15	20
UC	1097	1645	2184	2718
CRI	92	262	348	438
CAI	88	226	264	200
CAA	96	294	434	530
CSW	97	273	339	298

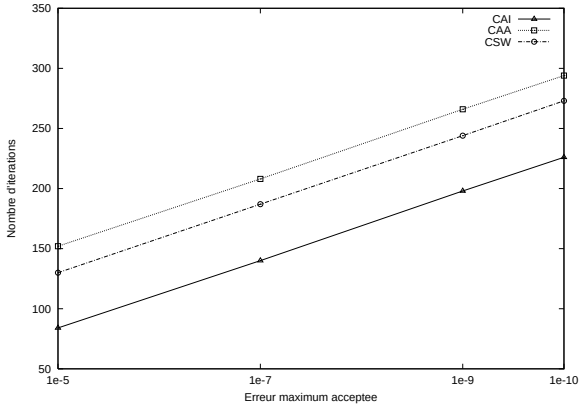
FIG. A.1 – FAS - Nombre d'itérations pour différents nombres de serveurs

5 serveurs



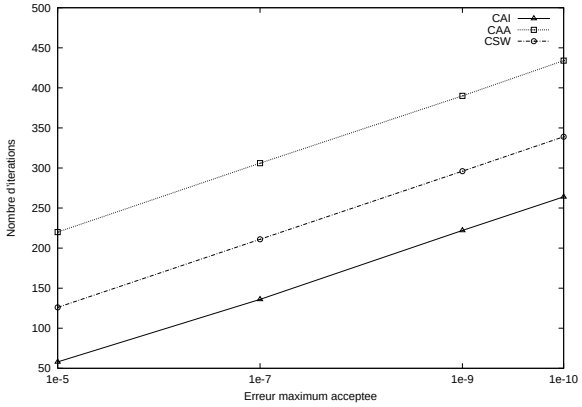
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	1031	1060	1086	1097
CRI	46	66	84	92
CAI	42	60	78	88
CAA	50	68	86	96
CSW	51	69	87	97

10 serveurs



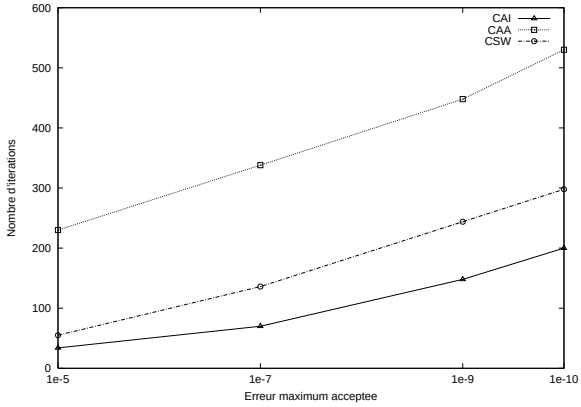
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	1562	1599	1630	1645
CRI	120	176	234	262
CAI	84	140	198	226
CAA	152	208	266	294
CSW	130	187	244	273

15 serveurs



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	2089	2131	2167	2184
CRI	136	220	306	348
CAI	58	136	222	264
CAA	220	306	390	434
CSW	126	211	296	339

20 serveurs



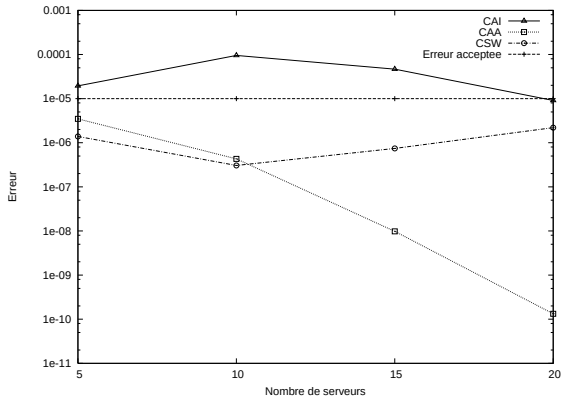
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	2612	2659	2700	2718
CRI	166	274	384	438
CAI	84	140	198	226
CAA	230	338	448	530
CSW	55	136	244	298

FIG. A.2 – FAS - Nombre d'itérations pour différentes erreurs maximum acceptées

A.1.2 Précision des résultats

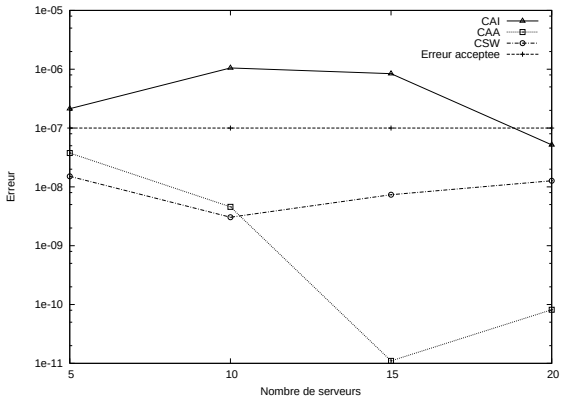
FIG. A.3 présente les mesures relevées organisées selon différents nombres de serveurs, alors que FIG. A.4 présente les mêmes mesures organisées selon l'erreur maximum acceptée.

Tolérance $1e^{-5}$



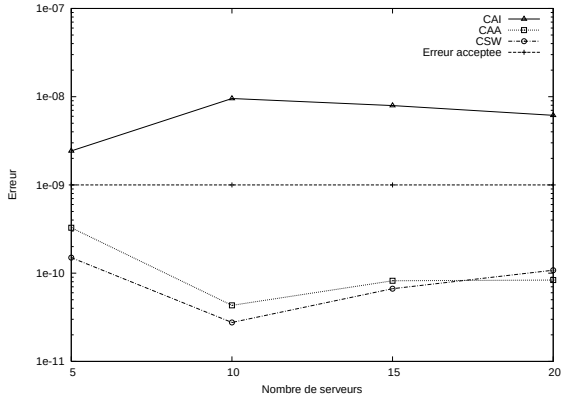
Nombre de serveurs	5	10	15	20
CAI	$1.95e-05$	$9.53e-05$	$4.66e-05$	$9.11e-06$
CAA	$3.48e-06$	$4.31e-07$	$9.82e-09$	$1.33e-10$
CSW	$1.39e-06$	$3.06e-07$	$7.43e-07$	$2.20e-06$

Tolérance $1e^{-7}$



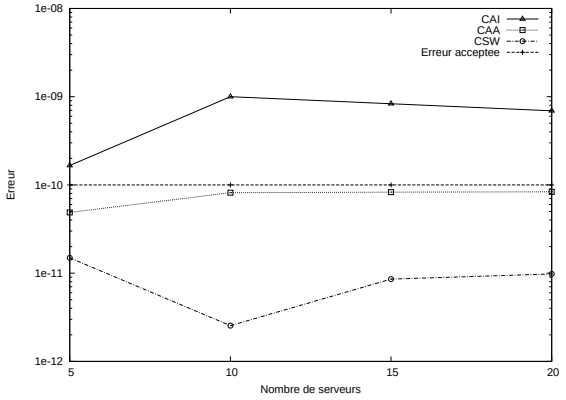
Nombre de serveurs	5	10	15	20
CAI	$2.13e-07$	$1.05e-06$	$8.41e-07$	$5.18e-08$
CAA	$3.76e-08$	$4.58e-09$	$1.10e-11$	$8.14e-11$
CSW	$1.51e-08$	$3.05e-09$	$7.36e-09$	$1.27e-08$

Tolérance $1e^{-9}$



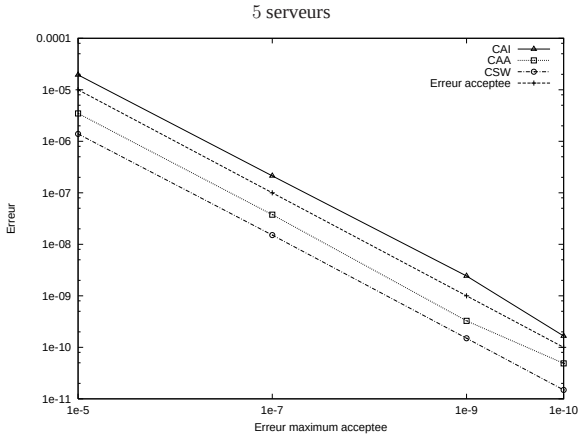
Nombre de serveurs	5	10	15	20
CAI	$2.43e-09$	$9.53e-09$	$7.93e-09$	$6.15e-09$
CAA	$3.27e-10$	$4.32e-11$	$8.20e-11$	$8.35e-11$
CSW	$1.50e-10$	$2.76e-11$	$6.66e-11$	$1.08e-10$

Tolérance $1e^{-10}$

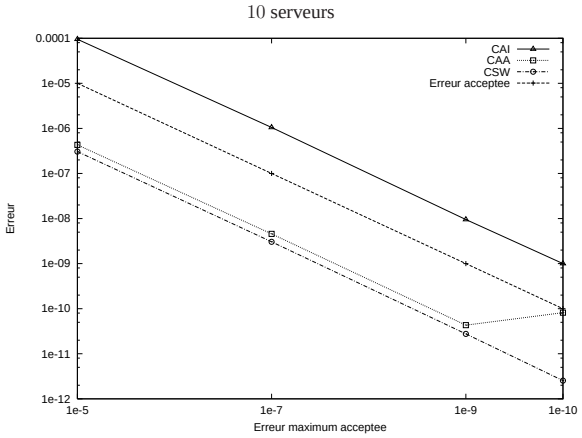


Nombre de serveurs	5	10	15	20
CAI	$1.66e-10$	$1.00e-09$	$8.31e-10$	$6.92e-10$
CAA	$4.88e-11$	$8.17e-11$	$8.29e-11$	$8.35e-11$
CSW	$1.49e-11$	$2.54e-12$	$8.56e-12$	$9.80e-12$

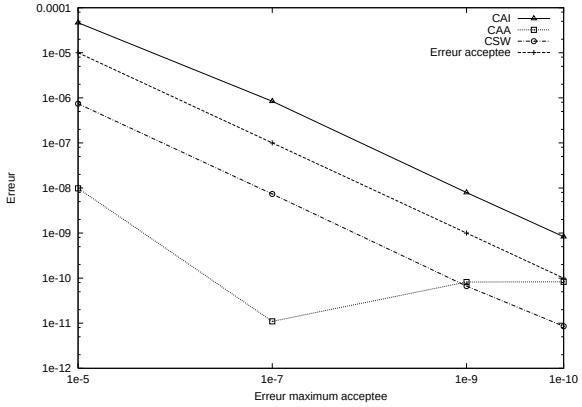
FIG. A.3 – FAS - Précision des résultats pour différents nombres de serveurs



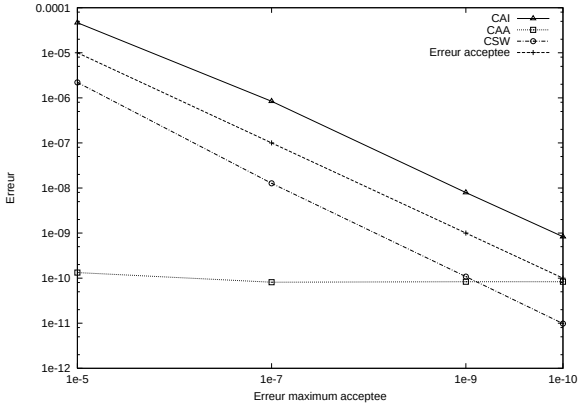
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$1.95e-05$	$2.13e-07$	$2.43e-09$	$1.66e-10$
CAA	$3.48e-06$	$3.76e-08$	$3.27e-10$	$4.88e-11$
CSW	$1.39e-06$	$1.51e-08$	$1.50e-10$	$1.49e-11$



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$9.53e-05$	$1.05e-06$	$9.53e-09$	$1.00e-09$
CAA	$4.31e-07$	$4.58e-09$	$4.32e-11$	$8.17e-11$
CSW	$3.06e-07$	$3.05e-09$	$2.76e-11$	$2.54e-12$



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$4.66e-05$	$8.41e-07$	$7.93e-09$	$8.31e-10$
CAA	$9.82e-09$	$1.10e-11$	$8.20e-11$	$8.29e-11$
CSW	$7.43e-07$	$7.36e-09$	$6.66e-11$	$8.56e-12$



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$9.11e-06$	$5.18e-08$	$6.15e-09$	$6.92e-10$
CAA	$1.33e-10$	$8.14e-11$	$8.35e-11$	$8.35e-11$
CSW	$2.20e-06$	$1.27e-08$	$1.08e-10$	$9.80e-12$

FIG. A.4 – FAS - Précision des résultats pour différentes erreurs maximum acceptées

A.2 MSA- Mesures relevées

On a adopté 5 configurations différentes pour tester le modèle MSA (Chapitre 4, Section 4.2). Ce modèle a été testé avec 16, 32, 64, 128 et 256 processeurs. Pour ces 5 configurations, on a, respectivement, des espaces d'états égaux à 34, 66, 130, 258 et 514 états.

Dans l'état initial du modèle, touts les processeurs sont disponibles et la mémoire tampon est en état de marche. La fonction msa_{init} définit cet état initial :

$$msa_{init} = (st\ \mathcal{P}\ ==\ N)\ \&\&\ (st\ \mathcal{B}\ ==\ A)$$

L'ensemble d'états UP contient tous les états où il y a, au moins, un processeur disponible et la mémoire tampon est en état de marche. Cet ensemble est défini par la fonction msa_{UP} :

$$msa_{UP} = (st\ \mathcal{P}\ !=\ 0)\ \&\&\ (st\ \mathcal{B}\ ==\ A)$$

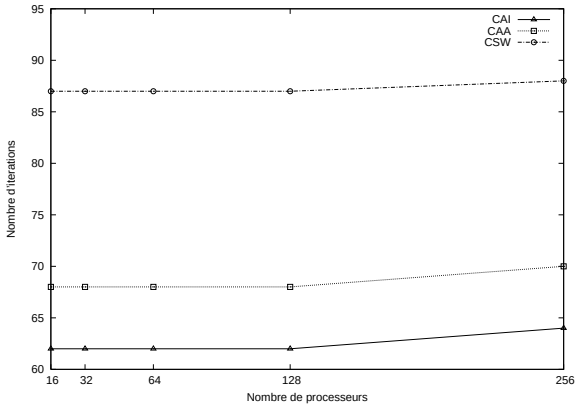
Les valeurs de taux de transitions adoptées dans nos tests sont conformes aux valeurs proposées dans [91]. De cette manière, on a le taux de défaillance $\lambda = 0,01$ par semaine et une réparation des processeurs de $\mu = 0,1666$ par heure. Le taux de défaillance individuel de la mémoire tampon est $\gamma = 0,22$ par semaine et le taux de réparation $\tau = 0,1666$ par heure. Pour nos tests, on fixe le nombre de niveaux de la mémoire tampon à $1\ 024$. On multiplie le taux de défaillance individuel de la mémoire tampon par le nombre de niveaux de la mémoire tampon pour avoir le taux réel de défaillance de la mémoire tampon.

On calcule la disponibilité ponctuelle pour un instant de temps t égal à $1\ 000$ heures.

A.2.1 Nombre d'itérations

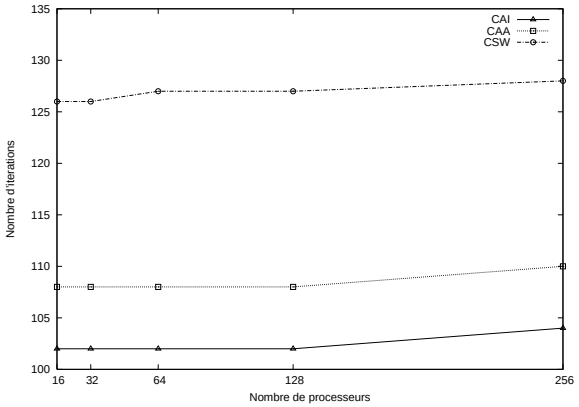
FIG. A.5 présente les mesures relevées organisées selon différents nombres de processeurs, alors que FIG. A.6 présente les mesures organisées selon l'erreur maximum acceptée.

Tolérance $1e^{-5}$



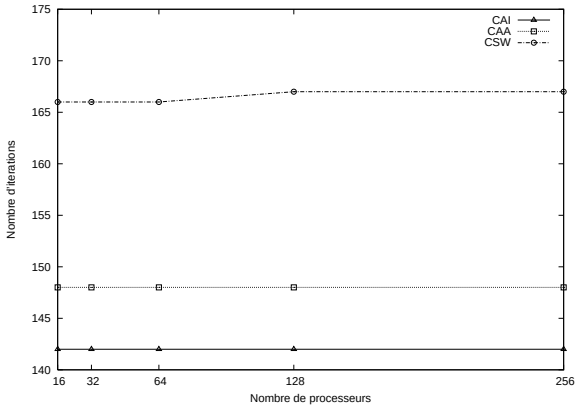
Nombre de processeurs	16	32	64	128	256
UC	1677	1678	1680	1684	1692
CRI	4	6	6	6	6
CAI	62	62	62	62	64
CAA	68	68	68	68	70
CSW	87	87	87	87	88

Tolérance $1e^{-7}$



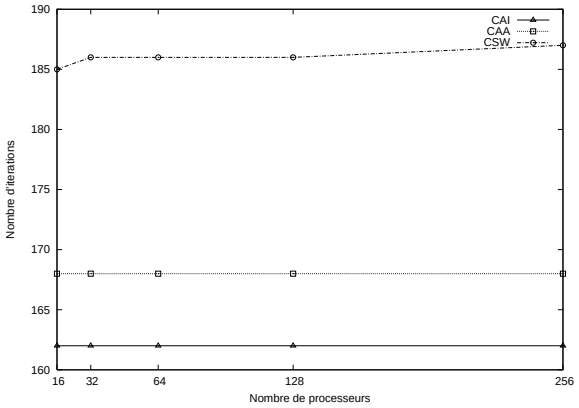
Nombre de processeurs	16	32	64	128	256
UC	1716	1716	1718	1722	1730
CRI	8	8	30	46	62
CAI	102	102	102	102	104
CAA	108	108	108	108	110
CSW	126	126	127	127	128

Tolérance $1e^{-9}$



Nombre de processeurs	16	32	64	128	256
UC	1748	1748	1750	1754	1763
CRI	50	64	78	94	110
CAI	142	142	142	142	142
CAA	148	148	148	148	148
CSW	166	166	166	167	167

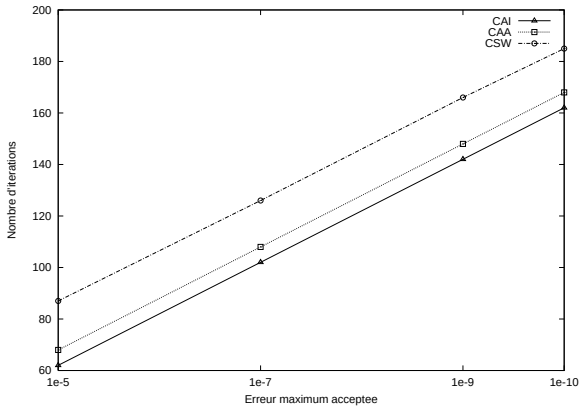
Tolérance $1e^{-10}$



Nombre de processeurs	16	32	64	128	256
UC	1762	1763	1765	1769	1777
CRI	72	86	100	116	134
CAI	162	162	162	162	162
CAA	168	168	168	168	168
CSW	185	186	186	186	187

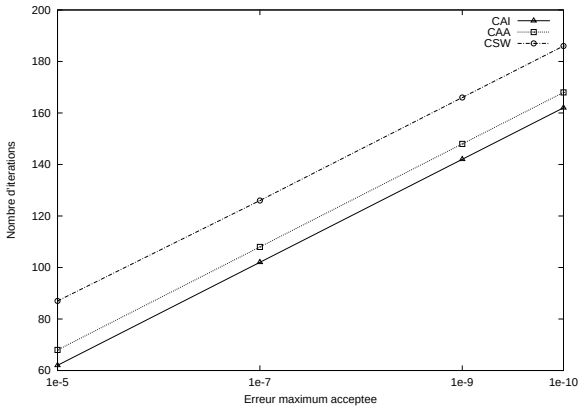
FIG. A.5 – MSA - Nombre d'itérations pour différents nombres de processeurs

16 processeurs



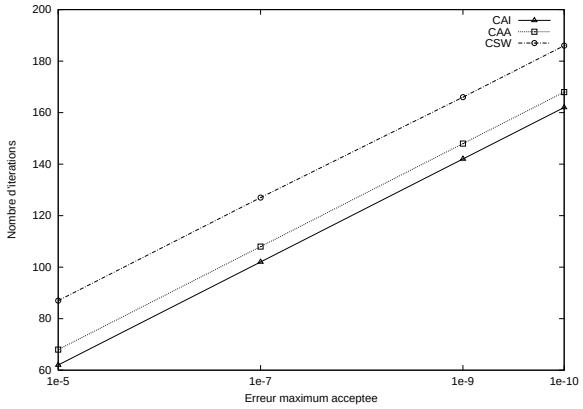
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	1677	1716	1748	1762
CRI	4	8	50	72
CAI	62	102	142	162
CAA	68	108	148	168
CSW	87	126	166	185

32 processeurs



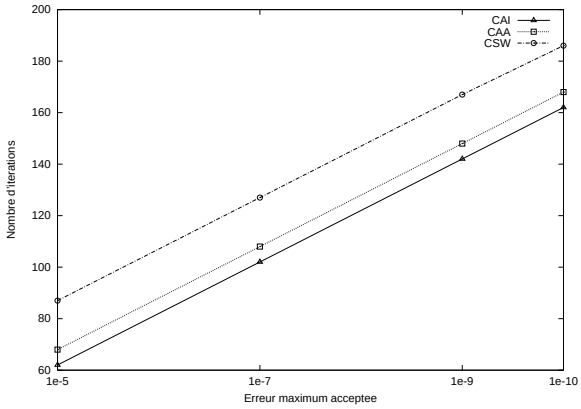
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	1678	1716	1748	1763
CRI	6	8	64	86
CAI	62	102	142	162
CAA	68	108	148	168
CSW	87	126	166	186

64 processeurs



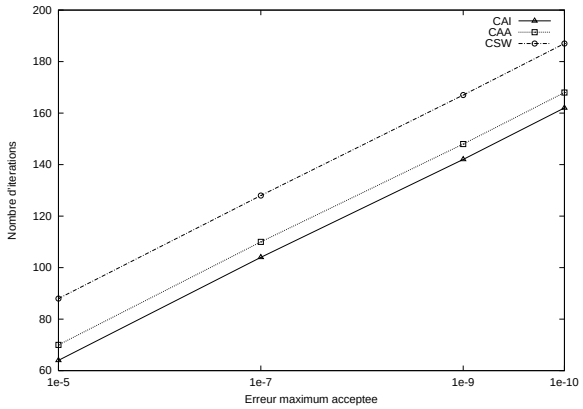
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	1680	1718	1750	1765
CRI	6	30	78	100
CAI	62	102	142	162
CAA	68	108	148	168
CSW	87	127	166	186

128 processeurs



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	1684	1722	1754	1769
CRI	6	46	94	116
CAI	62	102	142	162
CAA	68	108	148	168
CSW	87	127	167	186

256 processeurs

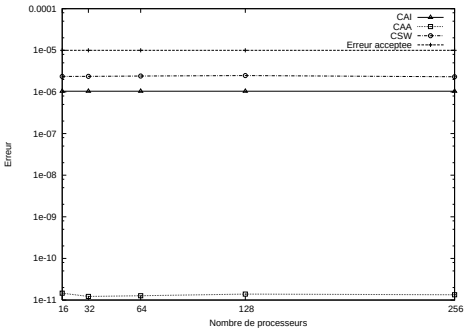


Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	1692	1730	1763	1777
CRI	6	62	110	134
CAI	64	104	142	162
CAA	70	110	148	168
CSW	88	128	167	187

A.2.2 Précision des résultats

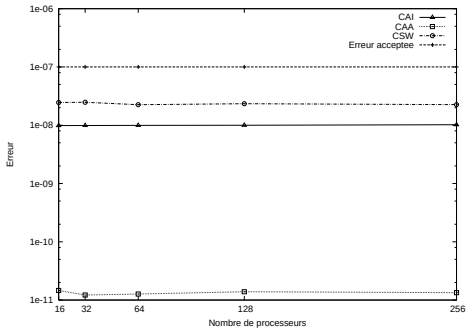
FIG. A.7 présente les mesures relevées organisées selon différents nombres de processeurs, alors que FIG. A.8 présente les mesures organisées selon l'erreur maximum acceptée.

Tolérance $1e^{-5}$



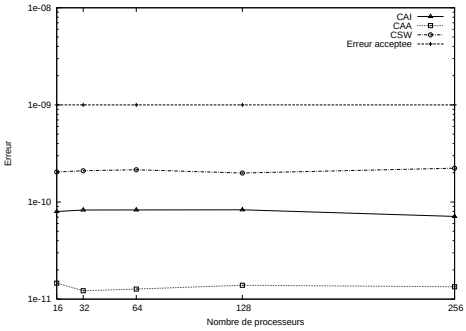
Nombre de processeurs	16	32	64	128	256
CAI	$1.04e-06$	$1.04e-06$	$1.04e-06$	$1.04e-06$	$1.04e-06$
CAA	$1.46e-11$	$1.22e-11$	$1.27e-11$	$1.39e-11$	$1.34e-11$
CSW	$2.36e-06$	$2.37e-06$	$2.41e-06$	$2.47e-06$	$2.32e-06$

Tolérance $1e^{-7}$



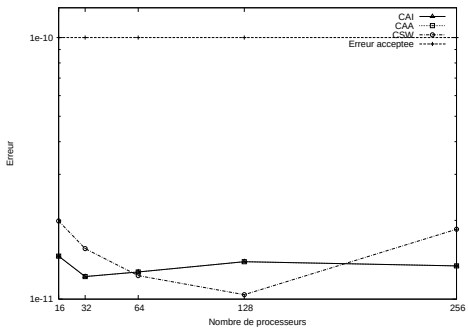
Nombre de processeurs	16	32	64	128	256
CAI	$9.87e-09$	$9.89e-09$	$9.94e-09$	$1.00e-08$	$1.02e-08$
CAA	$1.46e-11$	$1.22e-11$	$1.27e-11$	$1.39e-11$	$1.34e-11$
CSW	$2.46e-08$	$2.48e-08$	$2.25e-08$	$2.34e-08$	$2.26e-08$

Tolérance $1e^{-9}$



Nombre de processeurs	16	32	64	128	256
CAI	$7.99e-11$	$8.27e-11$	$8.29e-11$	$8.32e-11$	$7.11e-11$
CAA	$1.46e-11$	$1.22e-11$	$1.27e-11$	$1.39e-11$	$1.34e-11$
CSW	$2.04e-10$	$2.10e-10$	$2.15e-10$	$1.99e-10$	$2.23e-10$

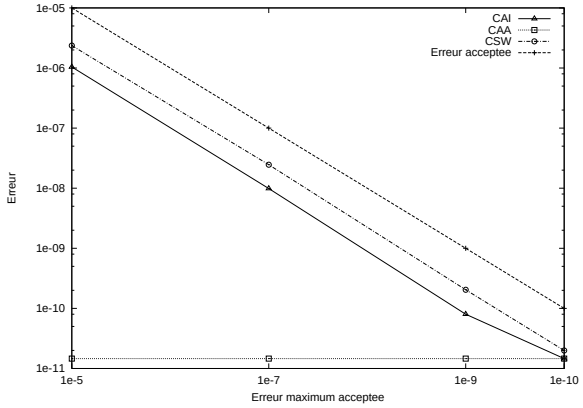
Tolérance $1e^{-10}$



Nombre de processeurs	16	32	64	128	256
CAI	$1.46e-11$	$1.22e-11$	$1.27e-11$	$1.39e-11$	$1.34e-11$
CAA	$1.46e-11$	$1.22e-11$	$1.27e-11$	$1.39e-11$	$1.34e-11$
CSW	$1.99e-11$	$1.56e-11$	$1.23e-11$	$1.04e-11$	$1.85e-11$

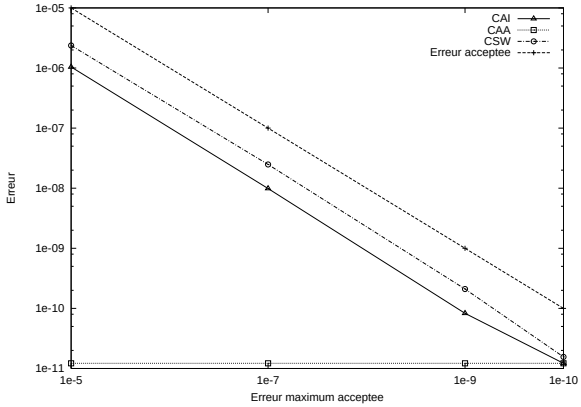
FIG. A.7 – MSA - Précision des résultats pour différents nombres de processeurs

16 processeurs



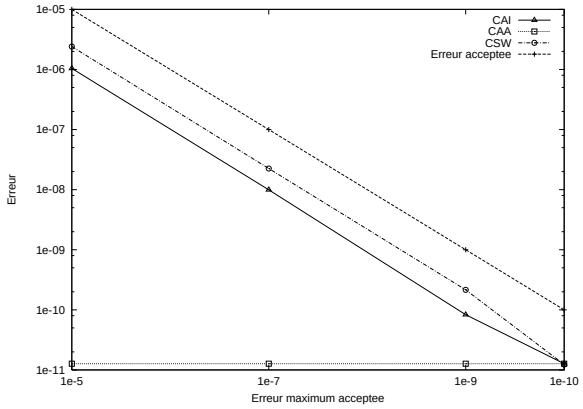
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$1.04e-06$	$9.87e-09$	$7.99e-11$	$1.46e-11$
CAA	$1.46e-11$	$1.46e-11$	$1.46e-11$	$1.46e-11$
CSW	$2.36e-06$	$2.46e-08$	$2.04e-10$	$1.99e-11$

32 processeurs



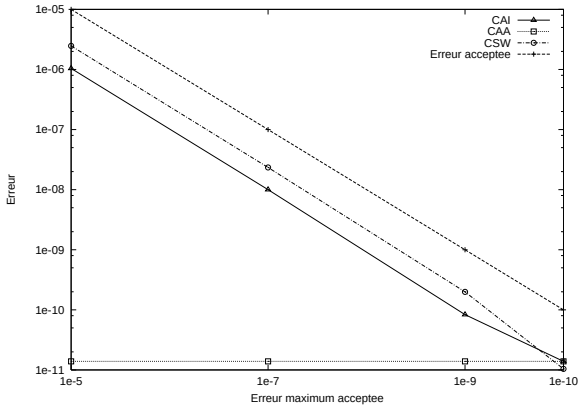
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$1.04e-06$	$9.89e-09$	$8.27e-11$	$1.22e-11$
CAA	$1.22e-11$	$1.22e-11$	$1.22e-11$	$1.22e-11$
CSW	$2.37e-06$	$2.48e-08$	$2.10e-10$	$1.56e-11$

64 processeurs



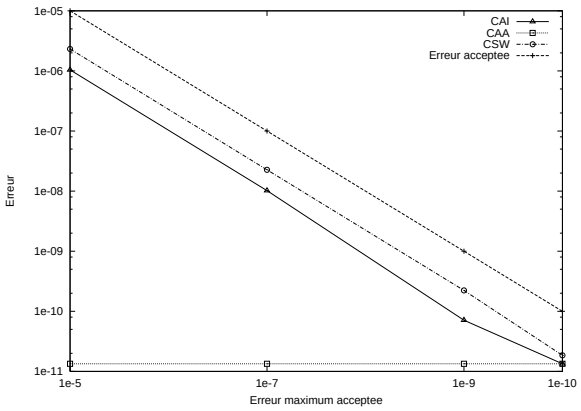
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$1.04e-06$	$9.94e-09$	$8.29e-11$	$1.27e-11$
CAA	$1.27e-11$	$1.27e-11$	$1.27e-11$	$1.27e-11$
CSW	$2.41e-06$	$2.25e-08$	$2.15e-10$	$1.23e-11$

128 processeurs



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$1.04e-06$	$1.00e-08$	$8.32e-11$	$1.39e-11$
CAA	$1.39e-11$	$1.39e-11$	$1.39e-11$	$1.39e-11$
CSW	$2.47e-06$	$2.34e-08$	$1.99e-10$	$1.04e-11$

256 processeurs



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$1.04e-06$	$1.02e-08$	$7.11e-11$	$1.34e-11$
CAA	$1.34e-11$	$1.34e-11$	$1.34e-11$	$1.34e-11$
CSW	$2.32e-06$	$2.26e-08$	$2.23e-10$	$1.85e-11$

FIG. A.8 – MSA - Précision des résultats pour différentes erreurs maximum acceptées

A.3 CFR- Mesures relevées

Comme on l'a décrit auparavant, le modèle CFR (Chapitre 4, Section 4.3, page 58) modélise un ensemble de noeuds configurés en anneau. Ce modèle a été testé avec 5, 10 et 15 noeuds. Le modèle avec 5 noeuds a un espace d'états égal à 243 états alors que les modèles avec 10 et 15 noeuds ont, respectivement, des espaces d'états égaux à $50\ 949$ et $14\ 348\ 907$ états.

Dans l'état initial de ces modèles, tous les noeuds sont actifs, c'est-à-dire, tous les automates sont dans l'état A. Cet état est défini par la fonction cfr_{init} ci-dessous :

$$cfr_{init} = (nb\ [\mathcal{N}^{(1)}\ ..\ \mathcal{N}^{(N)}]\ A)\ ==\ N$$

On considère qu'un état appartient à l'ensemble d'état UP s'il y a, au moins un noeud qui ne soit pas en panne, c'est-à-dire, s'il y a, au moins, un noeud dans l'état A ou dans l'état R. La fonction cfr_{UP} définit cet ensemble :

$$cfr_{UP} = (nb\ [\mathcal{N}^{(1)}\ ..\ \mathcal{N}^{(N)}]\ P)\ !=\ N$$

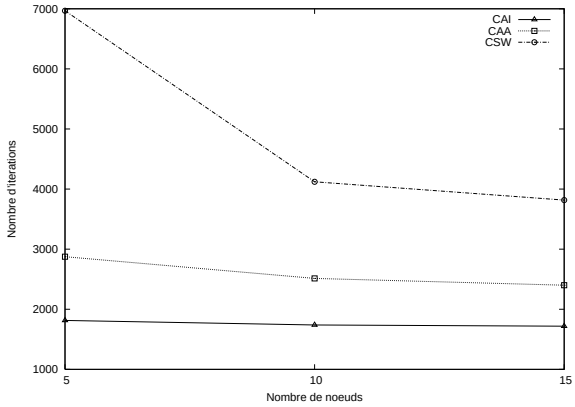
Dans nos tests, on considère que tous les noeuds sont identiques et qu'ils ont le même taux de défaillance $\lambda = 1$ par semaine, alors que le taux de réparation $\mu = 1$ par jour et le taux de reconfiguration $\tau = 12$ par heure. On remarque que les taux de transitions qu'on propose pour ces modèles sont différents des taux proposés dans [89, 90]. On préfère un autre choix de valeurs car, comme Sbeity l'a mentionné dans [89], le facteur entre les taux adoptés par lui sont plus petits (10^3) que ceux qu'on voit dans la réalité (10^6), donc on choisit les valeurs mentionnée ci-avant de façon à plus s'approcher à la réalité.

On calcule la disponibilité ponctuelle pour un instant de temps t égal à $1\ 000$ heures.

A.3.1 Nombre d'itérations

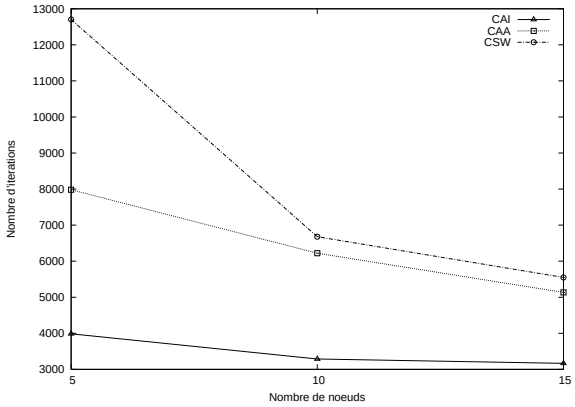
FIG. A.9 présente les mesures relevées pour les différents nombres de noeuds, alors que la FIG. A.10 présente les mesures organisées selon l'erreur maximum acceptée.

Tolérance $1e^{-5}$



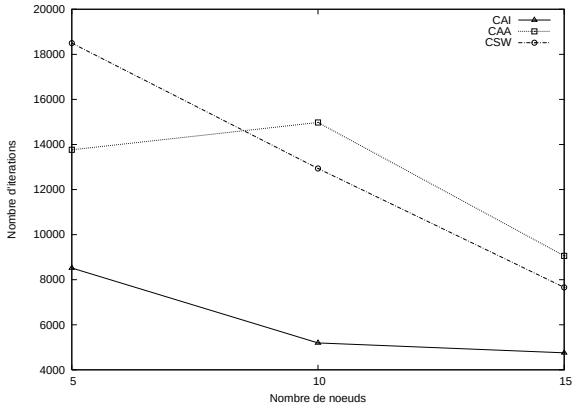
Nombre de noeuds	5	10	15
UC	61048	121480	181812
CRI	4270	7002	8468
CAI	1814	1738	1718
CAA	2874	2514	2400
CSW	6968	4121	3817

Tolérance $1e^{-7}$



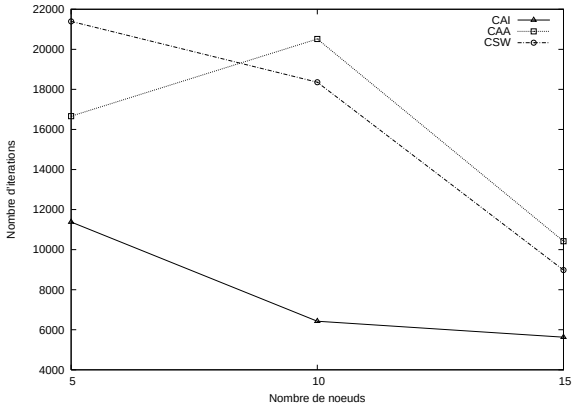
Nombre de noeuds	5	10	15
UC	61278	121805	182210
CRI	10040	18396	25608
CAI	1814	1738	1718
CAA	7982	6220	5134
CSW	12703	6679	5549

Tolérance $1e^{-9}$



Nombre de noeuds	5	10	15
UC	61475	122084	182550
CRI	15836	29986	42448
CAI	1814	1738	1718
CAA	13766	14974	9056
CSW	18498	12935	7659

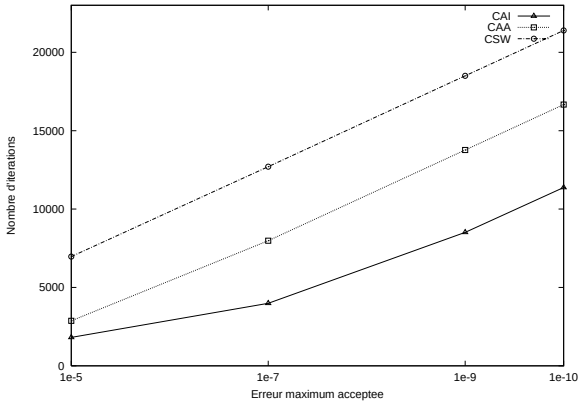
Tolérance $1e^{-10}$



Nombre de noeuds	5	10	15
UC	61565	122210	182705
CRI	18734	35782	51142
CAI	1814	1738	1718
CAA	16664	20518	10420
CSW	21395	18356	8980

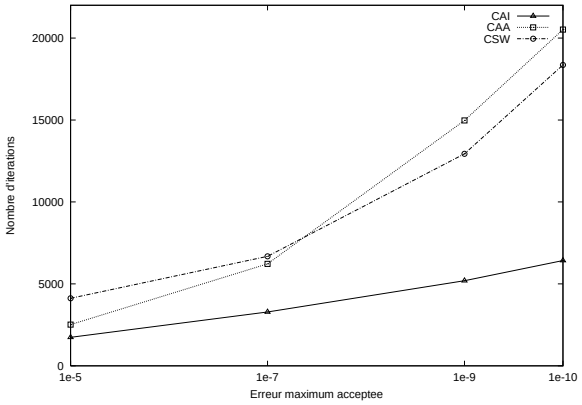
FIG. A.9 – CFR - Nombre d'itérations pour différents nombres de noeuds

5 noeuds



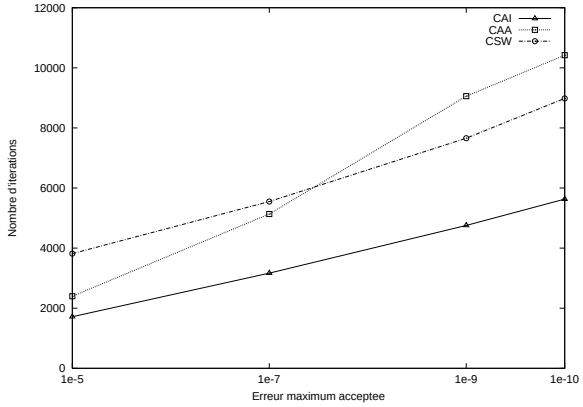
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	61048	61278	61475	61565
CRI	4270	10040	15836	18734
CAI	1814	3986	8516	11378
CAA	2874	7982	13766	16664
CSW	6968	12703	18498	21395

10 noeuds



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	121480	121805	122084	122210
CRI	7002	18396	29986	35782
CAI	1814	3986	8516	11378
CAA	2514	6220	14974	20518
CSW	4121	6679	12935	18356

15 noeuds



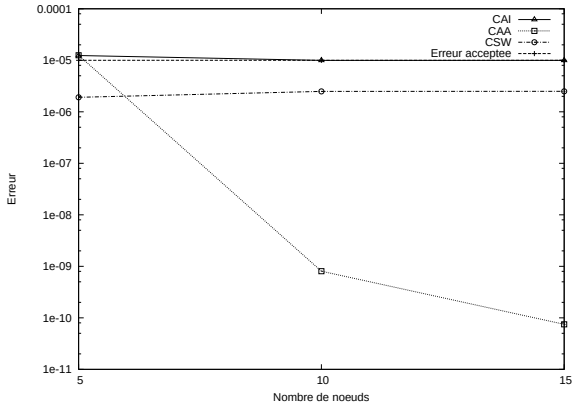
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	181812	182210	182550	182705
CRI	8468	25608	42448	51142
CAI	1814	3986	8516	11378
CAA	2400	5134	9056	10420
CSW	3817	5549	7659	8980

FIG. A.10 – CFR - Nombre d'itérations pour différentes erreurs maximum acceptées

A.3.2 Précision des résultats

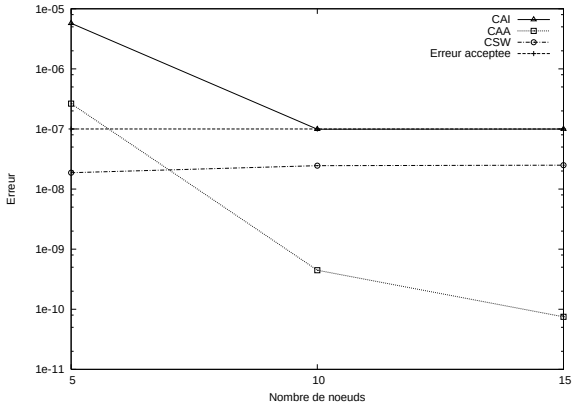
FIG. A.11 présente les mesures relevées pour différents nombre de noeuds, alors que la FIG. A.12 présente les mesures organisées selon l'erreur maximum acceptée.

Tolérance $1e^{-5}$



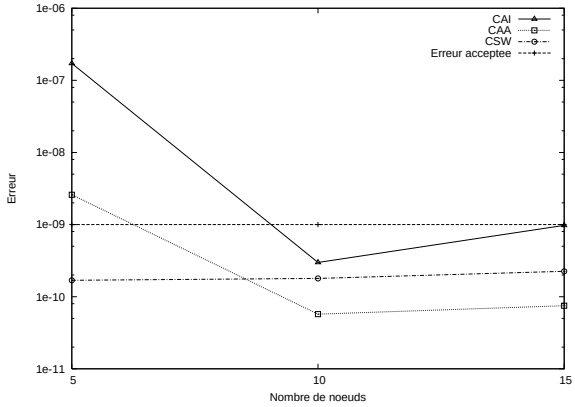
Nombre de noeuds	5	10	15
CAI	$1.24e-05$	$9.97e-06$	$9.98e-06$
CAA	$1.25e-05$	$8.04e-10$	$7.48e-11$
CSW	$1.91e-06$	$2.49e-06$	$2.50e-06$

Tolérance $1e^{-7}$



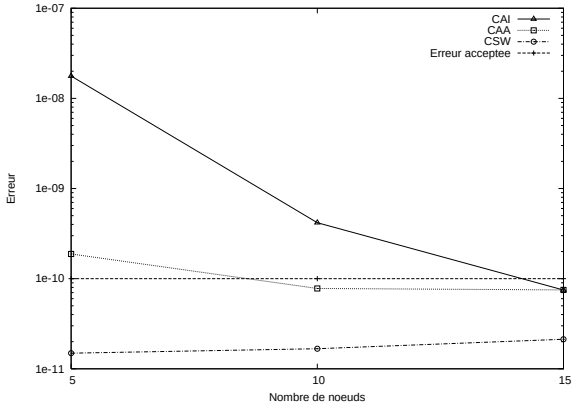
Nombre de noeuds	5	10	15
CAI	$5.73e-06$	$9.90e-08$	$9.96e-08$
CAA	$2.64e-07$	$4.46e-10$	$7.50e-11$
CSW	$1.87e-08$	$2.45e-08$	$2.50e-08$

Tolérance $1e^{-9}$



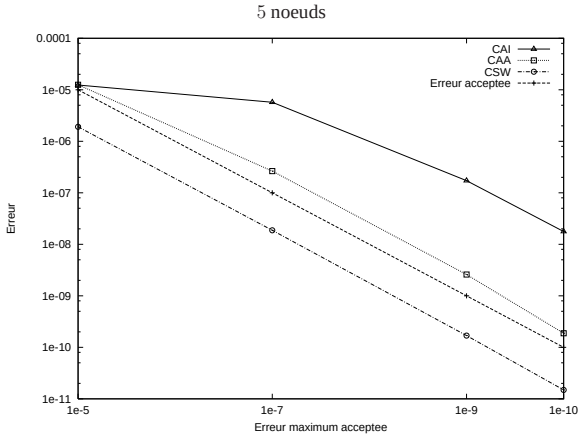
Nombre de noeuds	5	10	15
CAI	$1.72e-07$	$2.98e-10$	$9.75e-10$
CAA	$2.59e-09$	$5.73e-11$	$7.50e-11$
CSW	$1.69e-10$	$1.79e-10$	$2.25e-10$

Tolérance $1e^{-10}$

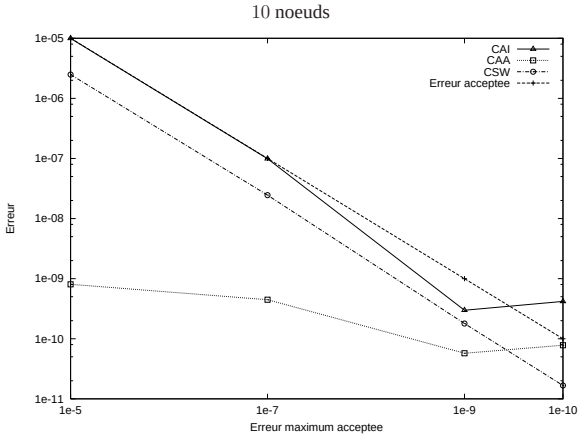


Nombre de noeuds	5	10	15
CAI	$1.77e-08$	$4.18e-10$	$7.49e-11$
CAA	$1.88e-10$	$7.79e-11$	$7.50e-11$
CSW	$1.49e-11$	$1.67e-11$	$2.13e-11$

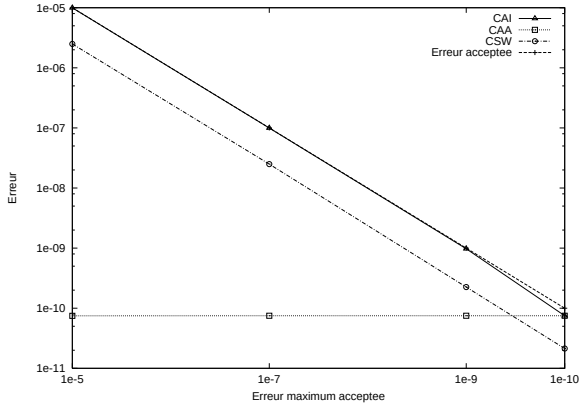
FIG. A.11 – CFR - Précision des résultats pour différents nombres de noeuds



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$1.24e-05$	$5.73e-06$	$1.72e-07$	$1.77e-08$
CAA	$1.25e-05$	$2.64e-07$	$2.59e-09$	$1.88e-10$
CSW	$1.91e-06$	$1.87e-08$	$1.69e-10$	$1.49e-11$



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$9.97e-06$	$9.90e-08$	$2.98e-10$	$4.18e-10$
CAA	$8.04e-10$	$4.46e-10$	$5.73e-11$	$7.79e-11$
CSW	$2.49e-06$	$2.45e-08$	$1.79e-10$	$1.67e-11$



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$9.98e-06$	$9.96e-08$	$9.75e-10$	$7.49e-11$
CAA	$7.48e-11$	$7.50e-11$	$7.50e-11$	$7.50e-11$
CSW	$2.50e-06$	$2.50e-08$	$2.25e-10$	$2.13e-11$

FIG. A.12 – CFR - Précision des résultats pour différentes erreurs maximum acceptées

A.4 DDS- Mesures relevées

Le modèle DDS présenté dans le chapitre 4 dans la section 4.4 (page 58) est un simplification d'un modèle classique proposé par Muntz, Souza e Silva, and Goyal dans [66]. La configuration testée pour ce modèle est basée sur le même nombre de processeurs, contrôleurs et disques que dans la version originale, comme cela a été décrit précédemment. Cependant, on considère dans notre modèle un seul type de panne, au lieu de deux type comme dans la version originale. L'espace d'états de ce modèle est de $3\ 515\ 625$ états.

L'état initial du modèle est défini par la fonction dds_{init} :

$$\begin{aligned} dds_{init} =& (st\ \mathcal{P}_{\mathcal{A}}\ ==\ 0)\ \&\&\ (st\mathcal{P}_{\mathcal{B}} == 0)\ \&\&(st\ \mathcal{C}_1 == 0)\ \&\&\ (st\ \mathcal{C}_2 == 0)\ \&\& \\ & (nb\ [\mathcal{D}_{11}\ ..\ \mathcal{D}_{23}]\ 0 == 6) \end{aligned}$$

Cette fonction indique l'absence de panne dans le modèle.

L'ensemble des états UP est un peu plus difficile à définir. On considère qu'un état appartient à l'ensemble d'état UP si il y a au moins un processeur en marche, soit il du type A ou du type B. Il y a au moins un contrôleur de chaque type en marche et, il y a au plus un disque en panne dans chaque pile de disque. La fonction dds_{UP} ci-dessous définit cet ensemble d'états UP :

$$\begin{aligned} dds_{UP} =& ((st\ \mathcal{P}_{\mathcal{A}}\ +\ st\ \mathcal{P}_{\mathcal{B}})\ <=\ 7)\ \&\&\ ((st\ \mathcal{C}_1\ !=\ 2)\ \&\&\ (st\ \mathcal{C}_2\ !=\ 2))\ \&\& \\ & (((nb\ [\mathcal{D}_{11}\ ..\ \mathcal{D}_{23}]\ 0)\ +\ (nb\ [\mathcal{D}_{11}\ ..\ \mathcal{D}_{23}]\ 1)) == 6) \end{aligned}$$

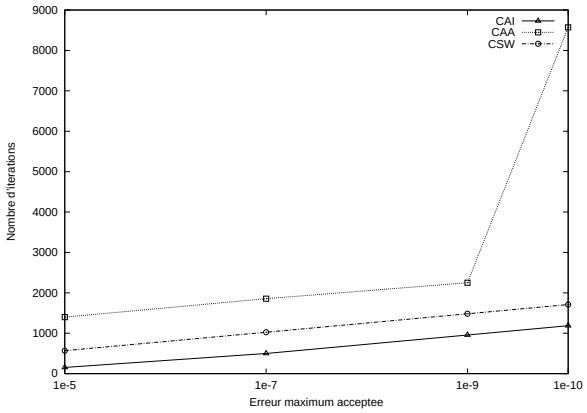
Les taux de transition adoptés dans nos tests sont égaux aux taux adoptés par Carrasco dans [17]. On rappelle les taux de panne et réparation des composants du modèle dans TAB. A.4. Tous les taux sont exprimés en heures.

Taux	Valeur	Taux	Valeur	Taux	Valeur
λ_a	$1/2000$	τ_{11}	$1/6000$	τ_{22}	$1/14000$
λ_b	$1/2000$	τ_{12}	$1/8000$	τ_{23}	$1/16000$
γ_1	$1/2000$	τ_{13}	$1/10000$	μ	$1/10$
γ_2	$1/4000$	τ_{21}	$1/12000$		

TAB. A.1 – Taux de panne et réparation

L'instant de temps utilisé pour le calcul de la disponibilité ponctuelle est $1\ 500$ heures.

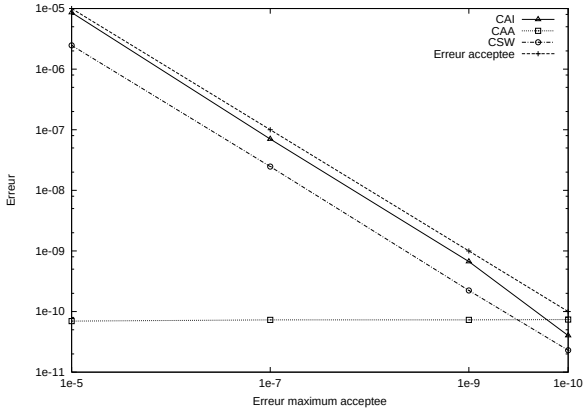
A.4.1 Nombre d'itérations



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$8.58e-06$	$7.03e-08$	$6.69e-10$	$4.02e-11$
CAA	$6.96e-11$	$7.28e-11$	$7.29e-11$	$7.38e-11$
CSW	$2.47e-06$	$2.47e-08$	$2.23e-10$	$2.30e-11$

FIG. A.13 – DDS - Nombre d'itérations pour différentes erreurs maximum acceptées

A.4.2 Précision des résultats



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	5306	5373	5431	5458
CRI	68	126	188	220
CAI	174	280	363	402
CAA	349	420	490	523
CSW	260	344	421	457

FIG. A.14 – DDS - Précision des résultats pour différentes erreurs maximum acceptées

A.5 RWP- Mesures relevées

L'inclusion du modèle RWP (Chapitre 4, Section 4.5, page 61) dans nos tests a pour objectif tester l'efficacité des méthodes de détection du régime stationnaire face à différents ensembles d'états UP. De cette manière, ce modèle est testé avec 3 configurations. Il faut remarquer qu'on change uniquement le vecteur d'états UP. La taille du modèle reste toujours la même. C'est-à-dire, $390\ 625$ états.

La fonction qui représente l'ensemble d'état UP est :

$$\begin{aligned} rwp_{UP} =& (max\ ((st\ \mathcal{LOC}_{\mathcal{X}}\ - pc),\ (pc - st\ \mathcal{LOC}_{\mathcal{X}})) + \\ & max((st\ \mathcal{LOC}_{\mathcal{Y}}\ - pc),\ (pc - st\ \mathcal{LOC}_{\mathcal{X}}))) < dist; \end{aligned}$$

pc représente le *point central* du plan cartésien. Par exemple, dans le plan adopté pour nos tests (25×25), pc est égal à 13, car il est au milieu du plan. $dist$ représente la distance acceptée à partir de pc, en nombre de mouvement, pour que le noeud soit considéré dans l'espace de couverture. On a choisi 5, 8 et 11 comme distances pour nos tests. En utilisant la fonction rwp_{UP} avec $dist\ =\ 5$ on a un ensemble d'états UP égal à $25\ 625$ états, $70\ 625$ états pour $dist = 8$ et $138\ 125$ états pour $dist = 11$. Ce qui représente, respectivement, 6.56%, 18.08% et 35.36% de l'espace d'états du modèle.

On considère qui à l'instant de temps initial le noeud est dans la position $0, 0$ et que sa destination est aussi $0, 0$. La fonction rwp_{init} définit cet état initial :

$$\begin{aligned} rwp_{init} =& (st\ \mathcal{DEST}_{\mathcal{X}}\ ==\ 0)\ \&\&\ (st\ \mathcal{DEST}_{\mathcal{Y}}\ ==\ 0)\ \&\& \\ & (st\ \mathcal{LOC}_{\mathcal{X}}\ ==\ 0)\ \&\&\ (st\mathcal{LOC}_{\mathcal{Y}} == 0); \end{aligned}$$

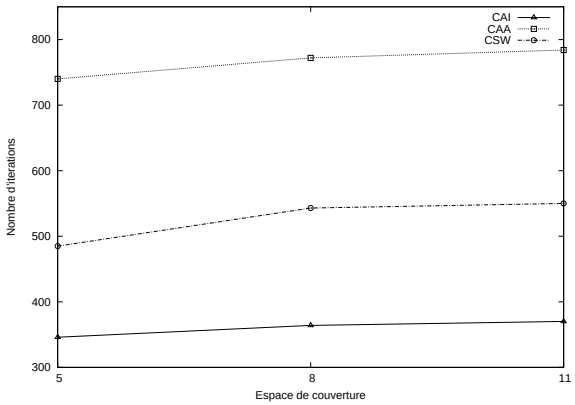
Dans nos tests, on considère une surface totale de $1\ 000$ mètres par $1\ 000$ mètres divisée en 25 morceaux. Donc chaque morceau a une surface de 10×10 mètres. Étant donné que la vitesse de déplacement du noeud est de $2\ m/s$, le taux de déplacement entre chaque morceau est $2/10$ par seconde. Le taux de tirage d'une nouvelle destination est calculé à partir du temps de pause du noeud. On définit un temps de pause de 10 secondes. Donc le taux de routage est $0, 1$ par seconde.

On calcule la disponibilité ponctuelle pour un instant de temps t égal à 250 minutes.

A.5.1 Nombre d'itérations

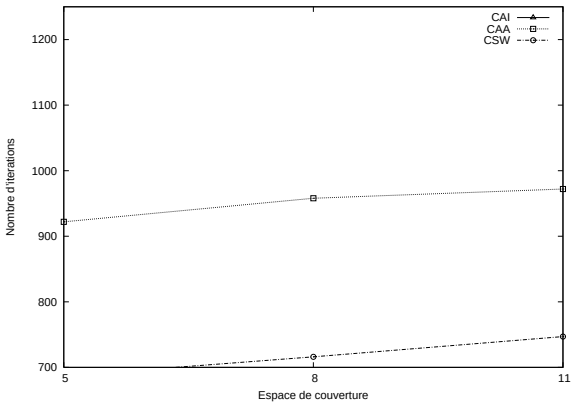
FIG. A.15 présente les mesures relevées pour les différents espace de couverture , alors que la FIG. A.16 présente les mesures organisées selon l'erreur maximum acceptée.

Tolérance $1e^{-5}$



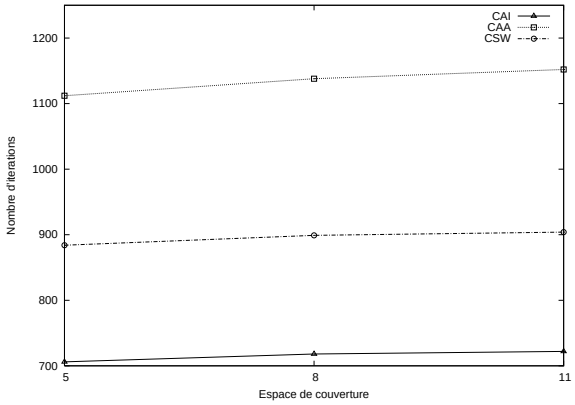
Espace de couverture	5	8	11
UC	1138	1138	1138
CRI	236	236	236
CAI	346	364	370
CAA	740	772	784
CSW	485	543	550

Tolérance $1e^{-7}$



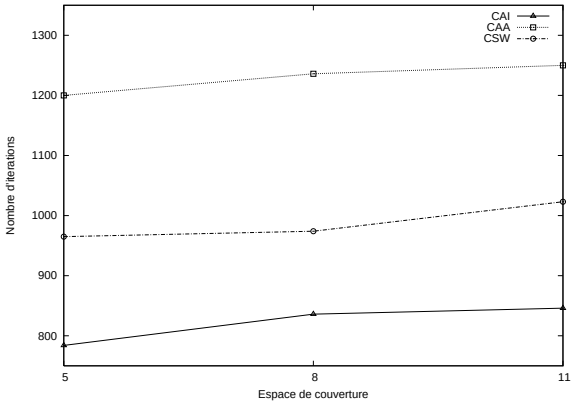
Espace de couverture	5	8	11
UC	1169	1169	1169
CRI	438	438	438
CAI	506	560	568
CAA	922	958	972
CSW	687	716	747

Tolérance $1e^{-9}$



Espace de couverture	5	8	11
UC	1195	1195	1195
CRI	498	498	498
CAI	706	718	722
CAA	1112	1138	1152
CSW	884	899	904

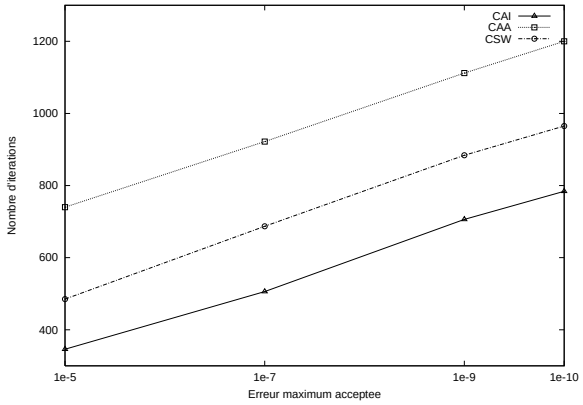
Tolérance $1e^{-10}$



Espace de couverture	5	8	11
UC	1208	1208	1208
CRI	778	778	778
CAI	784	836	846
CAA	1200	1236	1250
CSW	965	974	1023

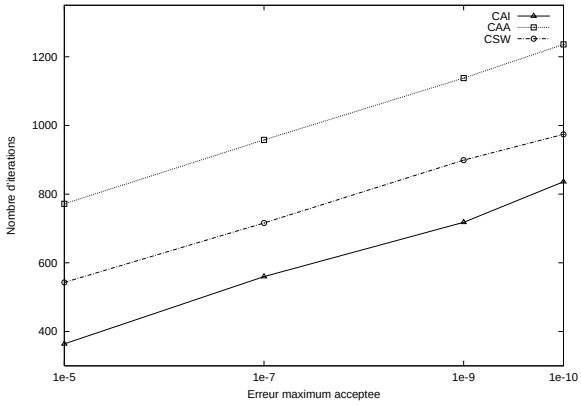
FIG. A.15 – RWP - Nombre d'itérations pour différentes aires de couverture

Espace de couverture de 5



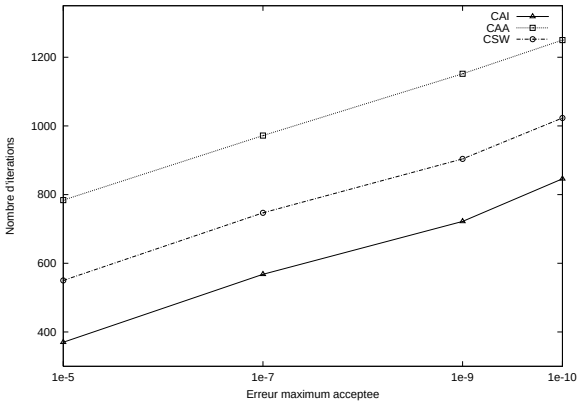
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	1138	1169	1195	1208
CRI	114	246	416	490
CAI	346	506	706	784
CAA	740	922	1112	1200
CSW	485	687	884	965

Espace de couverture de 8



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	1138	1169	1195	1208
CRI	114	246	416	490
CAI	364	560	718	836
CAA	772	958	1138	1236
CSW	543	716	899	974

Espace de couverture de 11



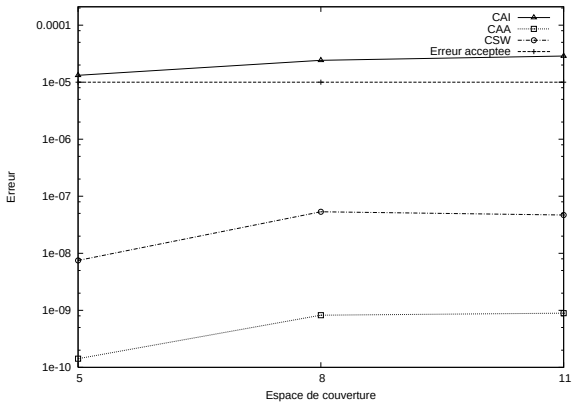
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	1138	1169	1195	1208
CRI	114	246	416	490
CAI	370	568	722	846
CAA	784	972	1152	1250
CSW	550	747	904	1023

FIG. A.16 – RWP - Nombre d'itérations pour différentes erreurs maximum acceptées

A.5.2 Précision des résultats

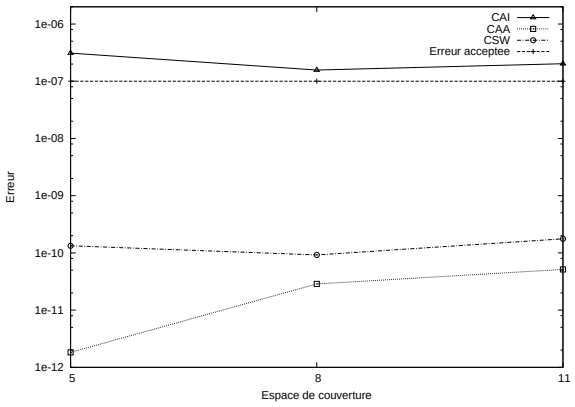
FIG. A.17 présente les mesures relevées pour différents espace de couverture, alors que la FIG. A.18 présente les mesures organisées selon l'erreur maximum acceptée.

Tolérance $1e^{-5}$



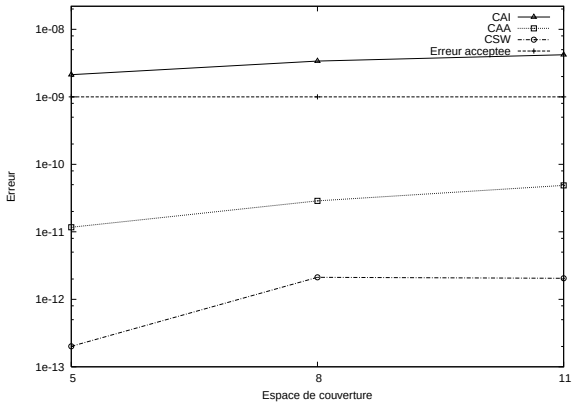
Espace de couverture	5	8	11
CAI	$1.32e-05$	$2.42e-05$	$2.88e-05$
CAA	$1.42e-10$	$8.21e-10$	$8.90e-10$
CSW	$7.49e-09$	$5.34e-08$	$4.69e-08$

Tolérance $1e^{-7}$



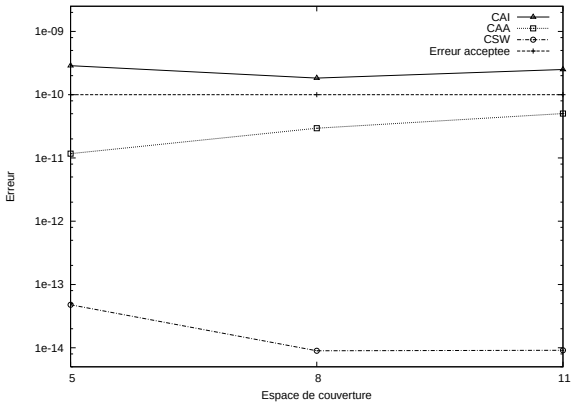
Espace de couverture	5	8	11
CAI	$3.10e-07$	$1.57e-07$	$2.02e-07$
CAA	$1.83e-12$	$2.87e-11$	$5.15e-11$
CSW	$1.33e-10$	$9.21e-11$	$1.76e-10$

Tolérance $1e^{-9}$



Espace de couverture	5	8	11
CAI	$2.13e-09$	$3.39e-09$	$4.20e-09$
CAA	$1.17e-11$	$2.88e-11$	$4.88e-11$
CSW	$2.01e-13$	$2.12e-12$	$2.05e-12$

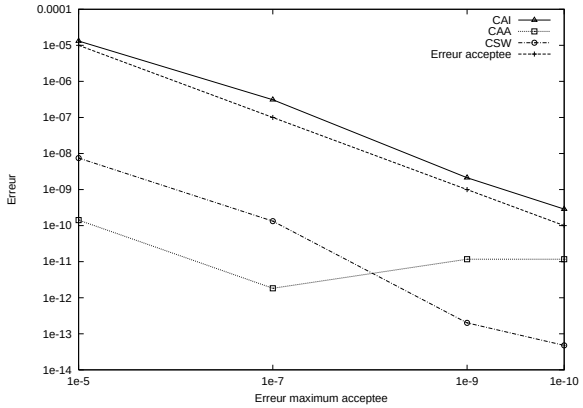
Tolérance $1e^{-10}$



Espace de couverture	5	8	11
CAI	$2.87e-10$	$1.83e-10$	$2.50e-10$
CAA	$1.17e-11$	$2.95e-11$	$5.04e-11$
CSW	$4.78e-14$	$8.96e-15$	$9.13e-15$

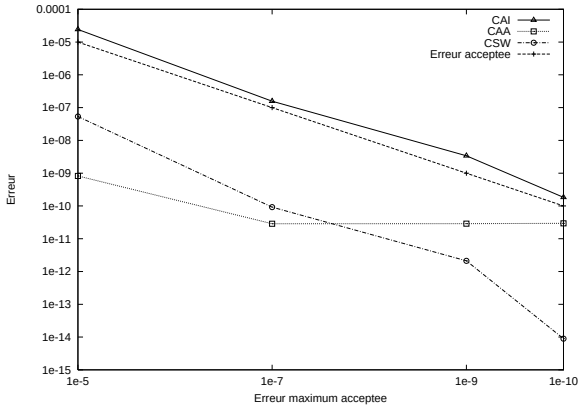
FIG. A.17 – RWP - Précision des résultats pour différentes aires de couverture

Espace de couverture de 5



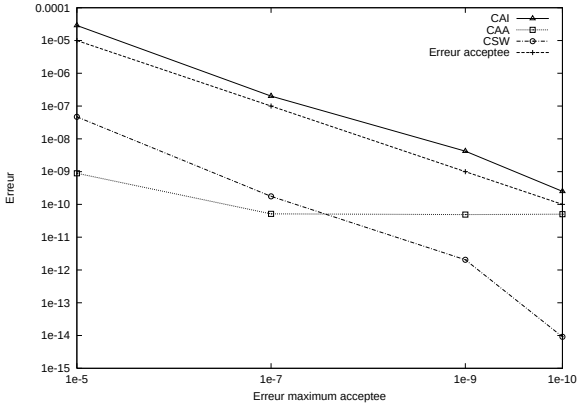
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$1.32e-05$	$3.10e-07$	$2.13e-09$	$2.87e-10$
CAA	$1.42e-10$	$1.83e-12$	$1.17e-11$	$1.17e-11$
CSW	$7.49e-09$	$1.33e-10$	$2.01e-13$	$4.78e-14$

Espace de couverture de 8



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$2.42e-05$	$1.57e-07$	$3.39e-09$	$1.83e-10$
CAA	$8.21e-10$	$2.87e-11$	$2.88e-11$	$2.95e-11$
CSW	$5.34e-08$	$9.21e-11$	$2.12e-12$	$8.96e-15$

Espace de couverture de 11



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$2.88e-05$	$2.02e-07$	$4.20e-09$	$2.50e-10$
CAA	$8.90e-10$	$5.15e-11$	$4.88e-11$	$5.04e-11$
CSW	$4.69e-08$	$1.76e-10$	$2.05e-12$	$9.13e-15$

FIG. A.18 – RWP - Précision des résultats pour différentes erreurs maximum acceptées

A.6 WORKSTATION- Mesures relevées

Le modèle WORKSTATION (Chapitre 4, Section 4.6, page 63) nous a permis de tester plusieurs configurations avec un seul exemple. D'abord, on a testé le modèle avec 4 nombres de serveurs. Les modèles testés ont 32, 256, 512 et $1\ 024$ serveurs par grappe. Dans nos tests, on a adopté un modèle avec seulement 2 grappes de façon à être conforme au modèle original proposé dans [52]. Toutefois, n'importe quel nombre de grappe peut être utilisé. On remarque que pour un modèle avec 2 grappe symétrique, lorsqu'on double le nombre de serveurs dans une grappe, la taille de l'espace d'états est multipliée par 4. Pour les nombres de serveurs qu'on a choisi, on a, respectivement, des espaces d'états de $17\ 424$, $1\ 056\ 784$, $4\ 210\ 704$ et $16\ 810\ 000$ états.

Pour chaque nombre de serveurs, on a testé également 5 niveaux de disponibilité. Chaque niveau de disponibilité définit un ensemble différents d'états UP. On a considéré dans nos tests les disponibilités suivantes 10%, 30%, 50%, 70% et 90% des serveurs. TAB. A.6 présente le nombre d'états UP pour les différents niveaux de disponibilité par rapport au nombre de serveurs.

Nombre de serveur par grappe	Espace d'états	Niveaux de disponibilité				
		10%	30%	50%	70%	90%
32 serveurs	17 424	11 066	6 568	4 980	420	56
256 serveurs	1 056 784	677 512	404 662	69 388	23 870	2 756
512 serveurs	4 210 704	2 703 386	1 609 096	269 836	95 172	10 712
1024 serveurs	16 810 000	10 800 220	6 429 210	1 063 948	378 840	42 230

TAB. A.2 – Nombre d'états UP pour les niveaux de disponibilité

La fonction $Workstation_{UP}$ définit l'ensemble d'états UP :

$$\begin{aligned} Workstation_{UP} =& ((st\ \mathcal{G}^{(1)}\ =>\ niv\_disp)\ \&\&(st\ \mathcal{S}^{(1)}\ ==\ A))\ ||\\ & ((st\ \mathcal{G}^{(2)}\ =>\ niv\_disp)\ \&\&(st\ \mathcal{S}^{(2)}\ ==\ A))\ ||\\ & (((st\ \mathcal{G}^{(1)} + st\ \mathcal{G}^{(2)})\ =>\ niv\_disp)\ \&\&\\ & (st\ \mathcal{S}^{(1)}\ ==\ A)\ \&\&\ (st\ \mathcal{S}^{(2)}\ ==\ A)\ \&\&\\ & (st\ \mathcal{B}\ ==\ A));\end{aligned}$$

On considère un état initial où il n'y a aucune panne dans le système et l'unité de réparation est en repos. La fonction $Workstation_{init}$ définit cet état :

$$\begin{aligned} Workstation_{init} =& (st\ \mathcal{G}^{(1)}\ =\ N)\ \&\&(st\ \mathcal{G}^{(2)}\ =\ N)\ \&\&\\ & (st\ \mathcal{S}^{(1)}\ ==\ A))\ \&\&(st\ \mathcal{S}^{(2)}\ ==\ A))\ \&\&\\ & (st\ \mathcal{B}\ ==\ A)\ \&\&(st\ \mathcal{URS}\ ==\ I);\end{aligned}$$

Les taux de transition adoptés dans nos tests sont égaux aux taux adoptés dans [52]. On rappelle les taux de panne et de réparation des composants du modèle dans TAB. A.6. Tous les taux sont exprimés en heures.

Taux	Valeur	Taux	Valeur	Taux	Valeur
λ_1	$1/500$	μ_1	2	γ	$1/5000$
λ_2	$1/500$	μ_2	2	δ	0.125
ρ_1	$1/4000$	τ_1	0.25	α	10
ρ_2	$1/4000$	τ_2	0.25		

TAB. A.3 – Taux de panne et réparation

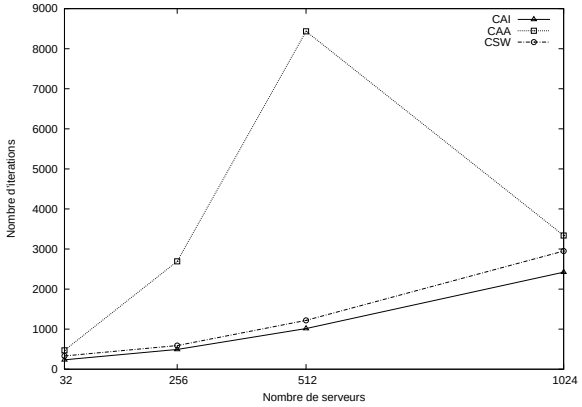
L'instant de temps utilisé pour le calcul de la disponibilité ponctuelle est $10\ 000$ heures.

A.6.1 WORKSTATION- 10% de serveurs disponibles

Nombre d'itérations

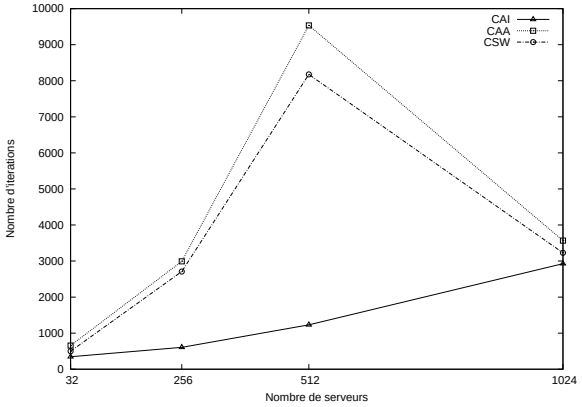
FIG. A.19 présente les mesures relevées pour les différents nombres de serveurs pour un niveau de disponibilité de 10%, alors que la FIG. A.20 présente les mesures organisées selon l'erreur maximum acceptée.

Tolérance $1e^{-5}$



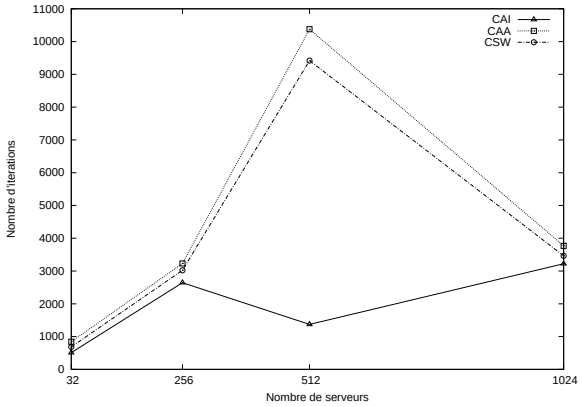
Nombre de serveurs	32	256	512	1024
UC	102645	111664	121968	142569
CRI	98	136	408	374
CAI	235	493	1015	2422
CAA	471	2693	8433	3338
CSW	331	590	1220	2949

Tolérance $1e^{-7}$



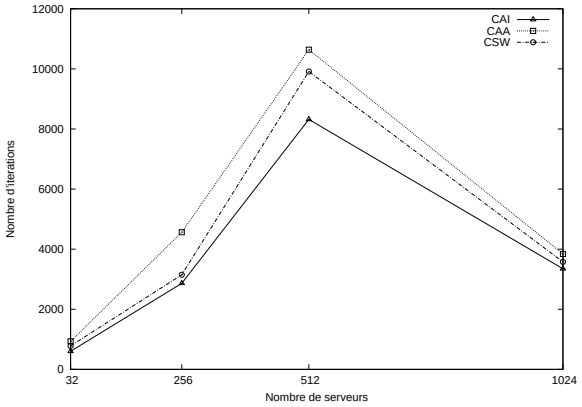
Nombre de serveurs	32	256	512	1024
UC	102944	111976	122294	142921
CRI	428	438	1088	7470
CAI	346	606	1232	2927
CAA	657	2991	9534	3566
CSW	499	2710	8175	3227

Tolérance $1e^{-9}$



Nombre de serveurs	32	256	512	1024
UC	103200	112244	122573	143223
CRI	802	886	1872	39796
CAI	508	2643	1372	3222
CAA	842	3234	10377	3769
CSW	685	3021	9418	3465

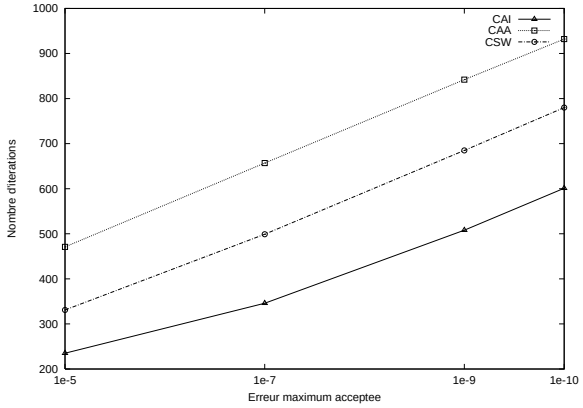
Tolérance $1e^{-10}$



Nombre de serveurs	32	256	512	1024
UC	103316	112364	122700	143360
CRI	990	1112	2278	56042
CAI	602	2862	8317	3348
CAA	932	4564	10640	3842
CSW	780	3147	9914	3576

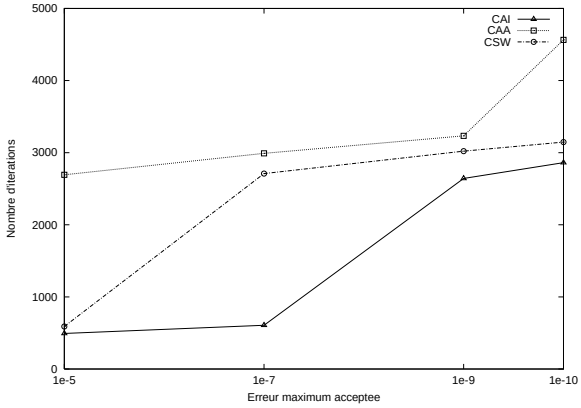
FIG. A.19 – WORKSTATION (10% des serveurs disponibles) - Nombre d'itérations pour différents nombres de serveurs

32 serveurs par grappe



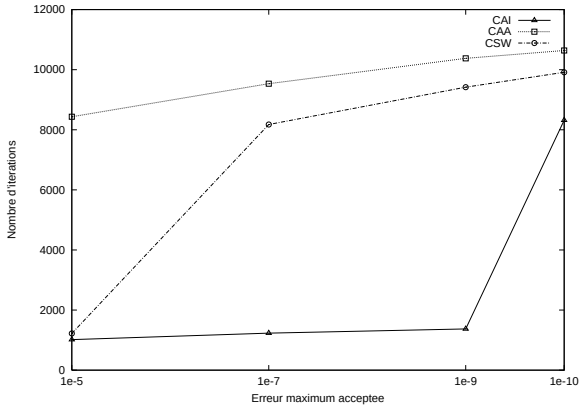
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	102645	102944	103200	103316
CRI	98	428	802	990
CAI	235	346	508	601
CAA	471	657	842	932
CSW	331	499	685	780

256 serveurs par grappe



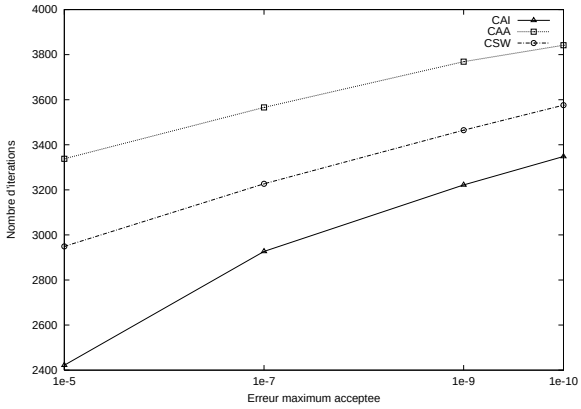
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	111664	111976	112244	112364
CRI	136	438	886	1112
CAI	493	606	2643	2862
CAA	2693	2991	3233	4564
CSW	590	2710	3021	3147

512 serveurs par grappe



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	121968	122294	122573	122700
CRI	408	1088	1872	2278
CAI	1015	1232	1372	8317
CAA	8433	9534	10377	10640
CSW	1220	8175	9418	9914

1024 serveurs par grappe



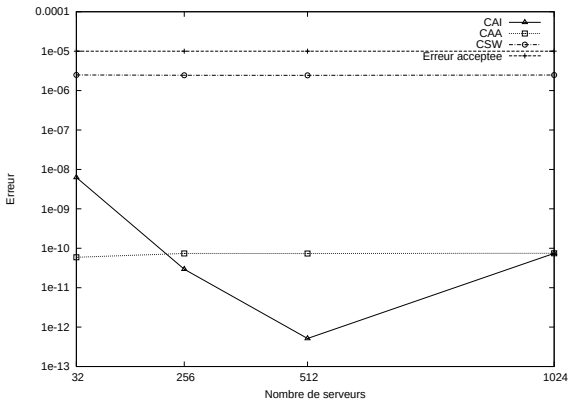
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	142569	142921	143223	143360
CRI	374	7470	39796	56042
CAI	2422	2927	3222	3348
CAA	3338	3566	3769	3842
CSW	2949	3227	3465	3576

FIG. A.20 – WORKSTATION (10% des serveurs disponibles) - Nombre d'itérations pour différentes erreurs maximum acceptées

Précision des résultats

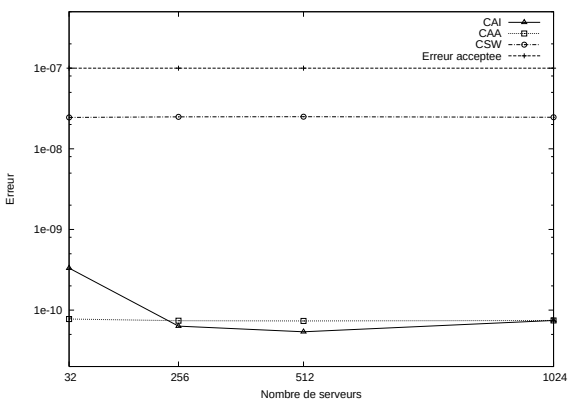
FIG. A.21 présente les mesures relevées pour différents nombres de serveurs pour un niveau de disponibilité de 10%, alors que la FIG. A.22 présente les mesures organisées selon l'erreur maximum acceptée.

Tolérance $1e^{-5}$



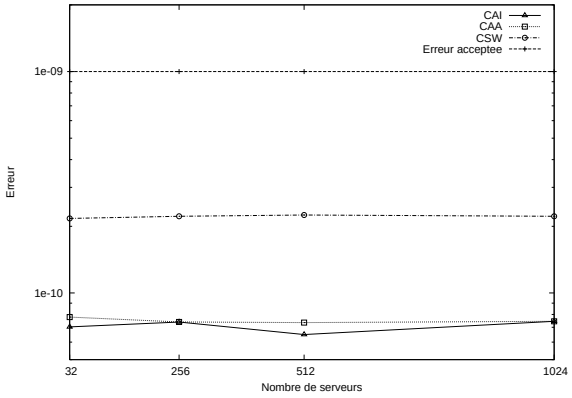
Nombre de serveurs	32	256	512	1024
CAI	$6.26e-09$	$2.94e-11$	$5.14e-13$	$7.44e-11$
CAA	$5.91e-11$	$7.39e-11$	$7.35e-11$	$7.45e-11$
CSW	$2.49e-06$	$2.45e-06$	$2.44e-06$	$2.48e-06$

Tolérance $1e^{-7}$



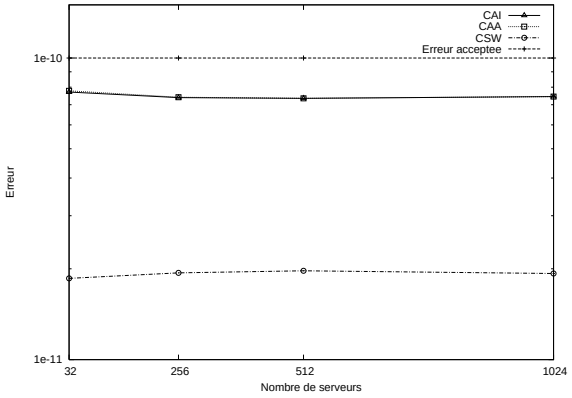
Nombre de serveurs	32	256	512	1024
CAI	$3.32e-10$	$6.34e-11$	$5.39e-11$	$7.45e-11$
CAA	$7.76e-11$	$7.39e-11$	$7.35e-11$	$7.45e-11$
CSW	$2.45e-08$	$2.49e-08$	$2.50e-08$	$2.46e-0$

Tolérance $1e^{-9}$



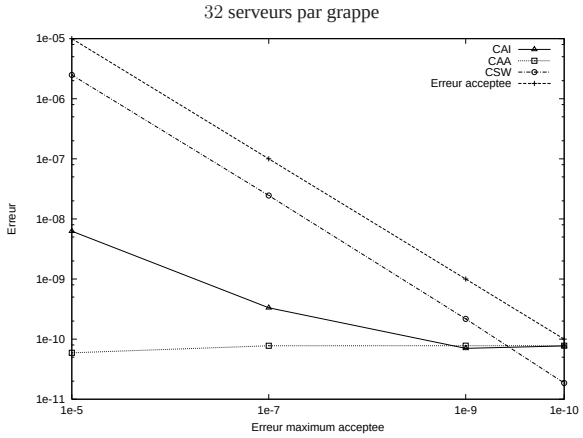
Nombre de serveurs	32	256	512	1024
CAI	$7.03e-11$	$7.39e-11$	$6.49e-11$	$7.45e-11$
CAA	$7.78e-11$	$7.40e-11$	$7.35e-11$	$7.45e-11$
CSW	$2.17e-10$	$2.22e-10$	$2.25e-10$	$2.22e-10$

Tolérance $1e^{-10}$

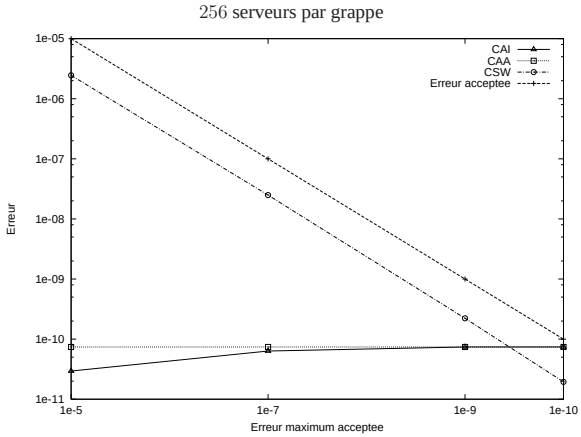


Nombre de serveurs	32	256	512	1024
CAI	$7.71e-11$	$7.39e-11$	$7.35e-11$	$7.45e-11$
CAA	$7.78e-11$	$7.41e-11$	$7.35e-11$	$7.45e-11$
CSW	$1.86e-11$	$1.94e-11$	$1.97e-11$	$1.93e-11$

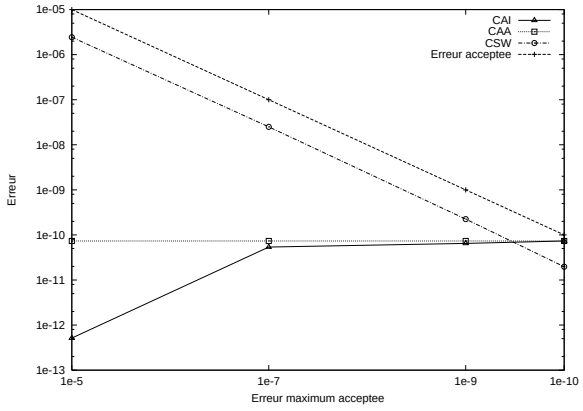
FIG. A.21 – WORKSTATION (10% des serveurs disponibles) - Précision des résultats pour différents nombres de serveurs



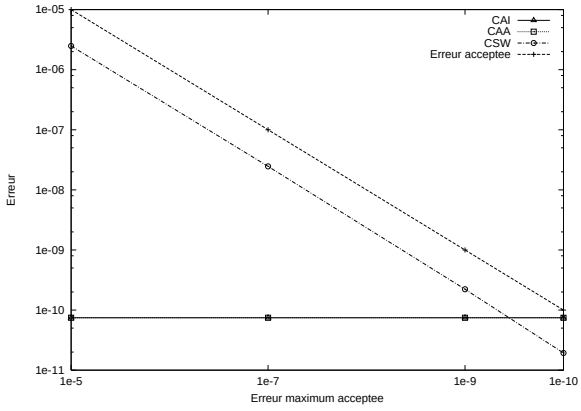
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$6.26e-09$	$3.32e-10$	$7.03e-11$	$7.71e-11$
CAA	$5.91e-11$	$7.76e-11$	$7.78e-11$	$7.78e-11$
CSW	$2.49e-06$	$2.45e-08$	$2.17e-10$	$1.86e-11$



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$2.94e-11$	$6.34e-11$	$7.39e-11$	$7.39e-11$
CAA	$7.39e-11$	$7.39e-11$	$7.40e-11$	$7.41e-11$
CSW	$2.45e-06$	$2.49e-08$	$2.22e-10$	$1.94e-11$



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$5.14e-13$	$5.39e-11$	$6.49e-11$	$7.35e-11$
CAA	$7.35e-11$	$7.35e-11$	$7.35e-11$	$7.35e-11$
CSW	$2.44e-06$	$2.50e-08$	$2.25e-10$	$1.97e-11$



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$7.44e-11$	$7.45e-11$	$7.45e-11$	$7.45e-11$
CAA	$7.45e-11$	$7.45e-11$	$7.45e-11$	$7.45e-11$
CSW	$2.48e-06$	$2.46e-08$	$2.22e-10$	$1.93e-11$

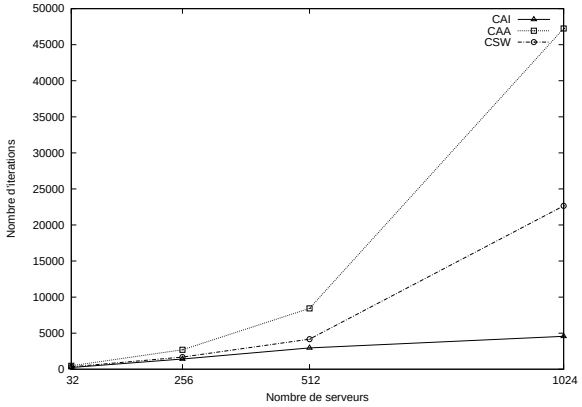
FIG. A.22 – WORKSTATION (10% des serveurs disponibles) - Précision des résultats pour différentes erreurs maximum acceptées

A.6.2 WORKSTATION- 30% de serveurs disponibles

Nombre d'itérations

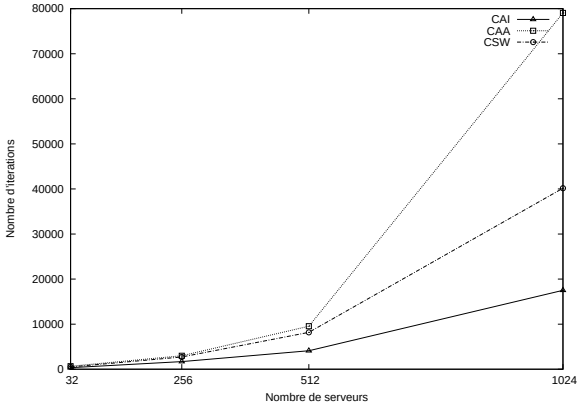
FIG. A.23 présente les mesures relevées pour les différents nombres de serveurs pour un niveau de disponibilité de 30%, alors que la FIG. A.24 présente les mesures organisées selon l'erreur maximum acceptée.

Tolérance $1e^{-5}$



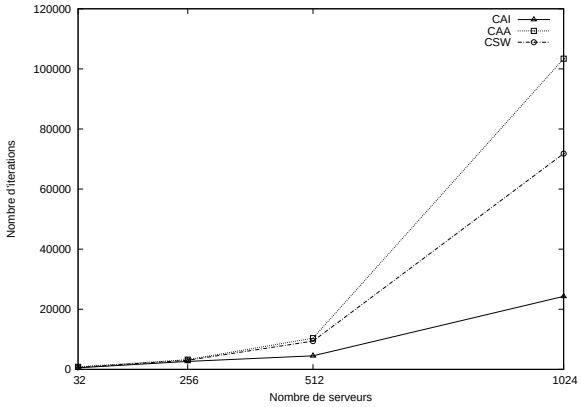
Nombre de serveurs	32	256	512	1024
UC	102645	111664	121968	142569
CRI	98	136	408	374
CAI	240	1413	2954	4565
CAA	471	2693	8433	47254
CSW	331	1685	4169	22657

Tolérance $1e^{-7}$



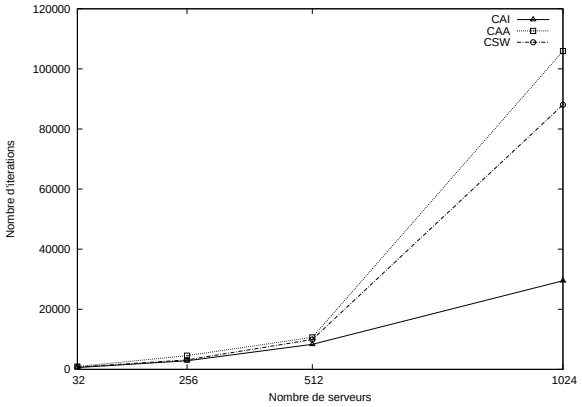
Nombre de serveurs	32	256	512	1024
UC	102944	111976	122294	142921
CRI	428	438	1088	7470
CAI	346	1688	4099	17494
CAA	657	2991	9534	79051
CSW	499	2710	8175	40158

Tolérance $1e^{-9}$



Nombre de serveurs	32	256	512	1024
UC	103200	112244	122573	143223
CRI	802	886	1872	39796
CAI	508	2643	4512	24291
CAA	842	3233	10377	103408
CSW	685	3021	9418	71810

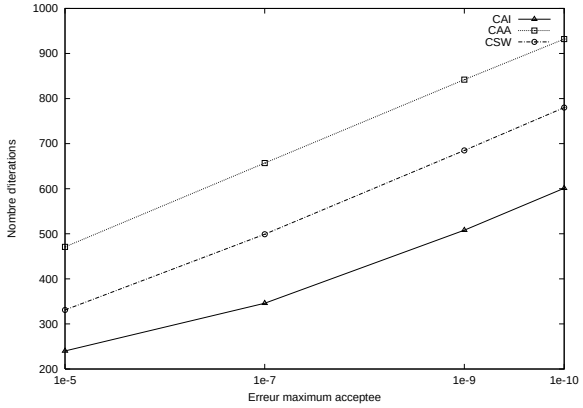
Tolérance $1e^{-10}$



Nombre de serveurs	32	256	512	1024
UC	103316	112364	122700	143360
CRI	990	1112	2278	56042
CAI	601	2862	8317	29479
CAA	932	4564	10640	105920
CSW	780	3147	9914	88048

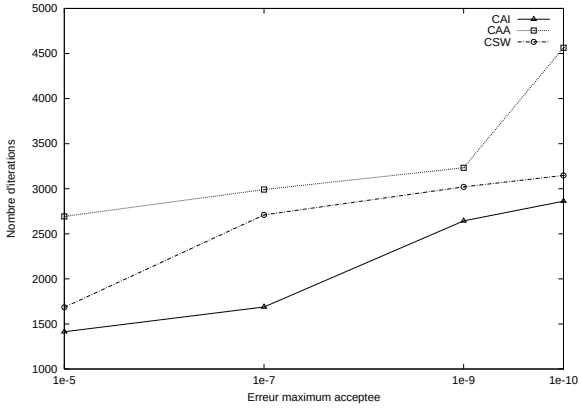
FIG. A.23 – WORKSTATION (30% des serveurs disponibles) - Nombre d'itérations pour différents nombres de serveurs

32 serveurs par grappe



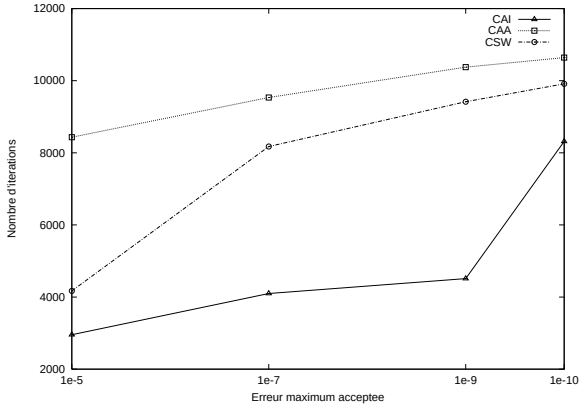
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	102645	102944	103200	103316
CRI	98	428	802	990
CAI	240	346	508	601
CAA	471	657	842	932
CSW	331	499	685	780

256 serveurs par grappe



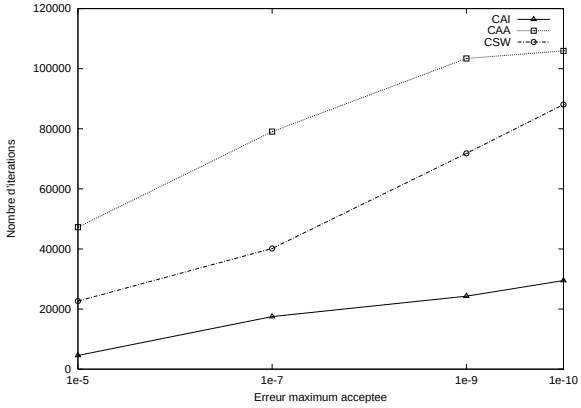
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	111664	111976	112244	112364
CRI	136	438	886	1112
CAI	1413	1688	2643	2862
CAA	2693	2991	3233	4564
CSW	1685	2710	3021	3147

512 serveurs par grappe



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	121968	122294	122573	122700
CRI	408	1088	1872	2278
CAI	2954	4099	4512	8317
CAA	8433	9534	10377	10640
CSW	4169	8175	9418	9914

1024 serveurs par grappe



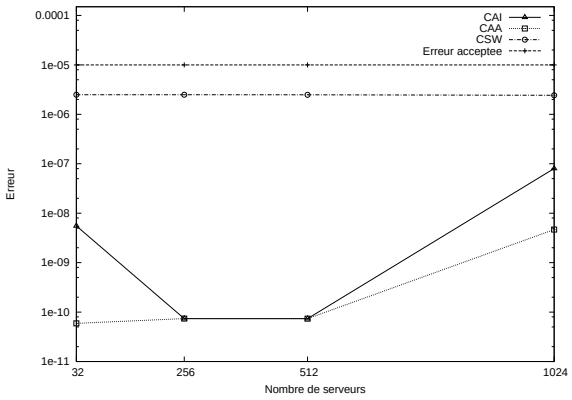
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	142569	142921	143223	143360
CRI	374	7470	39796	56042
CAI	4565	17494	24291	29479
CAA	47254	79051	103408	105920
CSW	22657	40158	71810	88048

FIG. A.24 – WORKSTATION (30% des serveurs disponibles) - Nombre d'itérations pour différentes erreurs maximum acceptées

Précision des résultats

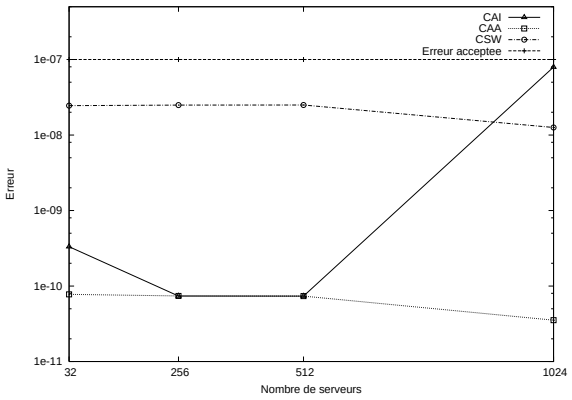
FIG. A.25 présente les mesures relevées pour différents nombres de serveurs pour un niveau de disponibilité de 30%, alors que la FIG. A.26 présente les mesures organisées selon l'erreur maximum acceptée.

Tolérance $1e^{-5}$



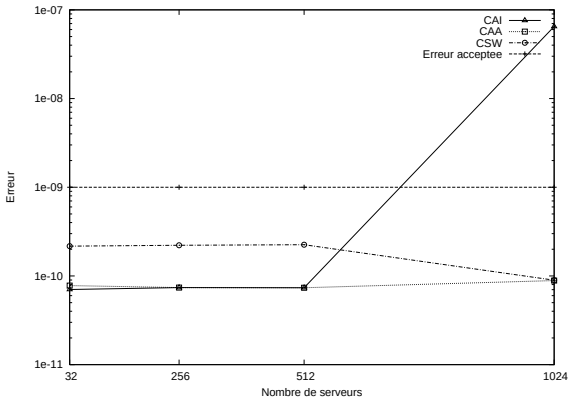
Nombre de serveurs	32	256	512	1024
CAI	$5.53e-09$	$7.37e-11$	$7.36e-11$	$8.02e-08$
CAA	$5.91e-11$	$7.39e-11$	$7.36e-11$	$4.67e-09$
CSW	$2.49e-06$	$2.49e-06$	$2.48e-06$	$2.43e-06$

Tolérance $1e^{-7}$



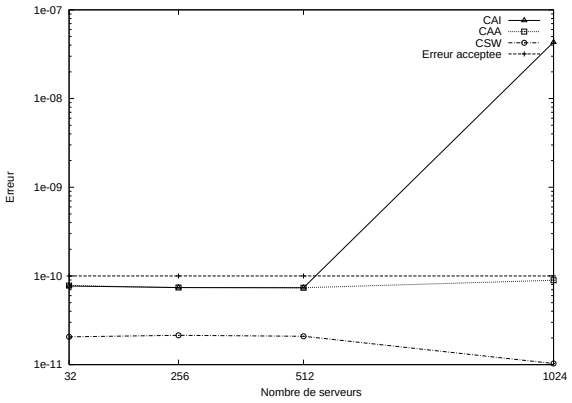
Nombre de serveurs	32	256	512	1024
CAI	$3.32e-10$	$7.37e-11$	$7.36e-11$	$7.95-08$
CAA	$7.76e-11$	$7.39e-11$	$7.36e-11$	$3.53e-11$
CSW	$2.45e-08$	$2.49e-08$	$2.50e-08$	$1.26e-08$

Tolérance $1e^{-9}$



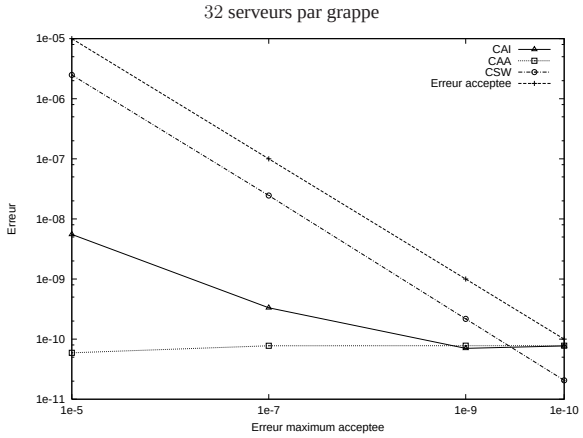
Nombre de serveurs	32	256	512	1024
CAI	$7.03e-11$	$7.39e-11$	$7.36e-11$	$6.49e-08$
CAA	$7.78e-11$	$7.40e-11$	$7.36e-11$	$8.87e-11$
CSW	$2.17e-10$	$2.22e-10$	$2.25e-10$	$8.97e-11$

Tolérance $1e^{-10}$

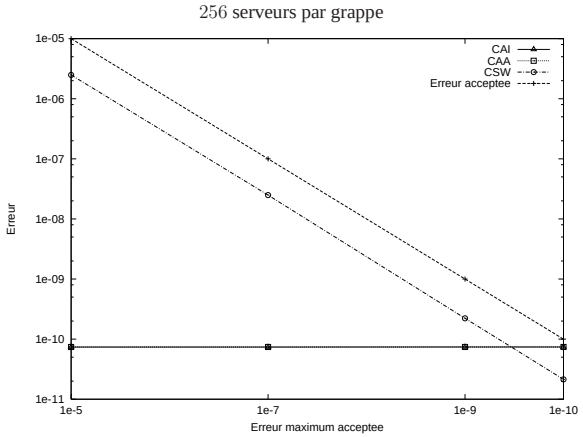


Nombre de serveurs	32	256	512	1024
CAI	$7.70e-11$	$7.39e-11$	$7.36e-11$	$4.33e-08$
CAA	$7.78e-11$	$7.41e-11$	$7.36e-11$	$8.94e-11$
CSW	$2.06e-11$	$2.14e-11$	$2.09e-11$	$1.03e-11$

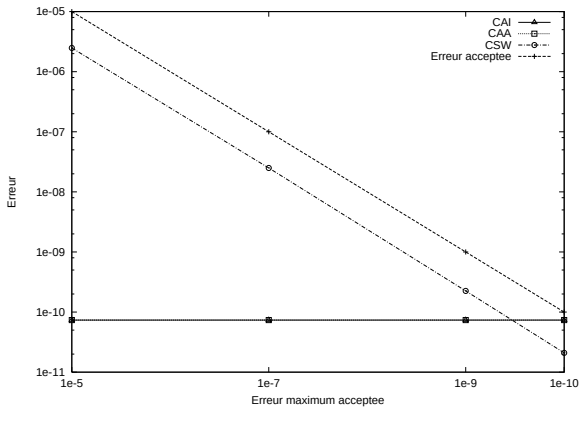
FIG. A.25 – WORKSTATION (30% des serveurs disponibles) - Précision des résultats pour différents nombres de serveurs



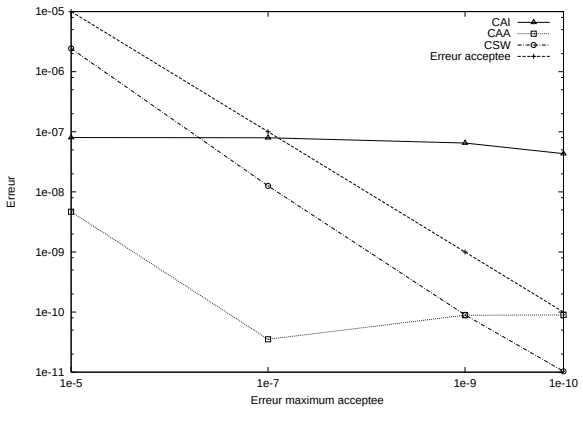
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$5.53e-09$	$3.32e-10$	$7.03e-11$	$7.70e-11$
CAA	$5.91e-11$	$7.76e-11$	$7.78e-11$	$7.78e-11$
CSW	$2.49e-06$	$2.45e-08$	$2.17e-10$	$2.06e-11$



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$7.37e-11$	$7.37e-11$	$7.39e-11$	$7.39e-11$
CAA	$7.39e-11$	$7.39e-11$	$7.40e-11$	$7.41e-11$
CSW	$2.49e-06$	$2.49e-08$	$2.22e-10$	$2.14e-11$



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$7.36e-11$	$7.36e-11$	$7.36e-11$	$7.36e-11$
CAA	$7.36e-11$	$7.36e-11$	$7.36e-11$	$7.36e-11$
CSW	$2.48e-06$	$2.50e-08$	$2.25e-10$	$2.09e-11$



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$8.02e-08$	$7.95-08$	$6.49e-08$	$4.33e-08$
CAA	$4.67e-09$	$3.53e-11$	$8.87e-11$	$8.94e-11$
CSW	$2.43e-06$	$1.26e-08$	$8.97e-11$	$1.03e-11$

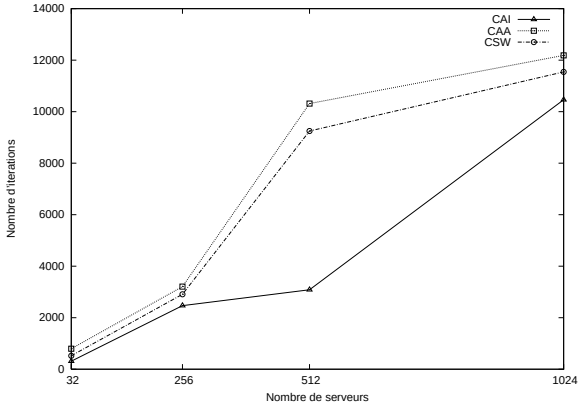
FIG. A.26 – WORKSTATION (30% des serveurs disponibles) - Précision des résultats pour différentes erreurs maximum acceptées

A.6.3 WORKSTATION- 50% de serveurs disponibles

Nombre d'itérations

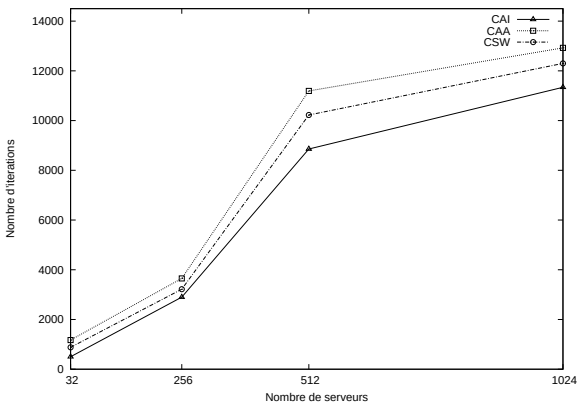
FIG. A.27 présente les mesures relevées pour les différents nombres de serveurs pour un niveau de disponibilité de 50%, alors que la FIG. A.28 présente les mesures organisées selon l'erreur maximum acceptée.

Tolérance $1e^{-5}$



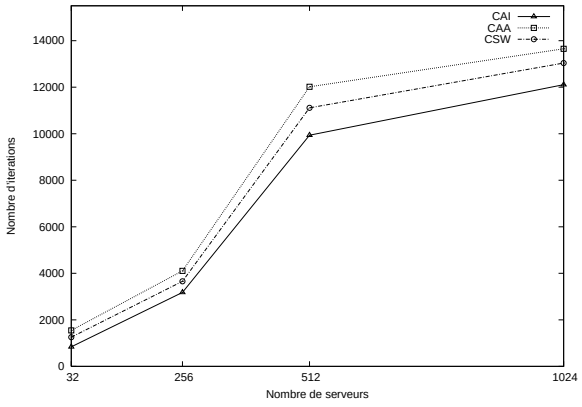
Nombre de serveurs	32	256	512	1024
UC	102645	111664	121968	142569
CRI	98	136	408	374
CAI	311	2468	3085	10461
CAA	797	3209	10314	12185
CSW	521	2909	9249	11544

Tolérance $1e^{-7}$



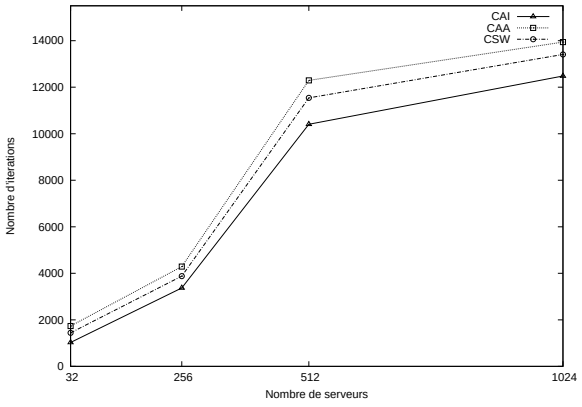
Nombre de serveurs	32	256	512	1024
UC	102944	111976	122294	142921
CRI	428	438	1088	7470
CAI	502	2899	8857	11336
CAA	1171	3652	11191	12924
CSW	879	3217	10223	12298

Tolérance $1e^{-9}$



Nombre de serveurs	32	256	512	1024
UC	103200	112244	122573	143223
CRI	802	886	1872	39796
CAI	840	3179	9938	12109
CAA	1546	4106	12019	13651
CSW	1254	3654	11112	13037

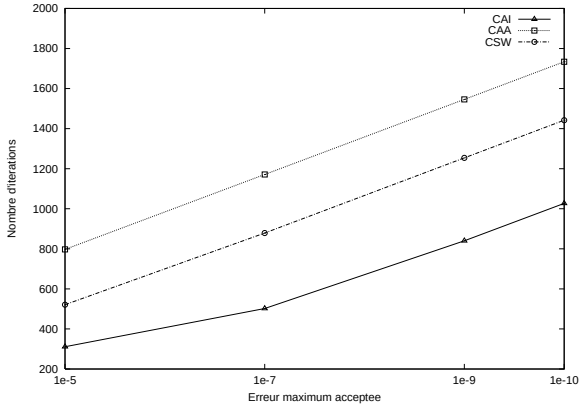
Tolérance $1e^{-10}$



Nombre de serveurs	32	256	512	1024
UC	103316	112364	122700	143360
CRI	990	1112	2278	56042
CAI	1027	3369	10408	12481
CAA	1734	4290	12295	13941
CSW	1442	3879	11543	13408

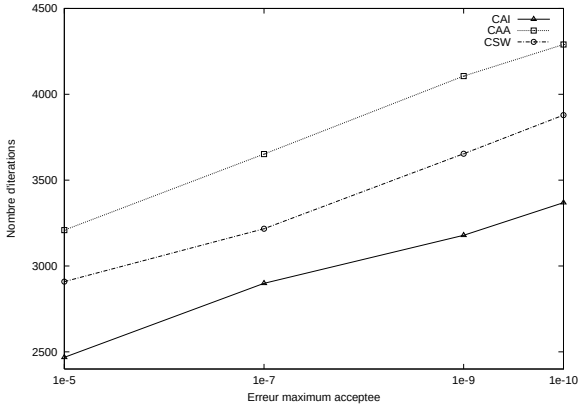
FIG. A.27 – WORKSTATION (50% des serveurs disponibles) - Nombre d'itérations pour différents nombres de serveurs

32 serveurs par grappe



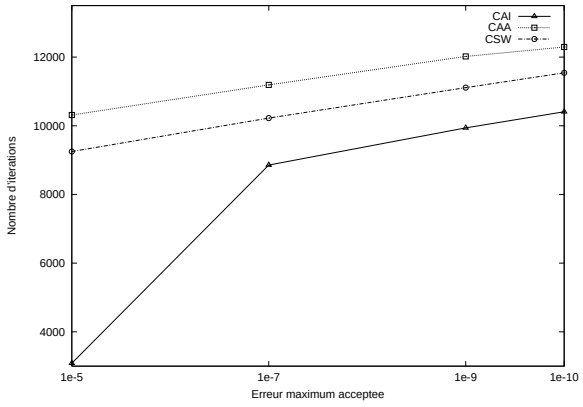
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	102645	102944	103200	103316
CRI	98	428	802	990
CAI	311	502	840	1027
CAA	797	1171	1546	1734
CSW	521	879	1254	1442

256 serveurs par grappe



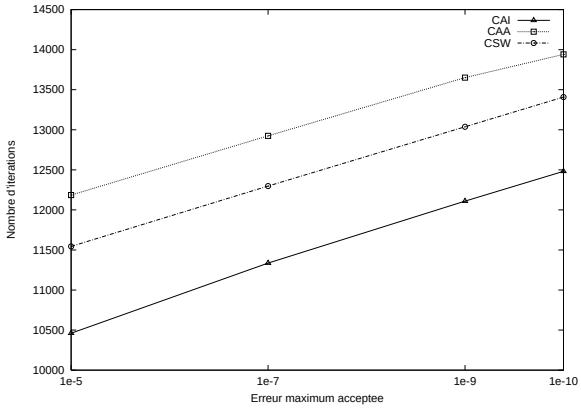
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	111664	111976	112244	112364
CRI	136	438	886	1112
CAI	2468	2899	3179	3369
CAA	3209	3652	4106	4290
CSW	2909	3217	3654	3879

512 serveurs par grappe



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	121968	122294	122573	122700
CRI	408	1088	1872	2278
CAI	3085	8857	9938	10408
CAA	10314	11191	12019	12295
CSW	9249	10223	11112	11543

1024 serveurs par grappe



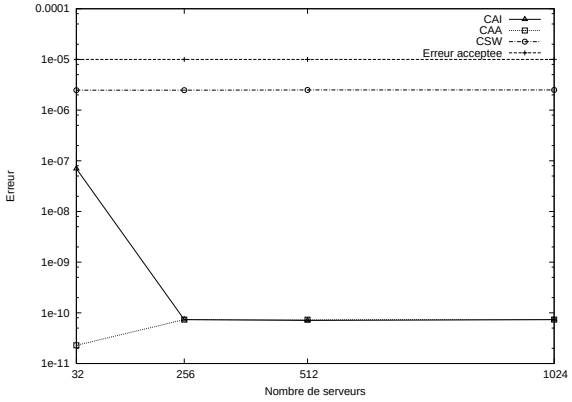
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	142569	142921	143223	143360
CRI	374	7470	39796	56042
CAI	10641	11336	12109	12481
CAA	12185	12924	13651	13941
CSW	11544	12298	13037	13408

FIG. A.28 – WORKSTATION (50% des serveurs disponibles) - Nombre d'itérations pour différentes erreurs maximum acceptées

Précision des résultats

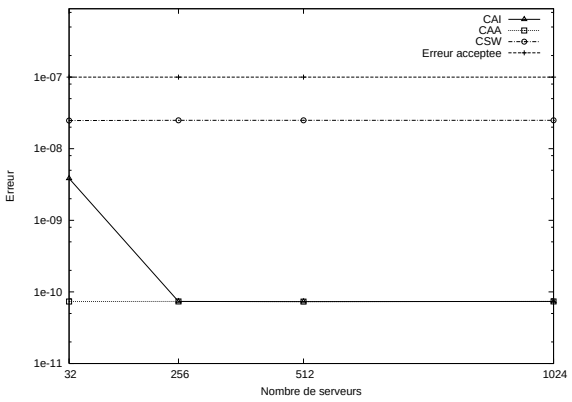
FIG. A.29 présente les mesures relevées pour différents nombres de serveurs pour un niveau de disponibilité de 50%, alors que la FIG. A.30 présente les mesures organisées selon l'erreur maximum acceptée.

Tolérance $1e^{-5}$



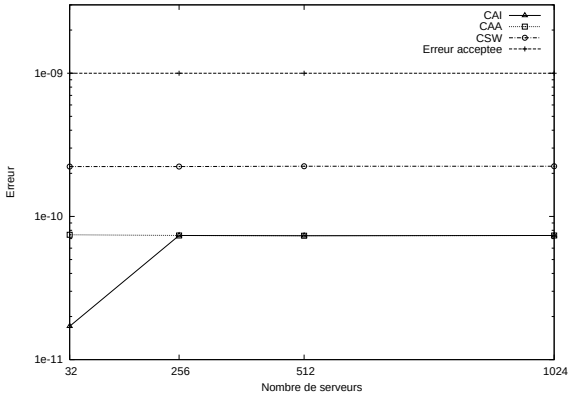
Nombre de serveurs	32	256	512	1024
CAI	$6.99e-08$	$7.36e-11$	$7.10e-11$	$7.36e-11$
CAA	$2.29e-11$	$7.36e-11$	$7.33e-11$	$7.36e-11$
CSW	$2.47e-06$	$2.47e-06$	$2.50e-06$	$2.50e-06$

Tolérance $1e^{-7}$



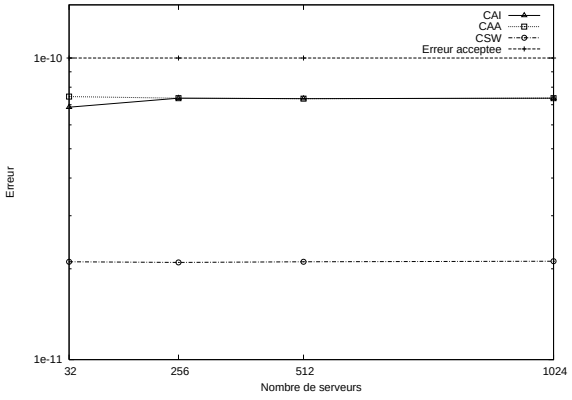
Nombre de serveurs	32	256	512	1024
CAI	$3.85e-09$	$7.36e-11$	$7.33e-11$	$7.36e-11$
CAA	$7.35e-11$	$7.36e-11$	$7.33e-11$	$7.36e-11$
CSW	$2.47e-08$	$2.49e-08$	$2.49e-08$	$2.49e-08$

Tolérance $1e^{-9}$



Nombre de serveurs	32	256	512	1024
CAI	$1.71e-11$	$7.36e-11$	$7.33e-11$	$7.36e-11$
CAA	$7.45e-11$	$7.36e-11$	$7.33e-11$	$7.36e-11$
CSW	$2.23e-10$	$2.23e-10$	$2.24e-10$	$2.24e-10$

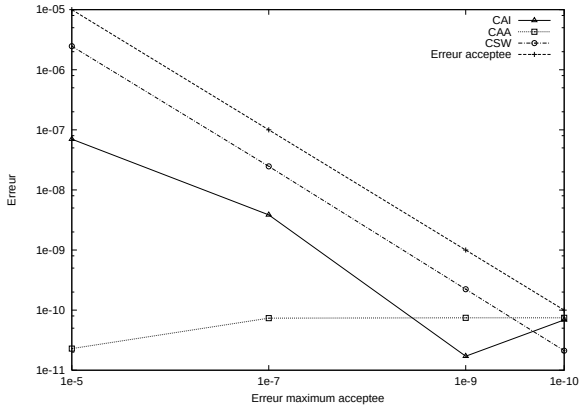
Tolérance $1e^{-10}$



Nombre de serveurs	32	256	512	1024
CAI	$6.87e-11$	$7.36e-11$	$7.33e-11$	$7.36e-11$
CAA	$7.45e-11$	$7.36e-11$	$7.33e-11$	$7.36e-11$
CSW	$2.11e-11$	$2.10e-11$	$2.11e-11$	$2.12e-11$

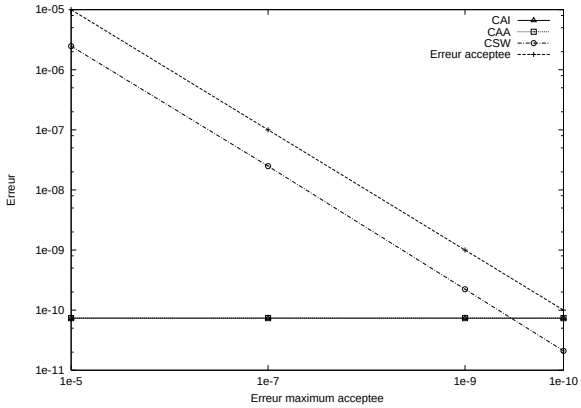
FIG. A.29 – WORKSTATION (50% des serveurs disponibles) - Précision des résultats pour différents nombres de serveurs

32 serveurs par grappe



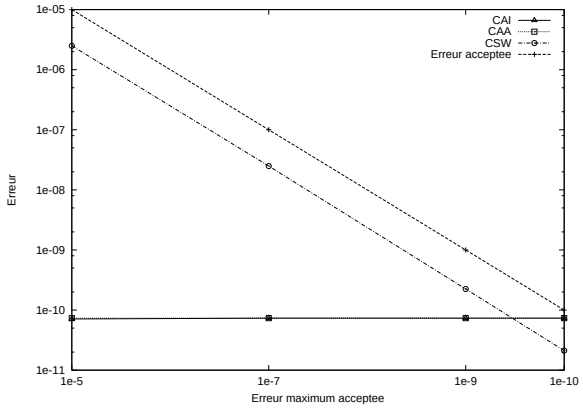
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$6.99e-08$	$3.85e-09$	$1.71e-11$	$6.87e-11$
CAA	$2.29e-11$	$7.35e-11$	$7.45e-11$	$7.45e-11$
CSW	$2.47e-06$	$2.47e-08$	$2.23e-10$	$2.11e-11$

256 serveurs par grappe



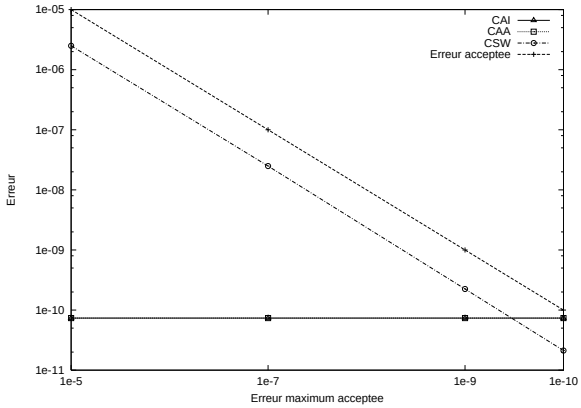
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$7.36e-11$	$7.36e-11$	$7.36e-11$	$7.36e-11$
CAA	$7.36e-11$	$7.36e-11$	$7.36e-11$	$7.36e-11$
CSW	$2.47e-06$	$2.49e-08$	$2.23e-10$	$2.10e-11$

512 serveurs par grappe



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$7.10e-11$	$7.33e-11$	$7.33e-11$	$7.33e-11$
CAA	$7.33e-11$	$7.33e-11$	$7.33e-11$	$7.33e-11$
CSW	$2.50e-06$	$2.49e-08$	$2.24e-10$	$2.11e-11$

1024 serveurs par grappe



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$7.36e-11$	$7.36e-11$	$7.36e-11$	$7.36e-11$
CAA	$7.36e-11$	$7.36e-11$	$7.36e-11$	$7.36e-11$
CSW	$2.50e-06$	$2.49e-08$	$2.24e-10$	$2.12e-11$

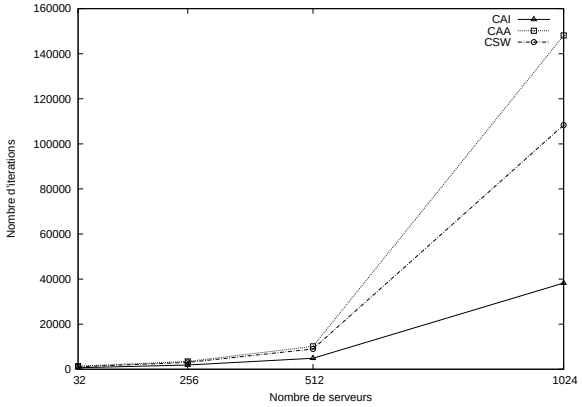
FIG. A.30 – WORKSTATION (50% des serveurs disponibles) - Précision des résultats pour différentes erreurs maximum acceptées

A.6.4 WORKSTATION- 70% de serveurs disponibles

Nombre d'itérations

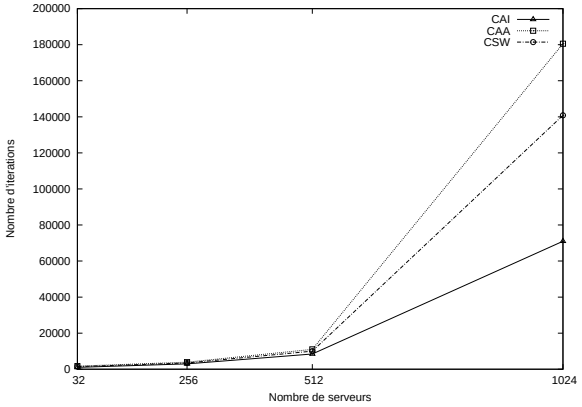
FIG. A.31 présente les mesures relevées pour les différents nombres de serveurs pour un niveau de disponibilité de 70%, alors que la FIG. A.32 présente les mesures organisées selon l'erreur maximum acceptée.

Tolérance $1e^{-5}$



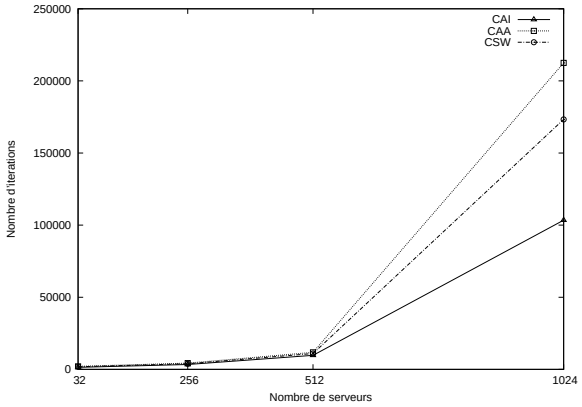
Nombre de serveurs	32	256	512	1024
UC	102645	111664	121968	142569
CRI	98	136	408	374
CAI	648	1861	4838	38222
CAA	1354	3443	10130	148091
CSW	1062	3005	8988	108373

Tolérance $1e^{-7}$



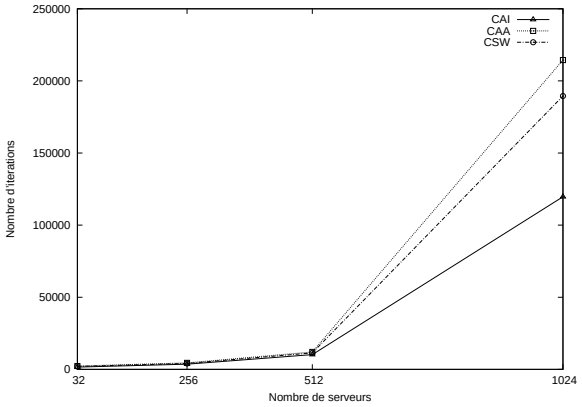
Nombre de serveurs	32	256	512	1024
UC	102944	111976	122294	142921
CRI	428	438	1088	7470
CAI	1023	2950	8468	70972
CAA	1729	3893	11025	180540
CSW	1437	3445	10030	140829

Tolérance $1e^{-9}$



Nombre de serveurs	32	256	512	1024
UC	103200	112244	122573	143223
CRI	802	886	1872	39796
CAI	1398	3380	9720	103429
CAA	2104	4333	11808	212499
CSW	1812	3895	10944	173286

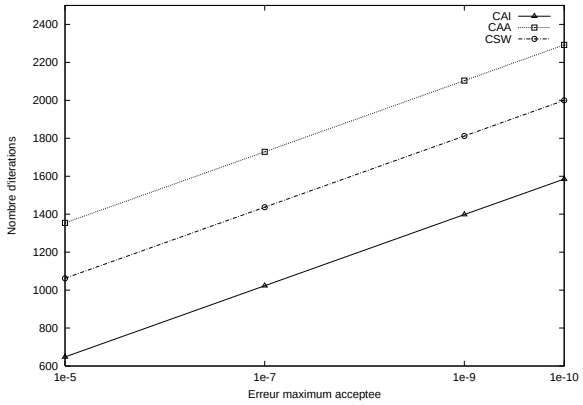
Tolérance $1e^{-10}$



Nombre de serveurs	32	256	512	1024
UC	103316	112364	122700	143360
CRI	990	1112	2278	56042
CAI	1585	3605	10215	119658
CAA	2292	4509	12069	214458
CSW	2000	4120	11382	189524

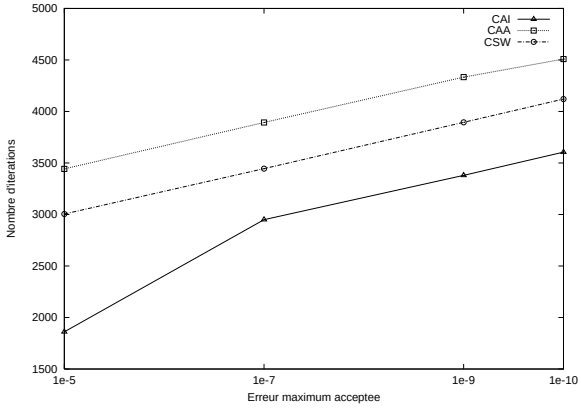
FIG. A.31 – WORKSTATION (70% des serveurs disponibles) - Nombre d'itérations pour différents nombres de serveurs

32 serveurs par grappe



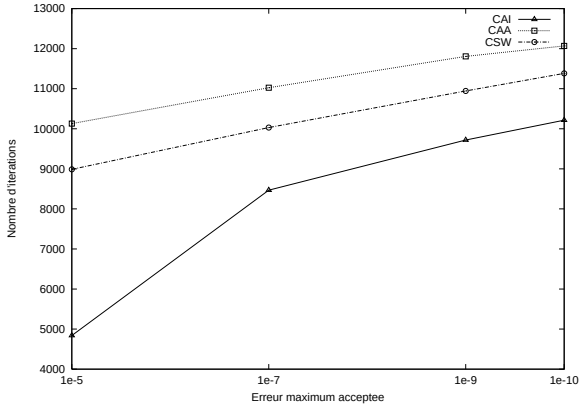
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	102645	102944	103200	103316
CRI	98	428	802	990
CAI	648	1023	1398	1585
CAA	1354	1729	2104	2292
CSW	1062	1437	1812	2000

256 serveurs par grappe



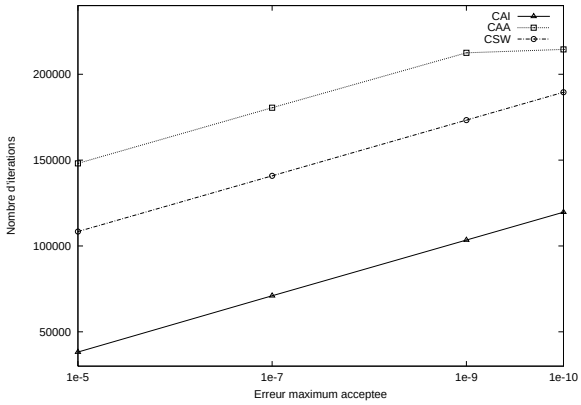
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	111664	111976	112244	112364
CRI	136	438	886	1112
CAI	1861	2950	3380	3605
CAA	3443	3893	4333	4509
CSW	3005	3445	3895	4120

512 serveurs par grappe



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	121968	122294	122573	122700
CRI	408	1088	1872	2278
CAI	4838	8468	9720	10215
CAA	10130	11025	11808	12069
CSW	8988	10030	10944	11382

1024 serveurs par grappe



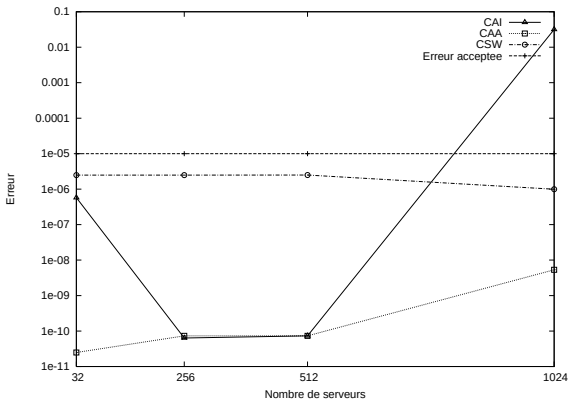
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	142569	142921	143223	143360
CRI	374	7470	39796	56042
CAI	38222	70972	103429	119658
CAA	148091	180540	212499	214458
CSW	108373	140829	173286	189524

FIG. A.32 – WORKSTATION (70% des serveurs disponibles) - Nombre d'itérations pour différentes erreurs maximum acceptées

Précision des résultats

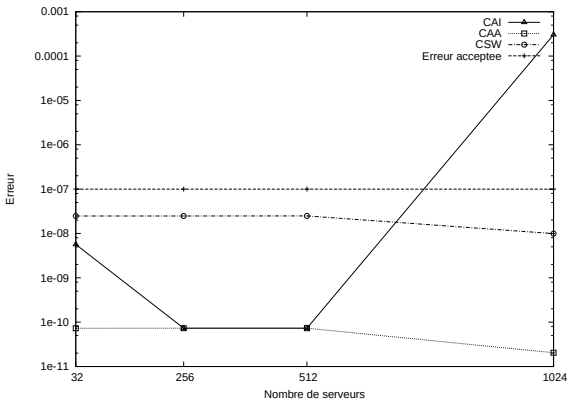
FIG. A.33 présente les mesures relevées pour différents nombres de serveurs pour un niveau de disponibilité de 70%, alors que la FIG. A.34 présente les mesures organisées selon l'erreur maximum acceptée.

Tolérance $1e^{-5}$



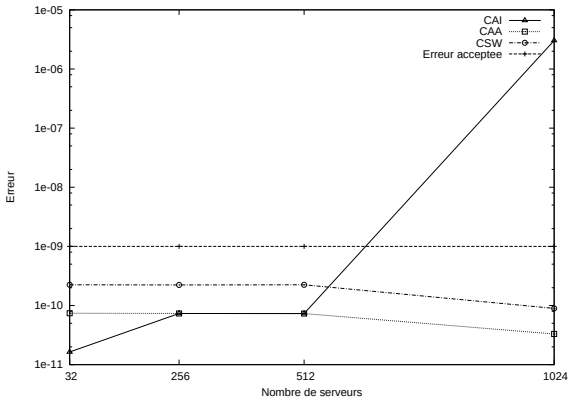
Nombre de serveurs	32	256	512	1024
CAI	$5.74e-07$	$6.37e-11$	$7.31e-11$	$3.12e-02$
CAA	$2.48e-11$	$7.33e-11$	$7.31e-11$	$5.34e-09$
CSW	$2.48e-06$	$2.48e-06$	$2.50e-06$	$9.97e-07$

Tolérance $1e^{-7}$



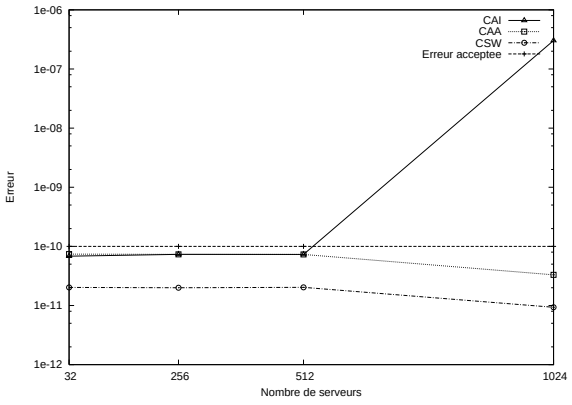
Nombre de serveurs	32	256	512	1024
CAI	$5.67e-09$	$7.33e-11$	$7.31e-11$	$3.04e-04$
CAA	$7.30e-11$	$7.33e-11$	$7.31e-11$	$2.05e-11$
CSW	$2.48e-08$	$2.48e-08$	$2.49e-08$	$9.94e-09$

Tolérance $1e^{-9}$



Nombre de serveurs	32	256	512	1024
CAI	$1.64e-11$	$7.33e-11$	$7.31e-11$	$3.03e-06$
CAA	$7.40e-11$	$7.33e-11$	$7.31e-11$	$3.30e-11$
CSW	$2.24e-10$	$2.23e-10$	$2.24e-10$	$8.94e-11$

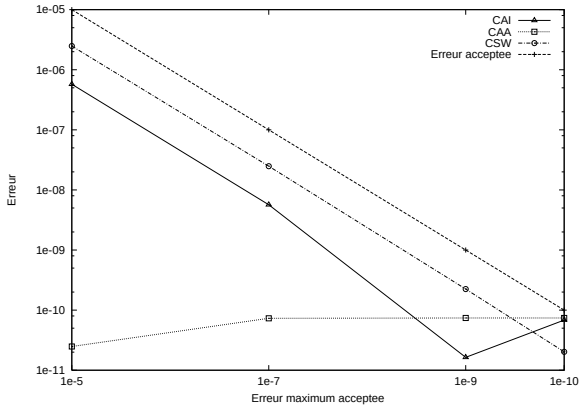
Tolérance $1e^{-10}$



Nombre de serveurs	32	256	512	1024
CAI	$6.82e-11$	$7.33e-11$	$7.31e-11$	$3.03e-07$
CAA	$7.40e-11$	$7.33e-11$	$7.31e-11$	$3.30e-11$
CSW	$2.02e-11$	$2.00e-11$	$2.03e-11$	$9.34e-12$

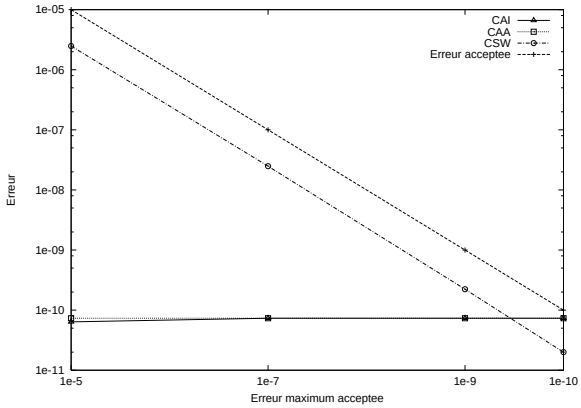
FIG. A.33 – WORKSTATION (70% des serveurs disponibles) - Précision des résultats pour différents nombres de serveurs

32 serveurs par grappe



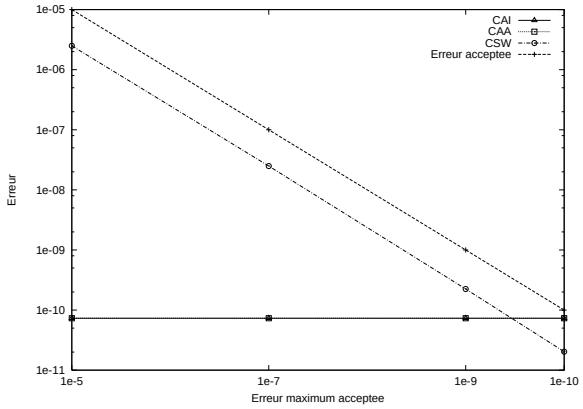
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$5.74e-07$	$5.67e-09$	$1.64e-11$	$6.82e-11$
CAA	$2.48e-11$	$7.30e-11$	$7.40e-11$	$7.40e-11$
CSW	$2.48e-06$	$2.48e-08$	$2.24e-10$	$2.02e-11$

256 serveurs par grappe



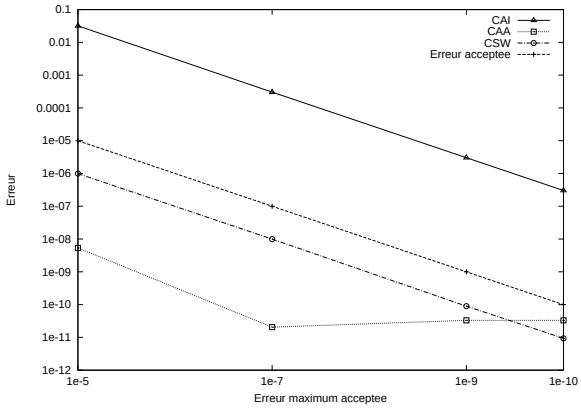
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$6.37e-11$	$7.33e-11$	$7.33e-11$	$7.33e-11$
CAA	$7.33e-11$	$7.33e-11$	$7.33e-11$	$7.33e-11$
CSW	$2.48e-06$	$2.48e-08$	$2.23e-10$	$2.00e-11$

512 serveurs par grappe



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$7.31e-11$	$7.31e-11$	$7.31e-11$	$7.31e-11$
CAA	$7.31e-11$	$7.31e-11$	$7.31e-11$	$7.31e-11$
CSW	$2.50e-06$	$2.49e-08$	$2.24e-10$	$2.03e-11$

1024 serveurs par grappe



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$3.12e-02$	$3.04e-04$	$3.03e-06$	$3.03e-07$
CAA	$5.34e-09$	$2.05e-11$	$3.30e-11$	$3.30e-11$
CSW	$9.97e-07$	$9.94e-09$	$8.94e-11$	$9.34e-12$

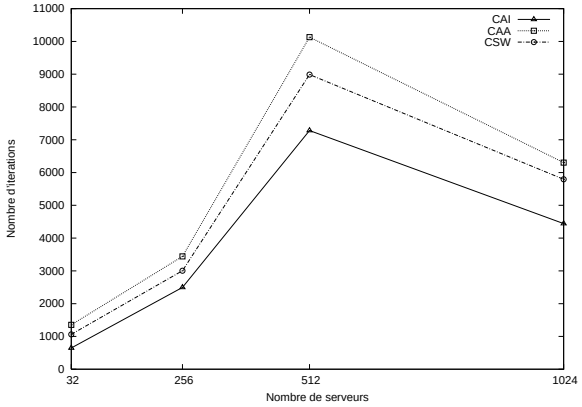
FIG. A.34 – WORKSTATION (70% des serveurs disponibles) - Précision des résultats pour différentes erreurs maximum acceptées

A.6.5 WORKSTATION- 90% de serveurs disponibles

Nombre d'itérations

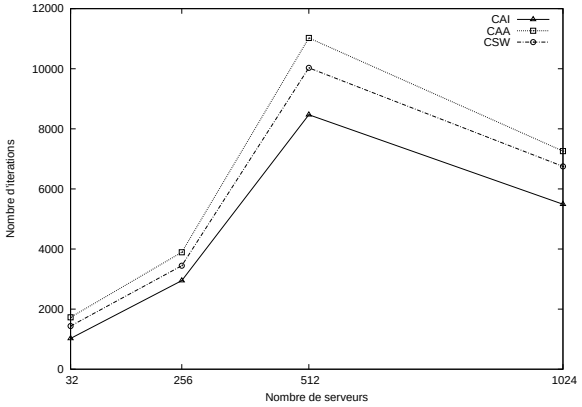
FIG. A.35 présente les mesures relevées pour les différents nombres de serveurs pour un niveau de disponibilité de 90%, alors que la FIG. A.36 présente les mesures organisées selon l'erreur maximum acceptée.

Tolérance $1e^{-5}$



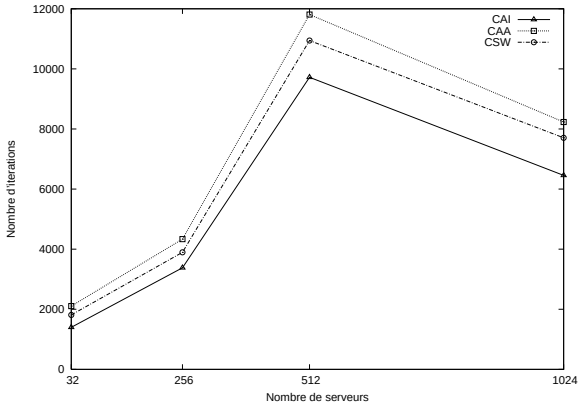
Nombre de serveurs	32	256	512	1024
UC	102645	111664	121968	142569
CRI	98	136	408	374
CAI	648	2495	7281	4442
CAA	1354	3443	10130	6303
CSW	1062	3005	5794	8988

Tolérance $1e^{-7}$



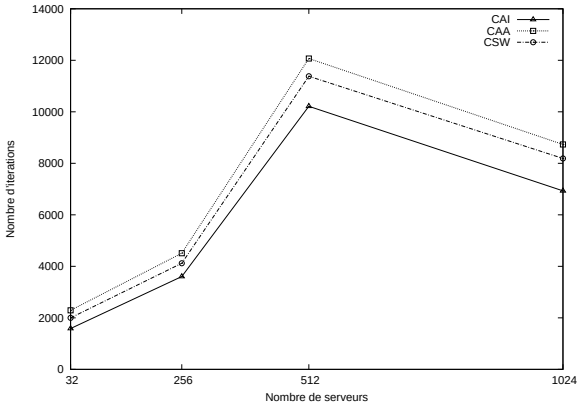
Nombre de serveurs	32	256	512	1024
UC	102944	111976	122294	142921
CRI	428	438	1088	7470
CAI	1023	2950	8469	5490
CAA	1729	3893	11025	7258
CSW	1437	3445	6749	10030

Tolérance $1e^{-9}$



Nombre de serveurs	32	256	512	1024
UC	103200	112244	122573	143223
CRI	802	886	1872	39796
CAI	1398	3380	9720	6456
CAA	2104	4333	11808	8233
CSW	1812	3895	7706	10944

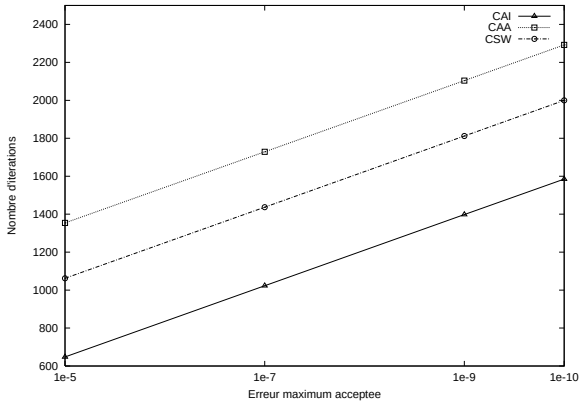
Tolérance $1e^{-10}$



Nombre de serveurs	32	256	512	1024
UC	103316	112364	122700	143360
CRI	990	1112	2278	56042
CAI	1585	3605	10215	6932
CAA	2292	4509	12069	8734
CSW	2000	4120	8190	11382

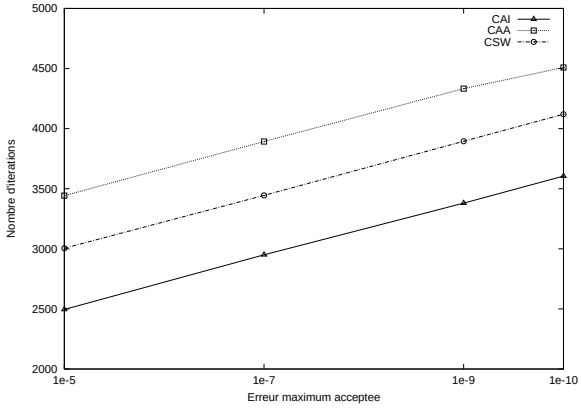
FIG. A.35 – WORKSTATION (90% des serveurs disponibles) - Nombre d'itérations pour différents nombres de serveurs

32 serveurs par grappe



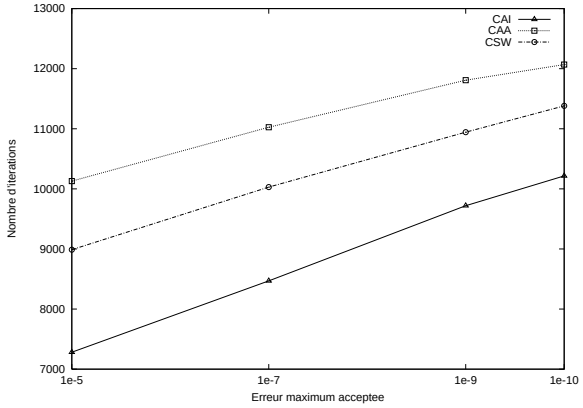
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	102645	102944	103200	103316
CRI	98	428	802	990
CAI	648	1023	1398	1585
CAA	1354	1729	2104	2292
CSW	1062	1437	1812	2000

256 serveurs par grappe



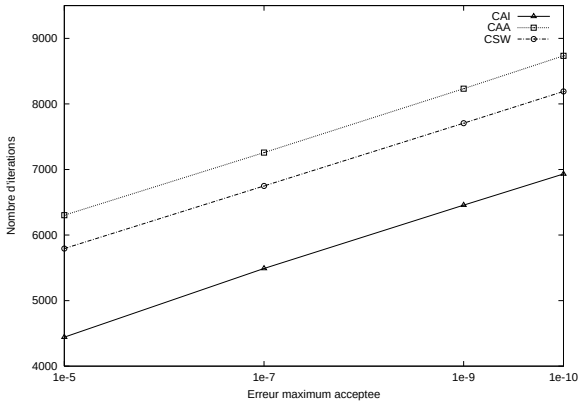
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	111664	111976	112244	112364
CRI	136	438	886	1112
CAI	2495	2950	3380	3605
CAA	3443	3893	4333	4509
CSW	3005	3445	3895	4120

512 serveurs par grappe



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	121968	122294	122573	122700
CRI	408	1088	1872	2278
CAI	7281	8469	9720	10215
CAA	10130	11025	12808	12069
CSW	5794	6749	7706	8190

1024 serveurs par grappe



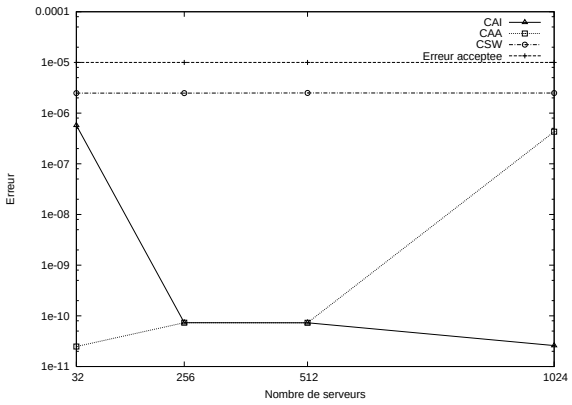
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
UC	142569	142921	143223	143360
CRI	374	7470	39796	56042
CAI	4442	5490	6456	6932
CAA	6303	7258	8233	8734
CSW	8988	10030	10944	11382

FIG. A.36 – WORKSTATION (90% des serveurs disponibles) - Nombre d'itérations pour différentes erreurs maximum acceptées

Précision des résultats

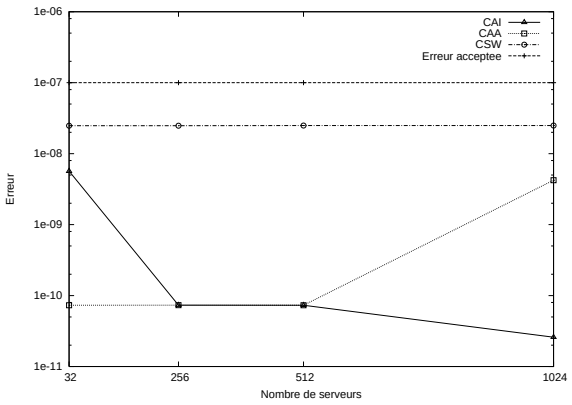
FIG. A.37 présente les mesures relevées pour différents nombres de serveurs pour un niveau de disponibilité de 90%, alors que la FIG. A.38 présente les mesures organisées selon l'erreur maximum acceptée.

Tolérance $1e^{-5}$



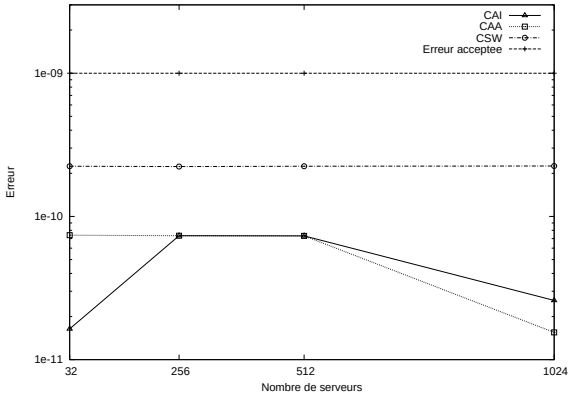
Nombre de serveurs	32	256	512	1024
CAI	$5.74e-07$	$7.33e-11$	$7.31e-11$	$2.59e-11$
CAA	$2.48e-11$	$7.33e-11$	$7.31e-11$	$4.30e-07$
CSW	$2.48e-06$	$2.48e-06$	$2.50e-06$	$2.49e-06$

Tolérance $1e^{-7}$



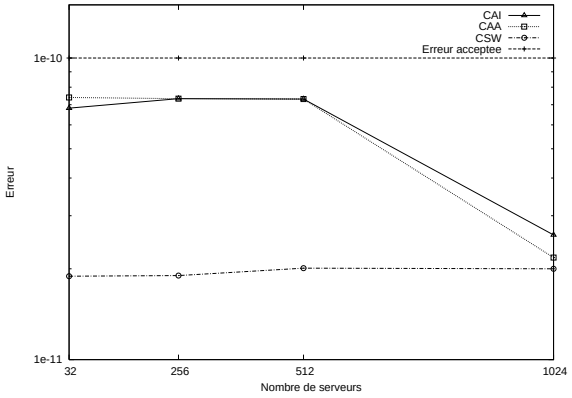
Nombre de serveurs	32	256	512	1024
CAI	$5.67e-09$	$7.33e-11$	$7.31e-11$	$2.59e-11$
CAA	$7.30e-11$	$7.33e-11$	$7.31e-11$	$4.23e-09$
CSW	$2.48e-08$	$2.48e-08$	$2.49e-08$	$2.49e-08$

Tolérance $1e^{-9}$



Nombre de serveurs	32	256	512	1024
CAI	$1.64e-11$	$7.33e-11$	$7.31e-11$	$2.59e-11$
CAA	$7.40e-11$	$7.33e-11$	$7.31e-11$	$1.55e-11$
CSW	$2.24e-10$	$2.23e-10$	$2.24e-10$	$2.25e-10$

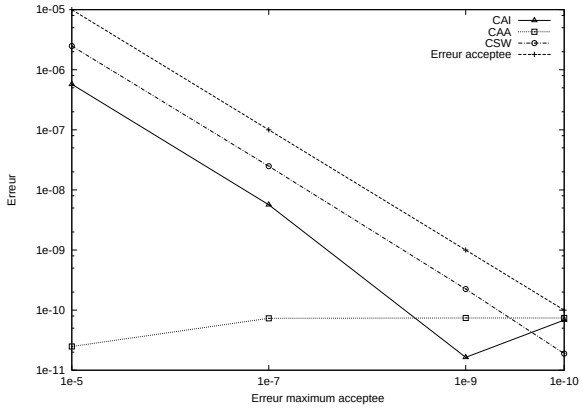
Tolérance $1e^{-10}$



Nombre de serveurs	32	256	512	1024
CAI	$6.82e-11$	$7.33e-11$	$7.31e-11$	$2.59e-11$
CAA	$7.40e-11$	$7.33e-11$	$7.31e-11$	$2.18e-11$
CSW	$1.89e-11$	$1.90e-11$	$2.01e-11$	$2.00e-11$

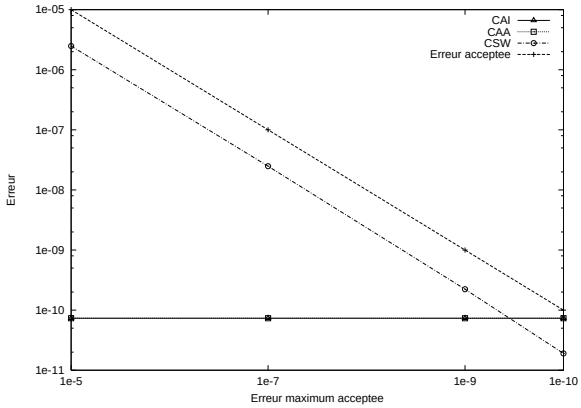
FIG. A.37 – WORKSTATION (90% des serveurs disponibles) - Précision des résultats pour différents nombres de serveurs

32 serveurs par grappe



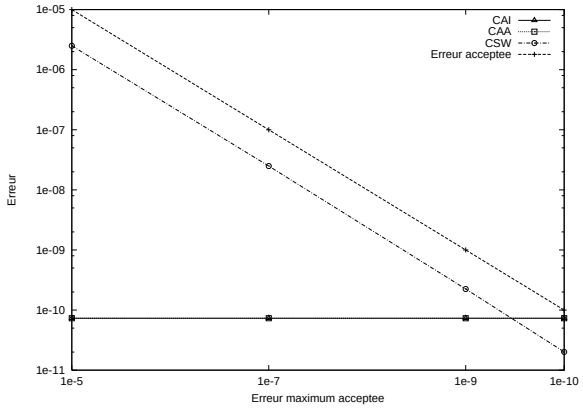
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$5.74e-07$	$5.67e-09$	$1.64e-11$	$6.82e-11$
CAA	$2.48e-11$	$7.30e-11$	$7.40e-11$	$7.40e-11$
CSW	$2.48e-06$	$2.48e-08$	$2.24e-10$	$1.89e-11$

256 serveurs par grappe



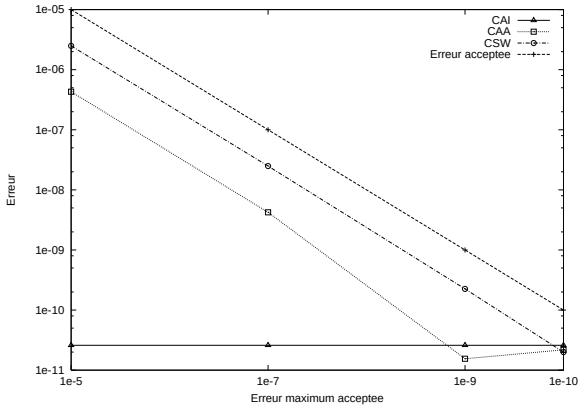
Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$7.33e-11$	$7.33e-11$	$7.33e-11$	$7.33e-11$
CAA	$7.33e-11$	$7.33e-11$	$7.33e-11$	$7.33e-11$
CSW	$2.48e-06$	$2.48e-08$	$2.23e-10$	$1.90e-11$

512 serveurs par grappe



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$7.31e-11$	$7.31e-11$	$7.31e-11$	$7.31e-11$
CAA	$7.31e-11$	$7.31e-11$	$7.31e-11$	$7.31e-11$
CSW	$2.50e-06$	$2.49e-08$	$2.24e-10$	$2.01e-11$

1024 serveurs par grappe



Tolérance	$1e-5$	$1e-7$	$1e-9$	$1e-10$
CAI	$2.59e-11$	$2.59e-11$	$2.59e-11$	$2.59e-11$
CAA	$4.30e-07$	$4.23e-09$	$1.55e-11$	$2.18e-11$
CSW	$2.49e-06$	$2.49e-08$	$2.25e-10$	$2.00e-11$

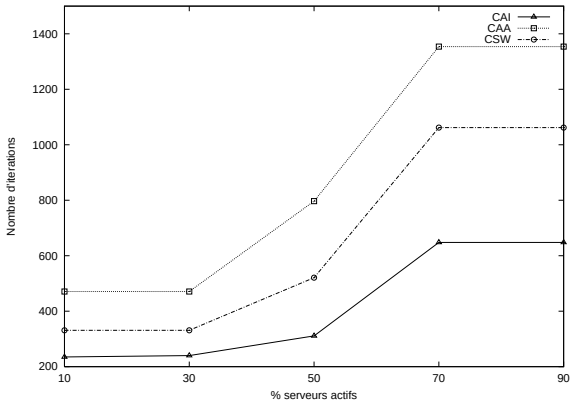
FIG. A.38 – WORKSTATION (90% des serveurs disponibles) - Précision des résultats pour différentes erreurs maximum acceptées

A.6.6 WORKSTATION avec 32 serveurs

Nombre d'itérations

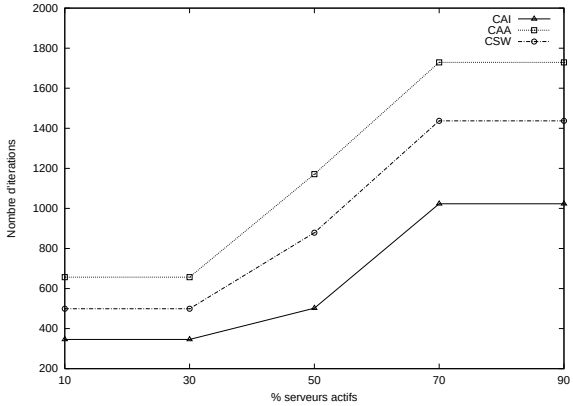
FIG. A.39 présente les mesures relevées pour différents niveaux de disponibilité pour 32 serveurs par grappe.

Tolérance $1e^{-5}$



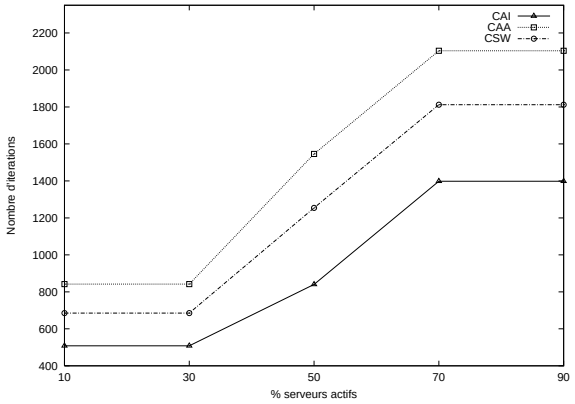
Serveurs actives	10	30	50	70	90
UC	102645	102645	102645	102645	102645
CRI	98	98	98	98	98
CAI	235	240	311	648	648
CAA	471	471	797	1354	1354
CSW	331	331	521	1062	1062

Tolérance $1e^{-7}$



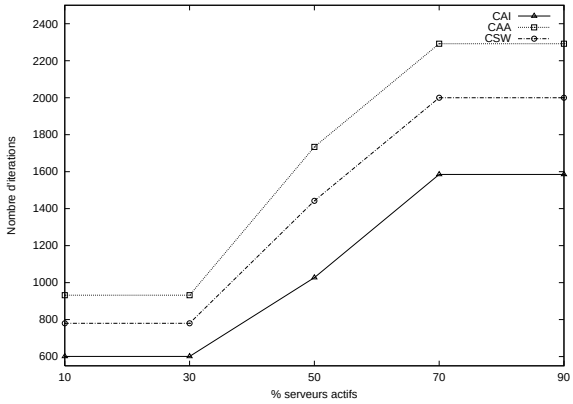
Serveurs actives	10	30	50	70	90
UC	102944	102944	102944	102944	102944
CRI	428	428	428	428	428
CAI	346	346	502	1023	1023
CAA	657	657	1171	1729	1729
CSW	499	499	879	1437	1437

Tolérance $1e^{-9}$



Serveurs actives	10	30	50	70	90
UC	103200	103200	103200	103200	103200
CRI	802	802	802	802	802
CAI	508	508	840	1398	1398
CAA	842	842	1546	2104	2104
CSW	685	685	1254	1812	1812

Tolérance $1e^{-10}$



Serveurs actives	10	30	50	70	90
UC	103316	103316	103316	103316	103316
CRI	990	990	990	990	990
CAI	601	601	1027	1585	1585
CAA	932	932	1734	2292	2292
CSW	780	780	1442	2000	2000

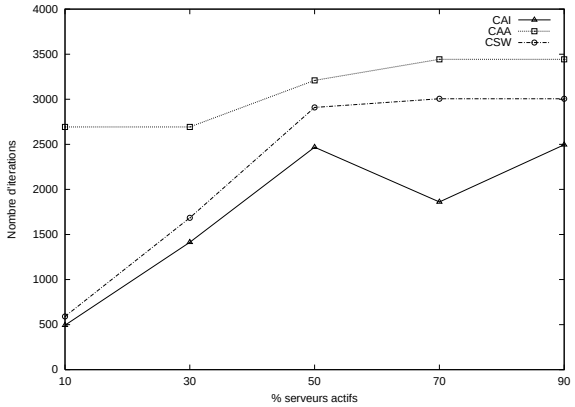
FIG. A.39 – Différents ensembles UP pour 32 serveurs

A.6.7 WORKSTATION **avec** 256 **serveurs**

Nombre d'itérations

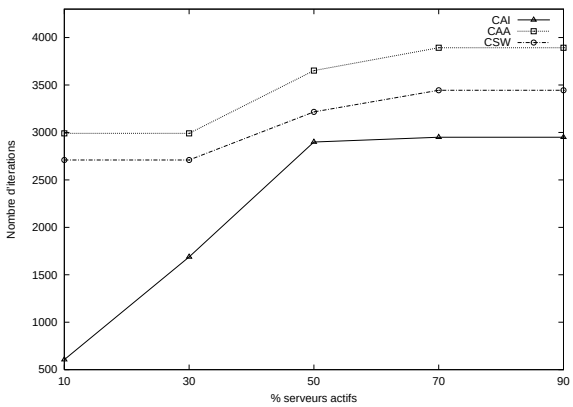
FIG. A.40 présente les mesures relevées pour différents niveaux de disponibilité pour 256 serveurs par grappe.

Tolérance $1e^{-5}$



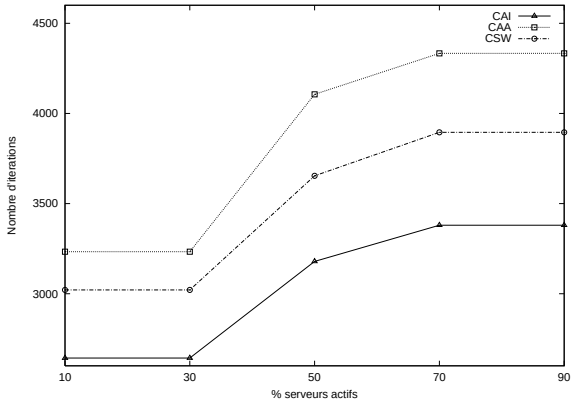
Serveurs actives	10	30	50	70	90
UC	111664	111664	111664	111664	111664
CRI	136	136	136	136	136
CAI	493	1413	2468	1861	2495
CAA	2693	2693	3209	3443	3443
CSW	590	1685	2909	3005	3005

Tolérance $1e^{-7}$



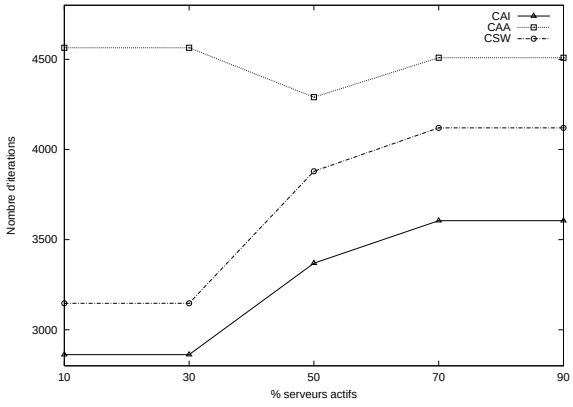
Serveurs actives	10	30	50	70	90
UC	111976	111976	111976	111976	111976
CRI	438	438	438	438	438
CAI	606	1688	2899	2950	2950
CAA	2991	2991	3652	3893	3893
CSW	2710	2710	3217	3445	3445

Tolérance $1e^{-9}$



Serveurs actives	10	30	50	70	90
UC	112244	112244	112244	112244	112244
CRI	886	886	886	886	886
CAI	2643	2643	3179	3380	3380
CAA	3233	3233	4106	4333	4333
CSW	3021	3021	3654	3895	3895

Tolérance $1e^{-10}$



Serveurs actives	10	30	50	70	90
UC	112364	112364	112364	112364	112364
CRI	1112	1112	1112	1112	1112
CAI	2862	2862	3369	3605	3605
CAA	4564	4564	4290	4509	4509
CSW	3147	3147	3879	4120	4120

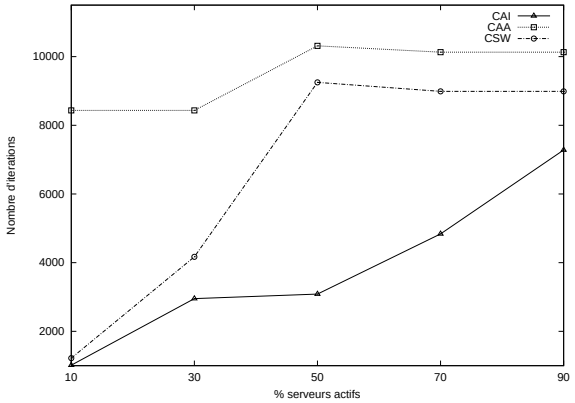
FIG. A.40 – Différents ensembles UP pour 256 serveurs

A.6.8 WORKSTATION avec 512 serveurs

Nombre d'itérations

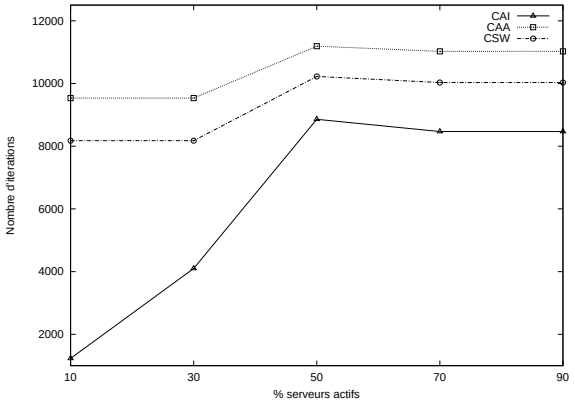
FIG. A.41 présente les mesures relevées pour différents niveaux de disponibilité pour 512 serveurs par grappe.

Tolérance $1e^{-5}$



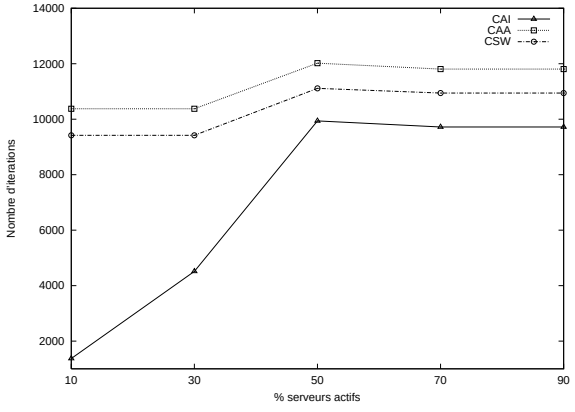
Serveurs actives	10	30	50	70	90
UC	121968	121968	121968	121968	121968
CRI	408	408	408	408	408
CAI	1015	2954	3085	4838	7281
CAA	8433	8433	10314	10130	10130
CSW	1220	4169	9249	8988	8988

Tolérance $1e^{-7}$



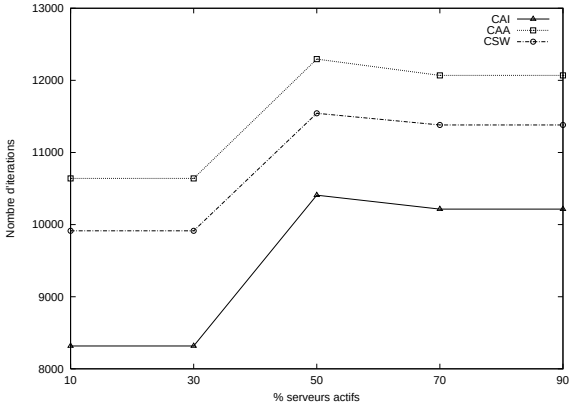
Serveurs actives	10	30	50	70	90
UC	122294	122294	122294	122294	122294
CRI	1088	1088	1088	1088	1088
CAI	1232	4099	8857	8468	8469
CAA	9534	9534	11191	11025	11025
CSW	8175	8175	10223	10030	10030

Tolérance $1e^{-9}$



Serveurs actives	10	30	50	70	90
UC	122573	122573	122573	122573	122573
CRI	1872	1872	1872	1872	1872
CAI	1372	4512	9938	9720	9720
CAA	10377	10377	12019	11808	11808
CSW	9418	9418	11112	10944	10944

Tolérance $1e^{-10}$



Serveurs actives	10	30	50	70	90
UC	122700	122700	122700	122700	122700
CRI	2278	2278	2278	2278	2278
CAI	8317	8317	10408	10215	10215
CAA	10640	10640	12295	12069	12069
CSW	9914	9914	11543	11382	11382

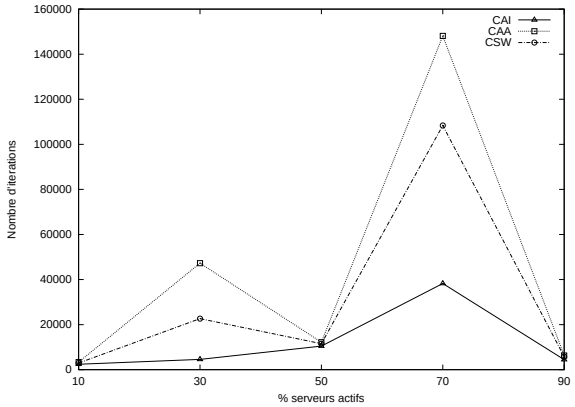
FIG. A.41 – Différents ensembles UP pour 512 serveurs

A.6.9 WORKSTATION **avec** 1024 **serveurs**

Nombre d'itérations

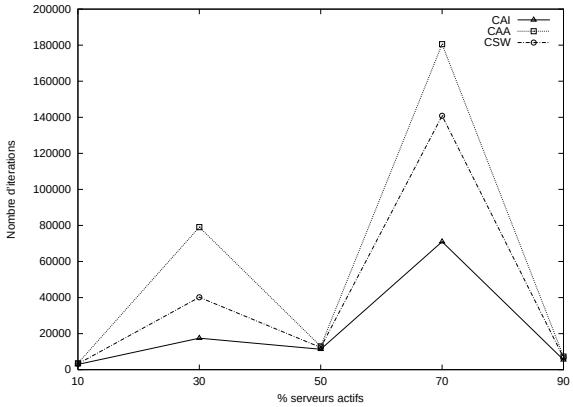
FIG. A.42 présente les mesures relevées pour différents niveaux de disponibilité pour 1024 serveurs par grappe.

Tolérance $1e^{-5}$



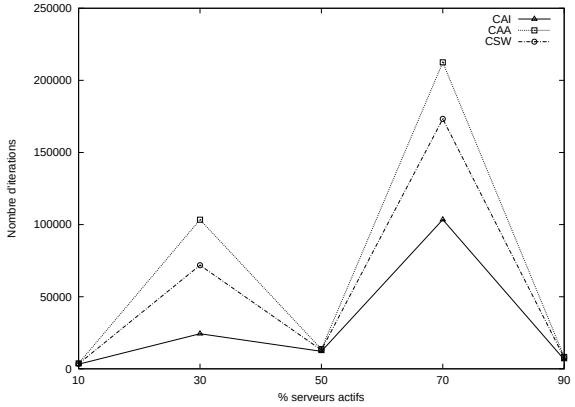
Serveurs actives	55	65	75	85	95
UC	142569	142569	142569	142569	142569
CRI	374	374	374	374	374
CAI	2422	4565	10461	38222	4442
CAA	3338	47254	12185	148091	6303
CSW	2949	22657	11544	108373	5794

Tolérance $1e^{-7}$



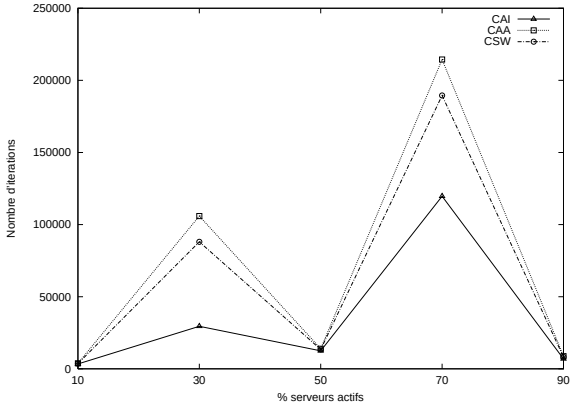
Serveurs actives	55	65	75	85	95
UC	142921	142921	142921	142921	142921
CRI	7470	7470	7470	7470	7470
CAI	2927	17494	11336	70972	5490
CAA	3566	79051	12924	180540	7258
CSW	3227	40158	12298	140829	6749

Tolérance $1e^{-9}$



Serveurs actives	55	65	75	85	95
UC	143223	143223	143223	143223	143223
CRI	39796	39796	39796	39796	39796
CAI	3222	24291	12109	103426	6456
CAA	3769	103408	13651	212499	8233
CSW	3465	71810	13037	173286	7706

Tolérance $1e^{-10}$



Serveurs actives	55	65	75	85	95
UC	143360	143360	143360	143360	143360
CRI	56042	56042	56042	56042	56042
CAI	3348	29479	12481	119658	6932
CAA	3842	105920	13941	214458	8734
CSW	3576	88048	13408	189524	8190

FIG. A.42 – Différents ensembles UP pour 1024 serveurs

Annexe B

Syntaxe de description d'un modèle SAN dans PEPS2007

Ce chapitre présente le format textuel pour la description d'un modèle SAN dans PEPS2007. La description d'un modèle SAN présente une structure divisée en 5 blocs : *identifiants*, *événements*, *fonction d'atteignabilité*, *réseau* et *résultats*.

FIG. B.1 présente la structure générale des blocs qui composent le format textuel d'un modèle SAN dans PEPS2007.

Le début de chaque bloc est délimité par des mots-clés en **gras** qui font partie des mots réservés du langage de description de PEPS2007. Les autres mots réservés du langage sont en *italique*. Les mots entre les symboles "<" et ">" sont des informations qui doivent être fournies par l'utilisateur et les symboles "{" et "}" indiquent des informations facultatives.

B.1 Bloc des identifiants

Le bloc des identifiants, délimité par le mot-clé "**identifiers**" contient toutes les déclarations de paramètres qui seront utilisées dans la description du modèle. Chaque paramètre intervenant dans la description du modèle doit avoir un identifiant et une valeur associée. Un identifiant (fnc_name ou dom_name) peut être n'importe quelle chaîne de caractères alpha-numériques. La valeur associée à l'identifiant peut être une simple valeur numérique, une fonction ou un ensemble d'indice (domaine). On va utiliser la notation "fnc_name" lorsqu'on fait référence à une fonctions et "dom_name" pour référencer un domaine. Les fonctions sont des expressions similaires à des expressions mathématiques communes, avec des opérateurs logiques et arithmétiques. Les valeurs numériques, fonctions et domaines sont définis selon une syntaxe ressemblant à celle du langage C. Les arguments de ces expressions peuvent être des nombres constants (paramètres du modèle), mais aussi des indices d'automates ou des états. Dans ce dernier cas, les expressions sont des fonctions définies sur l'espace d'états du modèle SAN.

identifiers
< fnc_name > {[fnc_rep]} = < exp > **;**
< dom_name > = **[** < domain >**] ;**

events
// without réplication
loc < evt_name > (< rate >)
syn < evt_name > (< rate >)
// with réplication
loc < evt_name > {[réplication_domain]} (< rate >)
syn < evt_name > {[réplication_domain]} (< rate >)

{*partial*} **reachability** = < exp > **;**

network < net_name > (< type >)
aut < aut_name > {[réplication_domain]}
stt < stt_name > {[réplication_domain]} {(reward)}
to(< stt_name > {[réplication_domain]} {/to_cond} *)*
< evt_name > {[réplication_domain]} {(< prob >)} {/evt_cond}
. . .
< evt_name > {[réplication_domain]} {(< prob >)} {/evt_cond}
. . .
stt < stt_name > {[réplication_domain]} {(reward)}
to(< stt_name > {[réplication_domain]} {/to_cond} *)*
< evt_name > {[réplication_domain]} {(< prob >)} {/evt_cond}
. . .
aut < aut_name > {[réplication_domain]}
. . .

results
< res_name > = < exp > **;**

FIG. B.1 – Structure modulaire du format textuel d'un modèle SAN.

Par exemple, "Le nombre d'automates dans l'état $n0$" (qui donne une valeur entière) peut être exprimé par "$nb\ n0$". Une fonction qui rend la valeur 4 si deux automates ($A1$ et $A2$) sont dans des états différents et la valeur 0 dans le cas contraire, est exprimée par "$(st\ A1\ != st\ A2)\ *\ 4$". Les opérateurs de comparaison rendent la valeur "1" pour un résultat vrai et la valeur "0" pour un résultat faux. Le format d'une expression est décrit dans la Section B.6.

syntaxe du bloc **identifiers**

identifiers
< fnc_name > {[fnc_rep]} = < exp > **;**
. . .
< dom_name > = **[** < domain >**] ;**
. . .

– "fnc_name" est un identifiant de fonction. Un identifiant de fonction est une chaîne de caractères qui commence par une lettre et est suivi d'une suite de lettres ou de chiffres ;

- "dom_name" est un identifiant d'un ensemble d'indices (domaine). Un identifiant de domaine est construit de façon identique à un identifiant de fonction.
- "fnc_rep" est un domaine de réplication. Une fonction peut être répliquée à l'aide de fonction de description (Section B.6.1) pour créer des fonctions du même type ayant comme argument l'indice des automates ou bien des états. Ceci va induire des évaluation différentes selon l'automate et l'état de départ de la transition sur laquelle elle figure.
- "exp" peut être un nombre constant ou une expression mathématique. PEPS2007 fournit un ensemble d'opérateurs logiques et arithmétiques ordinaires, ainsi que des opérateurs spécifiques pour la description et l'évaluation du modèle. Les expressions mathématiques sont décrites en détail dans la Section B.6.
- "domain" définit un ensemble d'indices qui peut être un intervalle continu "[1..3]", une liste "[1,2,3]" ou une liste d'intervalles "[1..3,5,7..9]". On peut utiliser des identifiants pour définir un intervalle, par exemple, "[1..ID1, 5, 7..ID2]", ou ID1 et ID2 sont des identifiant de fonctions avec des valeurs constantes. Les valeurs de ID1 et ID2 sont des paramètres du modèle. Une seule contrainte est imposée, les indices du domaine doivent être spécifié dans un ordre croissant de valeurs.

Les domaines sont utilisés dans les réplications d'automates, d'états et d'événements. Par exemple, un groupe d'automates répliqués A sur le domaine $[0..2, 5, 8..10]$ contient les automates $A[0]$, $A[1]$, $A[2]$, $A[5]$, $A[8]$, $A[9]$, et $A[10]$.

B.2 Bloc des événements

De la même façon que le bloc des identifiant, le bloc des événements, délimité par "**events**" contient l'ensemble des événements qui seront associés aux transitions du modèle. Chaque événement déclaré doit préciser :

- son type (local (loc) ou synchronisant (syn)) ;
- son identifiant ;
- son taux de franchissement.

Le bloc **events** a la syntaxe suivante :

Syntaxe du bloc **events**

events
loc < evt_name > {[evt_rep]} (< rate >)
syn < evt_name > {[evt_rep]} (< rate >)
...

- "*loc*" définit le type de l'événement comme un événement local ;
- "*syn*" définit le type de l'événement comme un événement synchronisant ;
- "evt_name" est l'identifiant de l'événement. Un identifiant d'événement est une chaîne de caractères qui commence par une lettre et est suivi d'une suite de lettres ou chiffres ;

– "evt_rep" est le domaine de réplication de l'événement. Le evt_rep doit être un identifiant de domaine défini dans le bloc **identifiers**. Un événement peut avoir jusqu'à trois niveaux de réplication. Chaque niveau est défini par $[evt\_rep]$. Par exemple, un événement répliqué à 2 niveaux a la forme "< evt_name > [evt_rep][evt_rep]" ;
– "rate" définit le taux de franchissement de l'événement. Le taux de franchissement doit être un identifiant de fonction défini dans le bloc **identifiers**. Dans le cas de réplication d'événements, une liste de taux de franchissement peut être fournie. Si un seul taux de franchissement est fourni pour des événements répliqués, tous les événements auront le même taux.

B.3 Bloc d'atteignabilité

Le mot-clé **reachability** délimite le bloc d'atteignabilité. Ce bloc définit l'espace d'états atteignables du modèle SAN. L'espace d'états atteignables est défini par une fonction évaluée sur l'espace d'états produit. Cette fonction renvoie une valeur non nulle pour les états de $\mathcal{S}$, et la valeur 0 pour les états de $\hat{\mathcal{S}} - \mathcal{S}$. On a aussi la possibilité de définir uniquement un sous-ensemble de l'espace d'états atteignables $\mathcal{S}$ en rajoutant le mot-clé "*partial*", l'ensemble des états atteignables est ensuite généré par PEPS.

La syntaxe du bloc **reachability** est :

Syntaxe du bloc **reachability**

{*partial*} **reachability** = < exp > **;**

B.4 Bloc du réseau

Délimité par le mot-clé **network**, le bloc du réseau est le composant principal de la description du modèle SAN. La description du modèle dans le bloc **network** est faite de façon hiérarchique (FIG. B.4). C'est-à-dire, un réseau d'automates stochastiques est constitué d'un ensemble d'automates ; chaque automate est constitué d'un ensemble d'états ; chaque état a un ensemble de transitions sortantes ; chaque transition a un ensemble d'événements associés.

Le premier niveau de cette structure hiérarchique contient des informations générales sur le modèle, tels que le nom du modèle et le type d'échelle de temps du modèle. La syntaxe employée dans ce bloc est :

Syntaxe du bloc **network**

network < net_name > (< type >)
  aut < aut_name > {[aut_rep]}
    stt < stt_name > {[stt_rep]} {(reward)}
      to(< stt_name > {[stt_rep]} {/to_cond} *)*

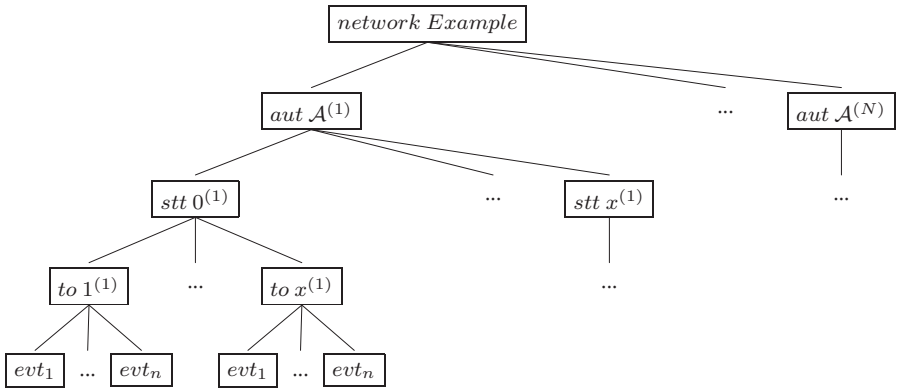


FIG. B.2 – Structure hiérarchique de la partie **network**

< evt_name > {[evt_rep]} {(<prob>)} {/evt_cond}
. . . // *Description des automates*

– “net_name” est le nom du modèle. Le nom du modèle doit être une chaîne de caractères alphanumérique débutant par une lettre ;
– “type” est l’échelle de temps du modèle. L’échelle de temps peut être “*continuous*” ou “*discrete*”. La première option définit un modèle à échelle de temps continue, alors que la deuxième définit un modèle à échelle de temps discret.

Les sous-sections suivantes fournissent plus de détails sur chaque niveau de la hiérarchie.

B.4.1 Description des automates

Dans ce niveau, chaque automate est décrit. La description de chaque automate est délimitée par le mot-clé "*aut*" suivi du nom de l’automate. Éventuellement, un automate peut être répliqué pour créer un certain nombre de copies de l’automate. Dans ce cas, un domaine de réplication doit être précisé. L’identifiant d’un automate répliqué est : si i est un indice qui appartient au domaine de réplication et A le nom de l’automate, alors $A[i]$ est l’identifiant de l’automate.

Syntaxe de la description des automates (*aut*)

aut < aut_name > {[aut_rep]}
stt < stt_name > {[stt_rep]} {(reward)}
to(< stt_name > {[stt_rep]} {/to_cond} *)*
< evt_name > {[evt_rep]} {(<prob>)} {/evt_cond}
. . . // *Description des états*

– "aut_name" est le nom de l'automate. Le nom du modèle doit être une chaîne de caractères alpha-numérique débutant par une lettre ;
– "aut_rep" est le domaine de réplication de l'automate. Le aut_rep doit être un identifiant de domaine déclaré dans le bloc **identifiers**.

B.4.2 Description des états

Le mot-clé "*stt*" annonce la définition d'un état local d'un automate. En ajoutant un domaine de réplication, on peut créer des copies de cet état. Éventuellement, on peut associer une récompense à chaque état. La syntaxe de la description des états est :

Syntaxe de la description des états (*stt*)

stt < stt_name > {[stt_rep]} {(reward)}
  to(< stt_name > {[stt_rep]} {/to_cond} *)*
    < evt_name > {[evt_rep]} {(<prob>)} {/evt_cond}
. . . // *Description des transitions*

– "stt_name" est l'identifiant de l'état. Cet identifiant pourra être utilisé dans l'évaluation des fonctions. PEPS utilise un indice interne pour les états locaux de chaque automate. Le premier état déclaré a l'indice interne 0, le deuxième l'indice 1 et ainsi de suite ;
– "stt_rep" définit le nombre de réplications de chaque état dans l'automate. stt_rep doit être un domaine de réplication déclaré dans le bloc **identifiers** ;
– "reward" est un paramètre optionnel et permet de définir une valeur de récompense différente de la valeur de l'indice de l'état interne attribué automatiquement par PEPS (de 0 à nombre d'états -1).

B.4.3 Description des transitions

La description d'une transition débute par le mot-clé "*to*". L'identifiant de l'état indiqué entre parenthèse définit l'état d'arrivée. "[x]" peut être utilisé pour pointer vers un état spécifique ou un ensemble d'états répliqués. Dans le cas d'une transition vers des états répliqués, 3 situations sont possibles :

– utiliser une valeur constante pour définir un état fixe ;
– utiliser une fonction pour définir un état répliqué, lorsque l'état d'arrivée est défini par rapport à l'indice de l'état courant de départ ;
– utiliser un domaine pour définir des transitions vers plusieurs états.

Dans un groupe d'états répliqués, l'expression d'autres états du groupe peut être faite par des références aux positions : état courant ($==$), précédent ($--$) ou suivant ($++$). Des sauts plus important, par exemple un saut de deux états peuvent également être définis ($+2$), mais il

faut bien noter que toute référence à une position qui pointe vers un état non existant ou un état en dehors du groupe d'états répliqués est ignorée.

Syntaxe de la description des transitions (*to*)

to(< stt_name > {[stt_rep]} {/to_cond} *)*
< evt_name > {[evt_rep]} {(<prob>)} {/evt_cond}
. . . // *Description des événements*

– "to_cond" définit une condition d'inclusion nécessaire à la réplication d'une transition. to_cond est un identifiant de fonction déclarée dans le bloc **identifiers**. Généralement, la valeur de la fonction de condition dépend de l'indice de l'automate et/ou de l'état courant de départ et/ou d'arrivée.

Lorsqu'on définit une condition d'inclusion dans une réplication d'un transition, la transition sera répliqué uniquement sur les états où la fonction est évaluée à une valeur non-nulle.

B.4.4 Description des événements

Enfin, pour chaque transition définie, on associe un événement ou un ensemble d'événements qui peuvent déclencher la transition. Ils sont exprimés par leur nom et, éventuellement par une probabilité de routage. Les événements associés à une transition doivent être déclarés auparavant dans le bloc **events**. Tout comme pour une transition, un événement peut être répliqué. Les 3 possibilités présentées dans la section ci-dessus sont valables pour la réplication des événements. Une condition de réplication peut être aussi associée à la réplication.

Syntaxe de la description des événements

< evt_name > {[evt_rep]} {(<prob>)} {/evt_cond}

– "evt_name" est le nom d'un événement déclaré dans le bloc **events**. On utilise "[x]" après le nom de l'événement pour spécifier un état répliqué ou un ensemble d'événements. On rappelle qu'il est possible d'avoir 3 niveaux de réplication pour les événements. Pour identifier un événement répliqué dans 2 ou 3 niveaux, on doit ajouter les indices des résultats "[x][x]" et "[x][x][x]", respectivement. On peut utiliser une fonction différente pour chaque niveau de réplication ;
– "prob" est la probabilité de routage de l'événement. Cette valeur peut être omise si la probabilité est égale à 1.0. "prob" doit être un identifiant déclaré dans le bloc **identifiers** ;
– "evt_cond" définit une condition d'inclusion nécessaire pour la réplication d'un événement. evt_cond est un identifiant de fonction déclaré dans le bloc **identifiers**. Généralement, la valeur de la fonction de condition dépend de l'automate et/ou de l'état courant de départ et/ou d'arrivée.

B.5 Bloc des résultats

Dans ce dernier bloc, délimité par “**results**”, définit les fonctions utilisées pour calculer les indices de performance du modèle. Les résultats calculés par PEPS sont la moyenne de ces fonctions évaluée pour chaque état global du modèle sur la distribution stationnaire.

Syntaxe du bloc **results**

results
< res_name > = < exp > **;**

– “res_name” est l’identifiant d’un résultat ;
– “exp” est une expression mathématique. Les expressions mathématiques sont décrites dans la Section B.6.

B.6 Fonctions

Cette section présente les possibilités fournies par PEPS2007 pour la construction de fonctions. De façon générale, les fonctions utilisées dans PEPS sont similaire aux expressions mathématiques, avec des opérateurs logiques et arithmétiques. Cependant, les arguments de ces fonctions peuvent être nombres constants, mais aussi des identifiants d’états et d’automates.

Le format des opérateurs est résumé dans TAB. B.1.

Format extern	Sémantique du format	Exemple
Description des opérateurs		
at	donne l'indice de l'automate courant	at
sts	donne l'indice de l'état courant	sts
std	donne l'indice de l'état d'arrivée	std
Opérateurs SAN		
st $< aut\_name >$	donne l'état courant de l'automate "$< aut\_name >$"	st processA
nb $< stt\_name >$	donne le nombre total d'automates dans l'état "$< stt\_name >$"	nb util
nb [$< aut\_name >$ .. $< aut\_name >$**]** $< stt\_name >$	pour les automates dans l'intervalle "[$< aut\_name >$ .. $< aut\_name >$]", donne le nombre total d'automates dans l'état "$< stt\_name >$"	nb [processA .. processD) util
rw $< aut\_name >$	donne la récompense associée à l'état courant de l'automate "$< aut\_name >$"	rw processA
sum_rw [$< aut\_name >$ .. $< aut\_name >$**]**	donne la somme des récompense des états courants des automates dans l'intervalle **[**$< aut\_name >$ .. $< aut\_name >$**]**	sum_rw [processA .. processD]
sum_rw [$< aut\_name >$ .. $< aut\_name >$**]** $< stt\_name >$	donne la somme des récompense des états courants des automates dans l'intervalle "$< aut\_name >$ .. $< aut\_name >$", lesquelles sont dans l'état $< stt\_name >$	sum_rw [processA .. processD] util
prod_rw [$< aut\_name >$ .. $< aut\_name >$**]**	donne le produit des récompense des états courants des automates dans l'intervalle "$< aut\_name >$ .. $< aut\_name >$"	prod_rw [processA .. processD]
prod_rw [$< aut\_name >$ .. $< aut\_name >$**]**$< stt\_name >$	donne la somme des récompense des états courants des automates dans l'intervalle "$< aut\_name >$ .. $< aut\_name >$", lesquelles sont dans l'état $< stt\_name >$	product_rw [processA .. processD] util
Opérateurs arithmétiques		
$< exp1 > + < exp2 >$	somme de "$< exp1 >$" et "$< exp2 >$"	$5 + 3$
$< exp1 > - < exp2 >$	différence de "$< exp2 >$" moins "$< exp1 >$"	$5 - 3$
$< exp1 > * < exp2 >$	produit de "$< exp1 >$" et "$< exp2 >$"	$5 * 3$
$< exp1 > / < exp2 >$	division de "$< exp1 >$" par "$< exp2 >$"	$5 / 3$
$< exp1 > div < exp2 >$	division entière de "$< exp1 >$" par "$< exp2 >$"	$5\ div\ 3$
$< exp1 > mod < exp2 >$	reste de la division entière de "$< exp1 >$" par "$< exp2 >$"	$5\ mod\ 3$
min ($< exp1 >, < exp2 >$**]**	donne le minimum des arguments	min(5,3)
max ($< exp1 >, < exp2 >$**]**	donne le maximum des arguments	max(5,3)
Opérateurs relationnels		
$< exp1 > == < exp2 >$	if "$< exp1 >$" est égal à "$< exp2 >$", donne vrai ("1") sinon faux ("0")	$5 == 3$
$< exp1 > != < exp2 >$	if "$< exp1 >$" n'est pas égal à "$< exp2 >$", donne vrai ("1") sinon faux ("0")	$5 != 3$
$< exp1 > < < exp2 >$	if "$< exp1 >$" est plus petit que "$< exp2 >$", donne vrai ("1") sinon faux ("0")	$5 < 3$
$< exp1 > <= < exp2 >$	if "$< exp1 >$" est plus petit ou égal à "$< exp2 >$", donne vrai ("1") sinon faux ("0")	$5 <= 3$
$< exp1 > > < exp2 >$	if "$< exp1 >$" est plus grand que "$< exp2 >$", donne vrai ("1") sinon faux ("0")	$5 > 3$
$< exp1 > >= < exp2 >$	if "$< exp1 >$" est plus grand ou égal à "$< exp2 >$", donne vrai ("1") sinon faux ("0")	$5 >= 3$
Opérateurs logiques		
$! < exp >$	"not" logique	$5 > 3) ! (7 < 8)$
$< exp1 > \|\| < exp2 >$	"or" logique	$(5 > 3) \|\| (7 < 8)$
$< exp1 > \&\& < exp2 >$	"and" logique	$(5 > 3)\ \&\&\ (7 < 8)$
$< exp1 > \wedge < exp2 >$	"XOR" logique	$(5 > 3) \wedge (7 < 8)$

TAB. B.1 – Opérateurs

Le format interne utilisé dans PEPS est présenté dans la quatrième colonne et il est outil pour le débogage de PEPS. Les opérateurs arithmétiques sont "$+$", "$-$", "$*$", "$/$", "div", "mod", "max" et "min" et ils ne sont pas typés (valeurs entières ou réelles). Les expressions dans PEPS n'ont pas d'ordre de priorité et elles sont évaluées de gauche à droite. Pour spécifier un ordre de priorité, il est nécessaire d'utiliser des parenthèses. Par exemple, $5+6*7$ est calculé comme $(5+6)*7$ dans PEPS.

Les opérateurs relationnels sont : "$==$", "$!=$", "$<$", "$>$", "$<=$" et "$>=$". Leurs résultats sont égal à 1 (pour *vrai*) si la relation est vérifiée et 0 (pour *faux*) dans le cas contraire. Les opérateurs logiques sont : "négation" codé par "!", "où" par "||", "et" par "&&", "XOR" par $\wedge$.

Comme on l'a déjà mentionné, les arguments de ces opérateurs peuvent être des valeurs constantes, mais aussi des fonctions des identifiants du modèle. On peut avoir deux types de fonctions. Le premier type de fonction est appelé "*fonctions de description*" et le deuxième "*fonctions SAN*".

B.6.1 Fonctions de description

Les fonctions de description sont utilisées pour la description du modèle. Ces fonctions ont été conçues pour faciliter la réplication d'états, de transitions et d'événements. Lorsque PEPS2007 génère le Descripteur Markovien (l'ensemble de matrices qui décrivent le modèle), il prend en compte ces fonctions pour créer des répliques et construire le modèle.

PEPS2007 définit 3 fonctions de base :

- **at** : rend l'indice de l'automate courant. Si l'automate est un automate répliqué alors cette fonction rend l'indice interne de la réplication sinon, elle rend l'indice explicite de l'automate dans le modèle ;
- **sts** : rend l'indice de l'état courant dans l'automate courant du modèle. Comme dans la fonction précédente, cette fonction rend l'indice interne de la réplication pour un état répliqué et l'indice explicite de l'état pour un état non répliqué ;
- **std** : cette fonction rend l'état d'arrivée d'une transition. L'indice rendu par cette fonction suit le patron des fonctions précédentes.

B.6.2 Fonctions SAN

Avec l'idée qu'un modèle SAN a un état qui évolue avec le temps, on utilise le terme "état courant". Ces fonctions sont utilisées pour exprimer le comportement d'un modèle SAN tel que "le taux d'un événement est 0 si le modèle est dans l'état x et égal à r dans le cas contraire, où pour calculer des indices de performance par une fonction d'intégration telle que "le nombre d'automates dans l'état 0". Avant de présenter ces fonctions, on décrit comment on gère les identifiants dans PEPS.

- une référence à un identifiant d'un automate est traduit dans PEPS pour une référence à l'indice interne de l'automate. Cet indice interne est calculé selon l'ordre de déclaration dans le bloc **network** . Le premier automate du réseau a l'indice interne 0, le deuxième 1 et ainsi de suite. Les automates répliqués sont indexés en utilisant l'indice interne du premier automate comme base. Par exemple, un modèle SAN avec 3 automates (A, B et C), définis dans cet ordre, ont les indices interne 0, 1 et 2, respectivement.
- une référence à un identifiant d'un état est aussi remplacée dans PEPS par une référence interne. Les identifiants externes correspondent à un indice interne calculé selon l'ordre de déclaration et le nombre de réplications.
- PEPS a un tableau global (pour tous les automates) qui contient les correspondances entre un identifiant externe d'un état et l'indice interne de cet état. Si différents automates ont les mêmes identifiant d'états, il faut remarquer que ces états peuvent ne pas avoir les mêmes indices internes. Cette situation peut entraîner des erreurs si les identifiants de ces états ne correspondent pas aux mêmes indices internes (les indices internes sont calculés par automates). L'utilisateur doit faire attention à cette implantation lorsqu'il écrit une fonction basée sur les identifiants externes, tel que "les nombre d'automates dans l'état $n0$".

Ces fonctions sont :

- **st** $< aut\_name >$: rend l'état courant de l'automate $< aut\_name >$. $< aut\_name >$ est un identifiant externe, et le résultat de cette fonction est l'indice interne de l'état ;
- **nb** $< stt\_name >$: rend le nombre total d'automates du modèle dans l'état $< stt\_name >$. $< stt\_name >$ est un identifiant externe qui PEPS traduit dans un indice interne ;
- **nb [**$< aut\_name > .. < aut\_name >$**]** $< stt\_name >$: est une extension de la fonction précédente, mais on compte le nombre d'automates dans l'état $< stt\_name >$ uniquement dans l'intervalle **[**$< aut\_name > .. < aut\_name >$**]** ;
- **rw** $< aut\_name >$: rend la récompense associé à l'état courant de l'automate $< aut\_name >$. $< aut\_name >$ est un identifiant externe et le résultat est une récompense, tel qu'un nombre entier ou réel ;
- **sum_rw [**$< aut\_name > .. < aut\_name >$**]** : rend la somme des récompenses associées aux états courants de l'intervalle d'automates **[**$aut\_name > .. < aut\_name >$**]** ;
- **sum_rw [**$< aut\_name > .. < aut\_name >$**]** $< stt\_name >$:rend la somme des récompenses associées aux états courants des automates dans l'intervalle **[**$< aut\_name > .. < aut\_name >$**)** qui sont dans l'état $< stt\_name >$;
- **prod_rw [**$< aut\_name > .. < aut\_name >$**]** et
  prod_rw [$< aut\_name > .. < aut\_name >$**]** $< stt\_name >$: sont similaires aux fonctions précédentes, mais pour l'opérateur de multiplication.

Titre : Réseaux d'Automates Stochastiques : Analyse transitoire en temps continu et algèbre tensorielle pour une sémantique en temps discret

Résumé : Cette thèse présente des méthodes et des algorithmes pour l'évaluation de performance de modèles avec très grands espace d'états décrits par des formalismes de haut niveau. Parmi les différents formalismes couramment utilisés, on se place dans le cadre des Réseaux d'Automates Stochastiques (SAN). Le formalisme SAN se caractérise par la représentation de très grands systèmes par la composition de sous-systèmes (automates), où ces automates interagissent entre eux par des événements synchronisants ou des taux et des probabilités fonctionnels.
La première partie de cette thèse s'intéresse au calcul des indices de performances transitoires pour des grands modèles. Lorsqu'on calcule des indices de performances transitoires, tel que la disponibilité ponctuelle, la méthode d'uniformisation est la plus souvent utilisée. Cependant le nombre d'itérations vecteur-matrice peut être très grand ce qui devient critique pour de très grands modèles. Des méthodes de détection du régime stationnaire peuvent réduire le coût de calcul en arrêtant les itérations lorsque le régime stationnaire est atteint. Dans cette thèse, nous proposons une adaptation et une comparaison de différentes méthodes de détection du régime stationnaire lorsque la matrice est stockée sous un format tensoriel. Les méthodes sont comparées selon deux critères : nombre d'itérations et précision des résultats.
Dans la deuxième partie, nous présentons le formalisme SAN à temps discret. La définition formelle du formalisme SAN présentée dans cette thèse nous permet de définir la sémantique des modèles en temps discret que nous souhaitons exploiter. Nous définissons une nouvelle algèbre tensorielle (appelée *Algèbre Tensorielle compleXe - ATX*) capable d'exprimer cette sémantique. Pour cela, trois opérateurs sont définis afin de décrire différents comportement d'un système, tels que la simultanéité, la concurrence et le choix. Enfin, le principal apport de cette thèse réside dans la définition d'une formule tensorielle (appelée *Descripteur discret*) qui utilise cette nouvelle algèbre pour représenter un modèle SAN à temps discret de façon compacte. Nous montrons que ce descripteur discret permet aisément de générer la chaîne de Markov représentée par le modèle SAN.

Mots clés : Évaluation des performances, Réseaux d'automates stochastiques, Méthodes numériques, Algèbre tensoriel, Analyse transitoire, Temps continu et temps discret.

Titre : Stochastic Automata Networks : Transient analysis for continuous time models and tensor algebra for a semantic of discrete time models

Abstract : This thesis presents methods and algorithms for the performance evaluation of large state space models described by high-level formalisms. Among the various formalisms commonly used we use Stochastic Automata Networks (SAN) formalism. SAN formalism is characterized by the representation of very large systems by the composition of its subsystems (automata), where these automata interact with each other by synchronizing events or rates and functional probabilities.
The first part of this thesis focuses on the computation of transient performance indices for large models. When we compute transient indices, such as point availability, the uniformisation method is often used. However, the number of iterations (vector-matrix multiplication) can be very large, which is critical for very large models. Stationnarity detection methods can reduce the computational cost by stopping the iterations when the steady state is reached. In this thesis, we propose an adaptation and a comparison of the different stationnarity detection methods when the matrix is stored in a tensor format. The methods are compared using two criteria : number of iterations and results accuracy.
In the second part, we present the SAN formalism for discrete time models. The SAN formalism formal definition presented in this thesis allows us to define the semantic of discrete time models that we wish to explore. We define a new tensor algebra (called *Complex Tensor Algebra - XTA*) capable of expressing this semantic. For that uses, three operators are defined to describe the different behavior of a system, such as simultaneity, competition and choice. Finally, the main contribution of this thesis lies in the definition of a tensor formula (called *discrete descriptor*) that uses this new algebra to describe a discrete time SAN model in a compact mode. We show that this discrete descriptor can easily generate the Markov chain represented by the SAN model.

Keywords : Performance evaluation, Stochastic automata networks, Numerical methods, Tensor algebra, Transient analysis, Continuous and discrete time models.